

Practical Examples for Becoming Fast and Productive in Vim Editor



Ramesh Natarajan
www.thegeekstuff.com

目 录

介绍	1
关于作者	2
版本和声明	3
前言	4
版本	5
第一章 Vim 基础.....	6
打开一个文件	6
保存一个文件	6
关闭文件	6
Vim 模式类型.....	7
基本模式	7
高级模式	7
在一个文件中移动	8
Vim 配置文件（~/vimrc）	8
本地 Vimrc.....	8
全局的 Vimrc.....	9
VIM 版本	9
Vim 安装.....	10
关于 Vim 帮助全部.....	10
使用 Vim 教程实践.....	11
第二章 基本的导航	12
技巧 1：滚动整屏或半屏	12
技巧 2：词导航	12
技巧 3：在一行中特殊位置上的位置光标	13
技巧 4：段落、章节和句子导航	13
第三章 高级导航	15
技巧 5：屏幕导航	15
技巧 6：重画在顶部、底部或者中间的当前行的屏幕	15
技巧 7：导航到文件的开始和结束	15

技巧 8: 导航到文件的第 N 个字符或百分之 N 处	15
技巧 9: 行号导航	16
技巧 10: 源代码导航	16
技巧 11: 从插入模式下导航	17
第四章 专家导航	18
技巧 12: 使用 CTRL-O 和 CTRL-I 跳转	18
技巧 13: 在非常长的行中导航	18
技巧 14: Vim 命令行导航	19
技巧 15: 使用标签来创建本地书签	19
技巧 16: 在 Vim 文件中创建全局书签	20
技巧 17: 如何显示所有书签	21
技巧 18: 导航使用 ctags 的任何源代码	22
技巧 19: 为任何编程语言转换 Vim 编辑器未漂亮的源代码浏览器	23
第五章 基本文本操作	27
技巧 20: 插入或者添加文本	27
技巧 21: 代替文本	27
技巧 22: 替换文本	28
技巧 23: 改变文本	28
技巧 24: 使用非合并空格选项合并多行	28
第六章 高级文本操作	30
技巧 25: 拷贝一个字符、单词、行或者到指定位置	30
技巧 26: 在拷贝行/词/其它之后或之前粘贴	30
技巧 27: 删除单个字符、词或者行	31
技巧 28: 从剪切板缓冲区插入内容	31
技巧 29: 从文件中插入内容到剪切板	31
技巧 30: 写文件的部分内容到另一个文件	31
技巧 31: 交换相邻的字符	32
技巧 32: .(点)命令的强大功能	32
技巧 33: 可视化模式命令	32
技巧 34: 使用:g 编辑	34

第七章 专家级文本操作	35
技巧 35: 拷贝多行到命名缓冲区随后使用	35
技巧 36: 转换插入文本到正常模式的命令	35
技巧 37: 简写和全写	35
技巧 38: 自动拼写更正	36
技巧 39: 记录和使用宏	37
技巧 40: 排序文件内容	39
技巧 41: 恢复删除的文本	40
技巧 42: 给文件使用添加自动头部	40
第八章 Vim 作为程序员编辑器	44
技巧 43: 让 Vim 智能的高亮你的代码	44
技巧 44: 智能格式缩进	44
技巧 45: 从 Vim 中访问 Unix 的函数帮助页	45
技巧 46: 跳到变量声明处	45
技巧 47: 对齐变量的赋值语句	46
技巧 48: 使用 CTRL 键来增加和减少数量	46
技巧 49: 在插入模式下执行一个 Vim 命令	47
第九章 Vim 命令行技巧	51
技巧 58: 在只读模式下打开文件	51
技巧 59: 简单地覆盖 Swap 文件	51
技巧 60: 在打开文件时执行任意 Vim 命令	52
第十章 gVim 技巧	53
技巧 64: 显示和隐藏 gVim 菜单和工具栏	53
技巧 65: 添加用户菜单或者菜单项到 gVim	53
技巧 66: gVim 中改变字体	54
第十一章 Vim 外观、标签和窗口	56
技巧 67: 水平和垂直分割窗口	56
技巧 68: 改变窗口标题	57
技巧 69: 改变 Vim 颜色	57
技巧 70: 在标签中编辑多个文件	58

第十二章 Vim 编辑器的其它特征.....	60
技巧 71: 重复一个操作 N 次	60
技巧 72: 撤销和重做操作	60
技巧 73: 打开在光标下面的文件	61
技巧 74: 使用传统的方式编辑多个文件	61
技巧 75: 自动保存文件	62
技巧 76: 在 Vim 加密文件	62
技巧 77: 存储和恢复 Vim 会话	62
技巧 78: Vim 中执行 Unix Shell 命令	63
技巧 79: 使用 vimdiff 查看文件之间的不同	64
技巧 80: Vim 的 map (映射) 命令	64
技巧 81: 使得 Bash 脚本如 Vim 编辑器一样工作	65
技巧 82: 设置 Vim 选项	66
技巧 83: 不设置 (取消) Vim 选项	66
技巧 84: 缺省寄存器及其使用	66
技巧 85: 数字寄存器和恢复删除	67
技巧 86: Vim 目录操作	67
第十三章 搜索的力量	69
技巧 87: 利用搜索导航	69
技巧 88: 移动到当前词的下一个/前一个出现位置	69
技巧 89: 在一行中搜索一个字符	70
技巧 90: 12 个有用的查找和替换例子	70
技巧 91: 使用 vimgrep 在多个文件中进行搜索	74
技巧 92: 颜色高亮搜索结果	74
技巧 93: Vim 增量搜索	75
技巧 94: :match 的力量	75
第十四章 自动完成	77
技巧 95: 自动补全单词	77
技巧 96: 自动行补全	77
技巧 97: 自动文件名补全	78

技巧 98: 字典补全	78
技巧 99: 同义词字典单词补全	79
技巧 100: 自动打开一个弹出菜单用于补全	80
技巧 101: 输入时自动提供单词补全	82
第十五章 额外技巧	84
额外技巧 1: 为项目的列表添加项目符号风格	84
额外技巧 2: 设置 Vim 作为通用缺省编辑器	85
额外技巧 3: 是的 Vim 作为缺省编辑器	85
额外技巧 4: 格式化段落	85
额外技巧 5: 编辑可重用的宏	86
额外技巧 6: 缩进代码块	86
额外技巧 7: 合并的力量	86
额外技巧 8: 标识文件的改变	87
额外技巧 9: 刷新屏幕	87
额外技巧 10: 插入非键盘字符	87
额外技巧 11: Vim ex 模式	88
额外技巧 12: 放置光标在匹配的最后	88
额外技巧 13: 查看字符的 ASCII 值	88
额外技巧 14: 在 Vim 编辑器中编辑二进制文件	88
额外技巧 15: 换行一只查看代码要求的部分	89
反馈和支持	90

介绍

生产力现在能够完成你以前不能做的事情。

--Franz Kafka--

如果你花费大量的时间在 **Unix** 或 **Linux** 环境下，你必须经常使用 **Vi/Vim** 编辑器。掌握 **Vim** 编辑器的基础知识，并且知道如何更有效的使用它，这会立即提高你的生产力。

本书包括 **101** 个 **Vim** 的技巧（例子），在使用 **Vim** 编辑器时，它们将帮助你变得更快和更有生产力。

本书中的所有技巧都通过适当的 **Vim** 编辑器命令例子进行说明，它们很简单，也非常容易执行。

本书包含 **15** 章内容。

- 第 1 章向新手讲解 **Vim** 编辑器的基础知识。
- 第 2 到 14 章 包括所有 **101** 个技巧。
- 第 15 章包括其它额外的技巧，在本书随后的版本中，我们会把更多的技巧添加到这个章节中。

在本书中使用的约定：

- **CTRL-A** — 同时按下 **CTRL** 和 **A** 键。
- **10j** — 在正常模式下连续输入这些字符。
- **:set nu** — 在命令行模式下输入这个命令。

关于作者



我叫 Ramesh Natarajan，是 Geek Stuff 博客 thegeekstuff.com 和本书的作者。

我已经使用几种语言完成大量的编程，C 是我最喜欢的语言。我做有许多基本架构方面的工作，包括 Linux 系统管理员、DBA、网络、硬件和存储（EMC）。

我也已经开发 passworlddragon.com — 一个免费、易于使用和安全的密码管理器，可以运行在 Windows、Linux 和 MAC 上。

我也是免费 Linux 101 Hacks 电子书籍的作者。

<http://www.thegeekstuff.com/linux-101-hacks-free-ebook/>

如果你对本电子书有任何的反馈意见，请使用下面的联系方式与我联系。

<http://www.thegeekstuff.com/contact>

版权和声明

版本（c）2009 – Ramesh Natarajan，版权所有。

本书中所有部分都不得以任何形式和任何方式进行复制、翻译、发布或者共享。

在本书中提供的信息与隐含的保证或者担保一起提供。

备注：感谢购买者 *XiuJuan Lu*(luxiujuan@gmail.com) 的贡献，本书的翻译版本仅供学习使用，不能用于任何商业用途，版权归作者所有。

前言

这里有许多编辑器，大部分编辑器仅提供适中的功能和少量的易用性。没错，这些工具都有它们的使用人群，但是专业的用户需要专业的工具。你已经选择了 Vim，这是一个好的选择。

当系统的资源受限时，Vim 诞生了。虽然这些时候已经过去，但是产生了一个更稳定的编辑器，它可以运行在各种平台上，并且有一个杰出的命令行概念。当然它是编程人员可以使用的最好编辑器之一。如果你使用它，你可以获得不可思议的生产力水平。Vim 可以提供给你所期望的编辑器的一切。它包含对宏、插件和命令行工具的处理。

掌握一个高级的编辑器不是一个小额投资。然而，在许多年实践之后，我可以确信你的努力会得到巨大的回报。请不要忘记，在这个过程中学习如何运用它的强大功能会获得很多的乐趣。

最好的情况是一个有经验的使用者给你指导和陪伴你学习。本书就是为此而著的。学习曲线非常的陡峭。在你掌握了基础知识之后，你会一步步学习更多高级的技巧。为了让自己变成一个专家，你只需要做三件事情：实践、实践、再实践。在几周或者几个月里本书会指导你学习。

现在有许多工作要做，但是你经历一个非常有兴趣的时间，并且从中获得很多。享受使用 Vim 的高效，让我们立即开始。

--Prof. Dr. Fritz Mehner, Fh Sudwestfalen 德国

（几个 [Vim 插件](#)的作者，包括 [bash 支持的 Vim 插件](#)）

版本

版本	日期	修订
1.0	2009 年 10 月 21 日	第一版

第一章 Vim 基础

在开始学习 101 个技巧之前，让我们了解一些 Vim 编辑器的基础知识。

打开一个文件

这里有两种打开一个文件的方法，下面的例子将会打开 `/etc/passwd` 文件。

方法 1: 从命令行中打开文件，正如上面讲解的。

```
$ vim /etc/passwd
```

方法 2: 在运行 Vim 之后，从 Vim 编辑器中打开文件。

```
$ vim
```

```
:e /etc/passwd
```

保存一个文件

下面是保存一个文件的方法。

保存方法	描述
<code>:w</code> 或 <code>:write</code>	保存工作文件
<code>:up</code> 或 <code>:update</code>	保存工作文件
<code>:w newfile.txt</code>	另存为 newfile.txt
<code>:up newfile.txt</code>	另存为 newfile.txt
<code>:w! newfile.txt</code>	另存为 newfile.txt(带有覆写选项)
<code>:up! newfile.txt</code>	另存为 newfile.txt(带有覆写选项)

关闭文件

下面是一些用于关闭一个文件并且退出 Vim 编辑器的方法。

退出方法	描述
<code>:x</code>	保存工作文件并退出
<code>:wq</code>	保存工作文件并退出
<code>ZZ</code>	保存工作文件并退出
<code>:q!</code>	不保存工作文件并退出
<code>:qa</code>	退出当前 Vim 会话中所有打开的文件

Vim 模式的类型

在 Vim 中有几种模式。为了容易理解，让我们把它们分为两类—基本模式和高级模式。

基本模式

为了更有效的使用 Vim 编辑器，绝对有必要知道这三种基本模式。

模式	描述
Normal（正常）	Vim 编辑器在这种模式下开始，你可以执行所有编辑器的命令。
Insert（插入）	该模式为了插入文本
Command Line（命令行）	该模式为了在编辑器的底部执行外部命令，例如， <code>:wq</code>

假设你希望创建一个带有“Hello World!”的 `helloworld.txt` 文件，下面步骤讲解如何使用这三种模式来完成它。

第 1 步：正常模式。在正常模式下打开新的文件。

```
$ vim helloworld.txt
```

第 2 步：插入模式。进入插入模式，然后输入 Hello World!

```
i
```

第 3 步：命令行模式。进入命令行模式，输入 `:wq` 保存文件并且退出编辑器。

```
<ESC> :wq
```

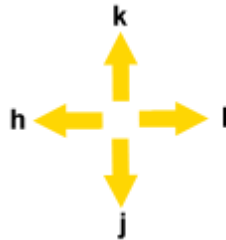
高级模式

对于一般使用而言，这些模式不是必须的，但是这有助于了解所有 Vim 的可用的模式。

模式	描述
Visual（可视化）	在可视化模式下你可以选择文本（使用 <code>v</code> ， <code>V</code> 或者 <code>CTRL-V</code> ）并且执行 Vim 命令。 例如：在可视化模式下，你可以选择文本中一行并且删除掉。
Select（选择）	摘自于:help vim-modes-intro “入一个可打印的字符删除选择并且开始插入模式。在这种模式下， “—SELECT—”显示在窗口的底部。”
Ex（额外）	摘自于:help vim-modes-intro “像命令行模式一样，但是在输入一个命令之后，你保留在 Ex 模式下。”

在一个文件中移动

在许多应用中，你可以使用向上、向下、向左和向右的箭头键来完成基本的导航。在 Vim 编辑器中，下面是一些基本的导航键：



图：基本导航键

导航键	描述
j	向下一行
k	向上一行
h	向右移动一个字符
l (小写的 L)	向左移动一个字符

一个历史注释：在 ADM-3A 终端中，h, j, k 和 l 键上有箭头，它通常在早期的 Unix 系统中使用。这是为什么使用这些键选择作为导航键。

注意：如果你不熟悉 j, k, h 和 l 键，你仍然可以使用箭头（方向）键来移动光标。

Vim 配置文件（~/.vimrc）

本地 Vimrc

你定义在 Vim 中的所有配置选项只对专门的 Vim 会话有用。

例如，如果你在 Vim 中输入 `:set number` 来显示行号，只有专门的 Vim 会话才会应用。如果你退出并且开始 Vim 编辑器，显示的行号不会出现。

如果希望让你的配置设置对以后的 Vim 会话永久有效，你应该把它添加到 ~/.vimrc 文件中，如下所示。

```
$ vim ~/.vimrc
```

```
set number
set list
```

本地 vimrc 文件的位置:

操作系统	位置
UNIX/Linux	<code>\$HOME/.vimrc</code> 例如: <code>/home/Ramesh/.vimrc</code> 注意: 在 Unix 下这里再 <code>vimrc</code> 前面有一个.(点号), 它是一个隐藏文件。
Windows	<code>\$HOME/_vimrc</code> 例如: <code>C:\Documents and Settings\ramesh_vimrc</code> 注意: 在 Windows 下, 这里再 <code>vimrc</code> 前面有一个_(下划线)。

全局的 Vimrc

全局 vimrc 用于系统管理员添加系统范围上的 Vim 配置选项, 它对于系统中的所有用户都有效。通常, 你只应该修改本地的 vimrc 文件。

全局 vimrc 文件的位置:

操作系统	位置
UNIX/Linux	<code>\$VIM/.vimrc</code> 例如: <code>/usr/share/vim/.vimrc</code>
Windows	<code>\$VIM/_vimrc</code> 例如: <code>C:\Program Files\Vim_vimrc</code>

VIM 版本

在本书第一次发行时, Vim 最新的稳定发行版是 7.2。在本书中所有的技巧都在最新稳定版本下测试通过。

从 Vim 编辑器中执行:version 可以查看你的 Vim 编辑器的版本号。

Ubuntu 上的 Vim 版本:

```
$ vim
```

```
:version
```

在 Windows 上 Vim 版本:

```
C:\> vim
```

```
:version
```

Vim 安装

Vim 几乎是所有 Unix 发行版中的缺省编辑器。如果你的操作系统上没有最新版本的 Vim 编辑器，使用下面的指令可以安装它。

在 Windows 下安装 Vim:

进入 vim.org->Download->PC:MS-DOS and MS-Windows->Self installing executable->gvim72.exe。

直接下载链接: <ftp://ftp.vim.org/pub/vim/pc/gvim72.exe>。

下载 gvim72.exe 并且安装它。

在 Ubuntu Linux 下安装 Vim:

```
$ sudo apt-get install vim-full
```

关于 Vim 帮助全部

输入:help 查看 Vim 编辑器自带的帮助文档。

```
$ vim
:help
```

当浏览 Vim 帮助文档时记住下面几点:

- 在||中的任何东西都是链接。
- 移动你的光标到||之间的任何字符上，点击 CTRL-]跳转到指定的帮助章节上。
- 例如: |quickref| 是一个链接。

帮助	描述
:help (或) :h	Vim 内建的帮助文档
:helpgrep pattern	使用模式搜索帮助 例如: :helpgrep saveas 提示: 使用:cn 跳转到下一个模式出现的地方。
:help 'option'	Vim 设置选项的帮助 例如: :help 'list'会给出关于:set list 的帮助信息。
:help CTRL-X	Vim CTRL-X 命令的帮助 使用相同的方法获得其它 CTRL-Vim 命令的帮助。
:help :x	Vim :x 命令的帮助 使用相同的方法获得其它:Vim 命令的帮助信息。
:help <CTRL-D>	自动完成的帮助。

例如: `:help <CTRL-D>`会显示帮助开始的所有命令。

使用 Vim 教程实践

Vimtutor 程序有一个自带的教程，它包括学习 Vim 编辑器的循序渐进的指令。

当你运行 `vimtutor` 程序时，它拷贝一个原始教程文件并且自动打开它。你可以修改这个文件并且当你希望的时候运行。

```
$ vimtutor
```

缺省情况下，`vimtutor` 打开英文的教程文件。为了打开一个指定语言的教程文件，在最后给一个语言的编码。

例如，下面使用西班牙语打开教程文件。

```
$ vimtutor es
```

第二章 基本的导航

这里使用三章用来说明导航—基础、高级和专家级导航。

如果你只是使用 h, j, k, l 字符来导航，你不久会认识到这是一个痛苦而费时的操作。

在导航章节中的技巧将帮助你：在很少击打键盘情况下非常高效的导航文件内容。

技巧1：滚动整屏或半屏

在大文件中，使用 j, k, h 和 l 键来滚动页面是非常低效的。

请使用下面的页导航按键：

导航键	描述
CTRL-F	向下滚动整页（Front）
CTRL-B	向上滚动整页（Bottom）
CTRL-D	向下滚动半页（Down）
CTRL-U	向上滚动半页（Up）

替代使用 j 和 k 键，你也可以使用 CTRL 键一次滚动一行，下面将会说明。

使用 j, k 键和 CTRL 键之间有一些视觉上的不同，你可以试着自己观察下有什么不一样。

导航键	描述
CTRL-E	向下移动一行
CTRL-Y	向上移动一行

技巧2：词导航

使用 h 和 l 键水平导航时非常痛苦和费时的。

你可以使用词导航键更有效的导航单词，如下所示：

导航键	描述
w	移动到下一个单词的开始
W	移动到下一个 WORD 的开始
e	移动到当前单词的结尾
E	移动到当前单词的结尾
b	移动到前一个词的开始
B	移动到前一个词的开始

单词(word)和 WORD

单词包括字母序列、数值和下划线，WORD 包括非空字符的序列，使用空格分隔。

- 例如：192.168.1.3 包括 7 个 word，但是 192.168.1.3 看作是一个 WORD。
- 如果你在“192.168.168.1.3 devserver”开始，并且按下 w（到下一个 word），你会移动到底一个.(点号)，因为 192 可以作为一个 word。
- 如果你在“192.168.168.1.3 devserver”开始，并且按下 W（移动到下一个 WORD），你会移动到“devserver”的 d 处，因为整个 192.168.1.3 看作为一个 WORD。

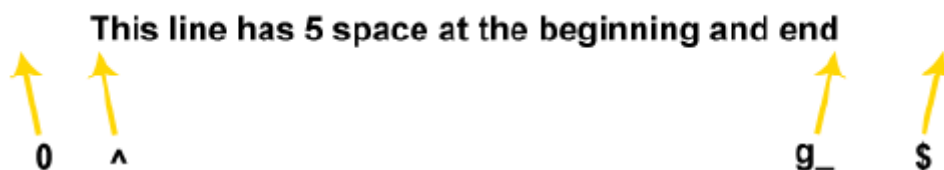


图：word 和 WORD 区别

技巧3：在一行中特殊位置上的位置光标

替代只使用 l 和 h 来在一行中导航，你可以使用下面同一行中不同位置上的位置光标。

导航键	描述
0 (zero)	移动到当前行的开始位置
\$(美元符号)	移动到当前行的结束位置
^(...)	移动到当前行的第一个非空字符
g_	移动到当前行的最后一个非空字符



图：行开始和结束的导航

技巧4：段落、章节和句子导航

使用下面的按键进行段落、章节和句子导航。

导航键	描述
{	移动到当前段落的开始
}	移动到下一个段落的开始
[[移动到当前章节的开始
]]	移动到下一个章节的开始

(移动到前一个句子的开始
)	移动到下一个句子的开始

第三章 高级导航

技巧 5：屏幕导航

移动光标到屏幕的顶部、中间和底部，如下说明：

导航键	描述
H	移动到当前屏幕的第一行 H 的含义是“home”的位置，“0，0”表示屏幕的左上角。
M	移动到当前屏幕的中间行
L	移动到当前屏幕的最后一行

技巧 6：重画在顶部、底部或者中间的当前行的屏幕

你可以重画在顶部、底部和中间的光标位置的当前行的屏幕，如下所示：

导航键	描述
z<ENTER>	重画屏幕顶部光标位置的当前行的屏幕
z- (小写 z 和减号)	重画屏幕底部光标位置的当前行的屏幕
z.(小写 z 和点号)	重画屏幕中间光标位置的当前行的屏幕

技巧 7：导航到文件的开始和结束

你可以快速的移动到文件的开始和结尾，如下所示：

导航键	描述
:0	移动到文件的顶部-方法 1
gg	移动到文件的顶部-方法 2
1G	移动到文件的顶部-方法 3
:\$	移动到文件的底部-方法 1
G	移动到文件的底部-方法 2

技巧 8：导航到文件的第 N 个字符或百分之 N 处

导航键	描述
50%	移动到文件的 50%处，跳转到文件的中间
75%	移动到文件的 75%处，跳转到文件的 3/4 处
100l	导航键是 100 后面跟 l，从当前位置移动 100 字符
100<space>	导航键是 100 后面跟空格

	另外一种方法从当前位置移动 100 个字符
<code>:goto 25</code>	移动到文件开始位置的第 25 个字符处
<code>25 </code>	导航键是 25 后面跟管道符，移动到当前行中的 25 个字符处

技巧 9：行号导航

下面的命令用于在 Vim 编辑器中设置行号。

命令	描述
<code>:set number</code> <code>:set nu</code>	显示行号
<code>:set nonumber</code> <code>:set nonu</code>	不显示行号
<code>:set numberwidth=5</code>	缺省的行号宽度为 4 个字符，使用 <code>numberwidth</code> 可以改变为 5 个字符。

你能跳到指定的行号位置，如下说明：

导航键	描述
<code>:50</code>	移动到第 50 行
<code>50gg</code>	另一种跳转到第 50 行的方法
<code>50G</code>	另一种跳转到第 50 行的方法

技巧 10：源代码导航

这些键对于使用 Vim 的编程人员或者编写 Shell 脚本的系统管理员来说非常有用。

对于普通的 Unix 用户，当浏览任何源代码时非常的易用。

导航键	描述
<code>%</code>	移动到匹配的字符对处 跳转到匹配的括号(), 或者大括号{}或方括号[]处。

当你调试代码或者缺少匹配的括号时，使用下面的快捷方式来补救。

导航键	描述
<code>[(</code>	移动到前一个不匹配的左括号(
<code>)</code>	移动到前一个不匹配的右括号)
<code>[{</code>	移动到前一个不匹配的左大括号(
<code>}</code>	移动到前一个不匹配的右大括号)

技巧11：从插入模式下导航

在正常模式下你使用 **w** 或者 **W** 来进行单词导航。然而如果你希望从插入模式下导航，因此你需要按下 **Shift** 和右箭头。

如果你在插入模式下，并且你认识到你必须导航到下一个单词，然后输入新的文本，你不需要按 **<ESC>w** 跳转到下一个单词，然后按 **i** 再次进入到插入模式。

替代的是，使用从插入模式下使用这些导航键来移动单词：

注：Windows 下的 **gvim** 这些键是文本选择键!!!

导航键	描述
SHIFT-<Right Arrow>	在插入模式下逐个单词的向右移动
SHIFT-<Left Arrow>	在插入模式下逐个单词的向左移动

第四章 专家导航

技巧 12：使用 CTRL-O 和 CTRL-I 跳转

Vim 使用跳转列表来跟踪你的导航，你可以通过这个列表来向前或者向后导航。

跳转列表保留所有地方的轨迹，它可以跟踪文件名、行号和列号。

查看调整列表：

```
:jumps
```

导航键	描述
CTRL-O	跳转到上一个位置点
CTRL-I	跳转到下一个位置点
5CTRL-O	跳转到显示在位置 0 上面的位置 5
5CTRL-I	跳转到显示在位置 0 下面的位置 5

让我们假设当前你编译一个 names.txt 文件，如下所示：

```
vim names.txt
:jumps
jump line col file/text
3 484 19 /home/ramesh/scsi-list.txt
2 5 0 /etc/passwd
1 6 19 /etc.y.p.conf
> 0 16 51 John Smith
1 10 7 /etc/sudoers
2 4 3 /etc/group
3 204 3 /home/ramesh/my-projects.txt
```

- 在本例子中，当编辑 names.txt 时候执行:jumps 命令
- 当前的位置将使用最前面的 location 0 和>来标识
- 在这个例子中，在当前文件 names.txt 中当前的位置 location0 是">0 16 61 Jonh Smth"
- 为了跳转到/etc/password，它位于当前位置上面的 location2，按 2CTRL-O
- 为了跳转到/etc/group，他位于当前位置下面的 location 2，按 2CTRL-I。

技巧 13：在非常长的行中导航

当你遇到非常长的行（没有任何新的行）时，Vim 对待它作为单个行。因此，当你对此一行输入 j 键后，它将跳到下一行上。然而，你会觉得它跳过了好多行。但实际上它只是跳

过一个长行。

可视化行：让我们假设有一个非常长的行，它回绕成 5 个可视化行。为了讨论的目的，让我们称每一单独的行为可视化行。

下面的快捷方式可以帮我们有效地导航一个非常长的行。

导航键	描述
gj	向下滚动一个可视化行
gk	向上滚动一个可视化行
g^	移动到当前可视化行的开始位置
g\$	移动到当前可视化行的结束位置
gm	移动到当前可视化行的中间位置

技巧 14: Vim 命令行导航

当从命令行打开一个文件，你可以通过指定命令行参数来导航到一个特殊的位置，如下：

导航键	描述
\$ vim +142 <filename>	打开文件到 143 行
\$ vim +/search-term <filename>	打开文件移动到向下搜索到指定词语的位置
\$ vim +?search-term <filename>	打开文件移动到向上搜索到指定词语的位置
\$ vim -t TAG <filename>	移动到指定的 TAG 处

例如，如果你打开/etc/passwd 编辑用户 jsmith，你可以这样做。打开文件/etc/passwd 并且直接跳到 jsmith 处。

```
$ vim +/^jsmith /etc/passwd
```

技巧 15: 使用标签来创建本地书签

这里有两种类型的书签：局部书签和全局书签。本技巧中让我们学习下局部书签：

标签命令	描述
ma	在当前位置处创建一个名为“a”的标签
`a(反引号 a)	跳转到书签“a”的精确位置
'a(单引号 a)	跳转到包含标签“a”哪行的开始

在单个文件中，当你希望跳转到特殊位置或者行商，你可以使用本地标签。如果你的标签名称是小写字符，那么它是个局部标签。

输入 m{mark-name}，其中 mark-name 是一个单个的字母表字符，称为书签名称。

m{makr-name}

如何在 Vim 编辑器中创建书签

如果你输入 **ma**，它在当前位置的当前行上创建一个名为 **a** 的书签。在下面的例子中，输入 **ma** 在精确的位置上创建书签，哪里光标会高亮显示。

注意：Vim 与 Vi 不同的是在编辑器推出之后该标签还是存在的，这是一个让许多 UNIX 用户吃惊的强大特征。

访问书签的方法 1: ``{mark-name}`

反引号后面跟着书签名称会移动到精确的书签位置。它回跳转到行中精确的字符位置（先前创建标签的位置）。

例如，如果你输入 ``a`，这会返回到名称 **a** 的书签位置。它会返回到上图中高亮的光标所在位置。

访问书签的方法 2: `'{mark-name}`

单引号后面跟着书签名称。移动到书签所在行的开始位置。

例如，如果你输入 `'a`，它回返回到书签名为 **a** 所在行的起始位置。在上图中它会反馈到“CustonLog logs/access_log combined”行的开始位置。

`'a`

技巧 16: 在 Vim 文件中创建全局书签

当你有多个文件打开时，如果你希望跳转到打开文件中特殊的位置，那么你可以使用 vim 中的全局书签。如果书签名称为大写字母，那么它就是一个全局书签。

下面的几步将解释在编辑多个文件时如何使用全局书签。

- 1) 打开多个文件: `/etc/passwd /etc/group`
- 2) 当编辑 `/etc/passwd` 时移动到指定行，并且输入 `mP` 来创建全局书签 **P**；
- 3) 输入 `:n` 从 `/etc/passwd` 文件跳转到 `/etc/group` 文件；
- 4) 当你编辑 `/etc/group` 文件时移动到指定的行，输入 `mG` 创建一个全局书签 **G**；
- 5) 输入 ``P`（反引号后面跟着大写的 **P**），它回带你到 `/etc/passwd` 的书签位置；
- 6) 从 `/etc/passwd` 中，输入 ``G`（反引号+大写的 **G**），它会返回到 `/etc/group` 中书签处。

读写练习：使用至少两个文件，在两个文件中设置全局书签，然后修改当前的文件，跳转到另外一个没有保存的文件。Vim 做了什么呢？如果设置自动重写（`:set autowrite`），那么事情会变成怎么样？

技巧17：如何显示所有书签

如果你创建几个书签，忘记了它们的名称，你可以很容易的获得书签列表，输入:marks，显示如下：

```
:marks
```

这些说明创建下面一些书签：

- a – 局部书签 a，位于第 15 行的 9 列，它也显示为文本 line15。它来自当前打开的文件 yp.conf。
- b – 局部书签 b，位于第 11 行的 18 列，它也显示为 line18。它来自于当前打开的文件 yp.conf。
- G – 全局书签 G，位于 gourp 文件的第 56 行的 0 列。
- P – 全局书签 P，位于 passwd 文件的第 49 行的 0 列。

除了上面的书签以外，在 Vim 内部任何时候输入:marks，你可以获得下面几行。这些标识 ‘单引号，”双引号，[,], ^和.点号由 Vim 创建和管理。你不需要直接控制它们。

```
:marks
```

你可以使用上面显示的缺省标识，如下：

缺省标识	描述
``	到退出之前最后一次编辑的位置
\[到先前改变或者复制文本的第一个字符
\]	到先前改变或复制文本的最后一个字符
'<	到先前选择可视化区域的第一行
'>	到先前选择可视化区域的最后一行
‘	到最后一次该标的位置
^	到最后一次插入模式停止的光标所在位置

Vim 书签命令的快速总结：

- ma – 创建一个书签 a
- `a – 跳转到书签 a 的精确位置（行和列）
- 'a – 调整到书签 a 所在行的起始位置
- :marks — 显示所有的书签；
- :marks a – 显示名称为 a 书签的详细信息；
- `.- 跳转到最后一次执行改变的精确位置（行和列）。
- ‘.- 跳转到最后一次执行改变的行起始位置。

技巧 18：使用 *ctags* 导航任何源代码

安装 *ctags* 包

```
apt-get install exuberant-ctags  
或  
rpm -ivh ctags-5.5.4.1.i386.rpm
```

生成源代码的 *ctags*（标签）

进入到你源代码所在的目录，在下面的例子中，我把所有的 C 程序代码保存在~/src 目录下。

```
cd ~/src  
ctags *.c
```

Ctags 命令会创建一个名为 `tags` 的文件，它包括关于*.c 程序的信息（tags）。下面是一个 ctags 文件的实际内容。

```
cat tags
```

用途 1：根据指定的函数名称使用：`ta` 导航特殊的函数定义

在下面的例子中，`:ta main` 会带你到 `mycpgrogram.c` 文件中 `main` 函数定义的位置。

```
vim mycpgrogram.c  
:ta main
```

通过使用这个功能，你可以导航执行函数名称定义的任何函数。

用途 2：使用 `CTRL+]` 导航到“函数调用”的函数定义地方

当光标位于函数调用上，这是按下 `CTRL+]` 可以跳转到函数定义处。

下面的例子中，当光标位于 `ssh_xcalloc` 词组处，按下 `CTRL+]` 会跳转到 `ssh_xcalloc` 函数定义处。

```
vi mycpgrogram.c  
av = ssh_xcalloc(argc, sizeof(char*));
```

注意：如果 ctags 不能找到这个函数，你在 vim 状态栏上获得下面的消息：`E246 tag not found ssh_xcalloc`。

用途 3：使用 `CTRL+T` 返回到函数的调用者处

在使用 CTRL+J 跳转到函数定义之后，你可以按下 CTRL+T，它回返回到函数调用处。

用途 4：导航带有相似名称的函数列表

在这个例子中，:ta 会获得第一个函数定义，该函数名以 get 开始。Vim 也创建了所有以 get 开头的函数列表，通过它进行导航。

```
vim mycprogram.c
:ta /^get
```

下面的 Vim 命令可以用于导航匹配的 tag 列表。

Vim 命令	描述
:ts	显示标签列表
:tn	移动到列表中的下一个标签
:tp	移动到列表中上一个标签
:tf	移动到列表中第一个标签
:tl	移动到列表中最后一个标签

技巧 19：为任何编程语言把 Vim 编辑器变为漂亮的源代码浏览器

使用 tags 导航源代码是快速的和功能强大的，但是没有非常可视化的外观。如果你希望一役中类似于文件浏览器的方式导航源代码，你可以使用 Vim 的 taglist 插件来让 Vim 变为源代码浏览器。

Vim 的 taglist 插件作者 Yegappan Lakshmanan 对于插件的描述如下：

“TagList” 插件是一个 Vim 的源代码浏览器插件，提供源代码文件的结构概要，允许你高效地浏览不同编程语言的源代码文件。

安装和配置 Vim 的 Taglist 插件

从 vim.org 网站上下载 Vim 的 Taglist 插件，如下：

```
$ cd ~
$ wget -O taglist.zip
http://www.vim.org/scripts/download_script.php?src_id=7701
```

安装 Taglist Vim 插件如下：

```
$ mkdir ~/.vim
$ cd ~/.vim
```

```
$ unzip ~/taglist.zip
```

通过添加下面一行到`~/.vimrc` 中来启用插件:

```
$ vim ~/.vimrc
filetype plugin on
```

先决条件: 使用 `taglist` 必须先安装 `ctags`, 但是使用 `taglist` 时不必通过 `ctags` 自动生成标签列表。

用途 1: 在 Vim 中使用 `:TlistOpen` 打开 TagList 窗口

```
vim mycprogram.c
:TlistOpen
```

从 Vim 编译器中执行 `:TlistOpen` 如下所示, 它打开一个标签列表窗口, 带有当前文件的所有标签, 如下所示。

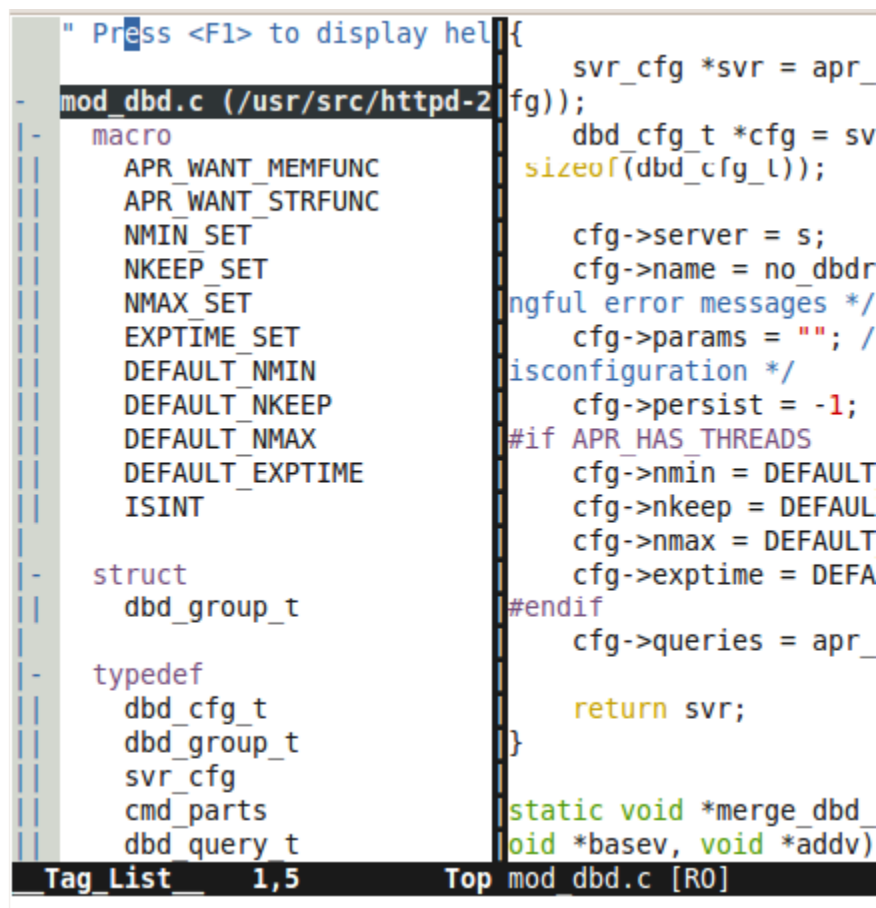
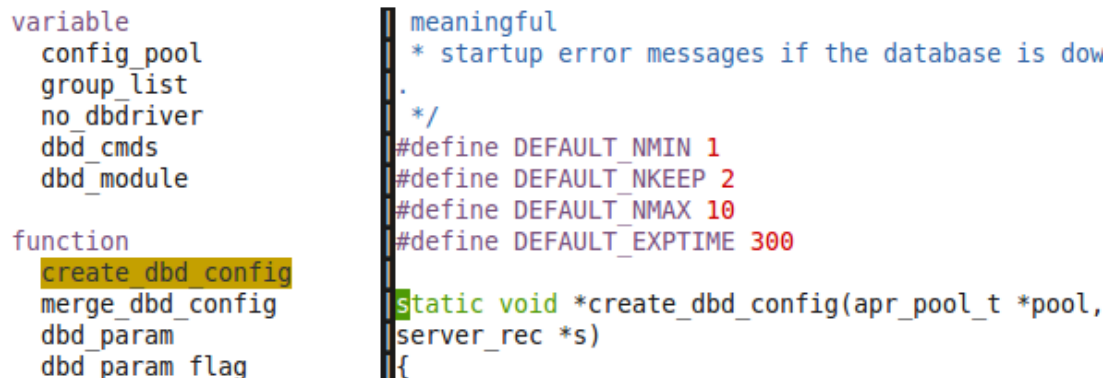


图: 打开 TagList

用途 2: 跳转到源代码中的函数定义

通过点击左边面板中的函数名称, 你可以跳转到函数的定义处, 如下图所示:

除了快速跳转到函数之外，你也可以跳转到类、结构、变量等，通过点击左边面板中标签浏览器里面的对应值。



```

variable
  config_pool
  group_list
  no_dbdriver
  dbd_cmds
  dbd_module

function
  create_dbd_config
  merge_dbd_config
  dbd_param
  dbd_param_flag

meaningful
  * startup error messages if the database is down
  */
#define DEFAULT_NMIN 1
#define DEFAULT_NKEEP 2
#define DEFAULT_NMAX 10
#define DEFAULT_EXPTIME 300
static void *create_dbd_config(apr_pool_t *pool,
server_rec *s)
{

```

图：快速跳转到指定函数处

用途 3：跳转到另一个文件中定义的函数处

当你遇到一个源代码文件的函数，它定义在另外的地方时，你希望移动到函数的定义处，你可以使用两种不同的方法。

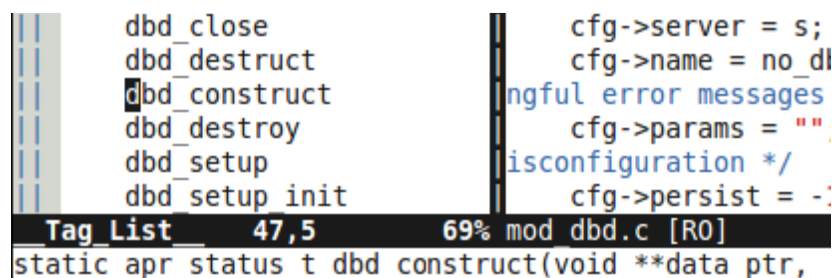
方法 1：如果你有 `ctags` 生成文件，那么光标停留在函数调用处，输入 `CTRL+]` 会跳转到函数定义处。Tag 列表窗口会显示新打开文件的标签。

方法 2：在相同的 Vim 会话中打开另外一个文件，Vim 将更新该文件信息的 Tag 列表窗口。在 Tag 列表窗口中搜索函数名称，并且在函数名上按下 `<CR>`，Vim 跳转到函数定义处。

用法 3：查看函数或者变量的原型/签名

当光标位于 Tag 列表窗口中的函数名或者变量名处，按下 `'space'` 在 Vim 状态栏中显示原型（函数签名），如下所示。

在下面的例子中，在 Tag 窗口中点击 `dbd_construct`，并且按下空格在 Vim 状态栏的底部显示函数的声明。



```

dbd_close
dbd_destruct
dbd_construct
dbd_destroy
dbd_setup
dbd_setup_init

Tag List 47,5 69% mod dbd.c [R0]
static apr_status_t dbd_construct(void **data_ptr,

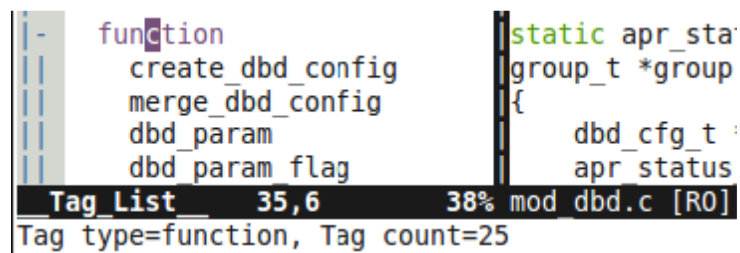
```

图：显示函数声明

用途 5：查看一个源代码文件中函数或者变量的总数量

当光标位于 Tag 类型（例如函数或变量）上时，按 `"space"` 显示该类型标签的数量。

在下面的例子中，当光标位于“function”上按下空格，会显示当前源代码中函数的总个数。



```
function
create_dbd_config
merge_dbd_config
dbd_param
dbd_param flag
static apr_status_t
group_t *group
{
    dbd_cfg_t
    apr_status
```

Tag List 35,6 38% mod dbd.c [R0]
Tag type=function, Tag count=25

图：显示函数的总个数

第五章 基本文本操作

技巧20：插入或者添加文本

插入文本

下面解释各种插入文本到文件的方法。

键	描述
i	在当前位置插入文本
I	在行的开始位置插入文本，键：大写的 I，例如 India
o	在当前行之后插入一行，并且插入文本， 键：小写 o，例如：orange
O	子当前行之前插入一行，并且插入文本 键：大写 O，例如：Orange
:r FILENAME	插入另外一个文件内容到当前文件的当前行之后
:r! COMMAND	插入执行命令的输出到当前文件的当前行之后

例如：你可以插入当前的日期和时间到你编辑的文件中，执行下面的命令：

```
:r! date
```

注：在 Windows 下 date 或 time 是一个时间设置命令，需要输入设置的日期值，这会导致 gVim 无响应。

添加文本

下面说明添加文本的方法：

键	描述
A	在当前光标位置之后添加文本
a	在当前行的结束添加文本

技巧21：代替文本

下面说明各种替换文件中文本的方法。

键	描述
r{c}	使用字符{c}来替换单个字符（当前光标所在的位置上）
R	替换字符直到你按下<ESC> 注意：当移动到一行的最后时，该动作如同 A 一样，而不是回绕替换下一行的字符。

技巧 22：替换文本

下面说明各种替换文件中文本的方法。

键	描述
s	使用新的字符替换当前字符
S	使用新的文本替换当前行
4s	使用新的文本替换 4 个字符（从当前位置开始）
4S	使用新的文本替换 4 行（从当前行开始）

让我们假设我们编辑下面这个文件

```
$ vim employee.txt
```

```
100 John Doe DBA
```

```
200 John Smith Sysadmin
```

```
300 Raj Patel Developer
```

- 如果你的光标在“John Deo”的 D 上，按下 2s，你将会使用你输入的文本替换掉“Do”。
- 如果你的光标在第一行上，按下 2S，你会使用输入的文本替换掉第 1 和第 2 行。

技巧 23：改变文本

下面说明各种改变文件中文本的方法。

键	描述
cc	改变当前整行；与 S 键一样 它会删除掉整行，并且进入到插入模式等待输入新文本
C	从当前光标位置改变当前行 这会使出当前行中光标位置之后的文本，进入插入模式等待输入新的文本。

技巧 24：使用非合并空格选项合并多行

为了合并（组合）两个行，执行下面命令：

```
J
```

如果在光标所在行的结尾没有特殊的字符，当合并两行时候，J 命令将只添加一个空格。

如果再光标所在行的结尾有特殊字符（例如括号），当合并两行时，J 命令将添加两个空格。

为了避免在合并两行时总是使用一个空格，设置下面的选项：

`:set nojoinspaces`

不合并空格!!!

第六章 高级文本操作

技巧25：拷贝一个字符、单词、行或者到指定位置

键	描述
y<字符导航键>	拷贝单个字符
y<词组导航键>	拷贝当个词组
y<行导航键>	拷贝单行
y<mark 名称>	拷贝到标签所在行
y`<标签名称>	拷贝到书签的位置

下面是需要记住的一些关键点：

- y 的全称为“yank”
- 你可以组合任何与导航键相关的操作，直到完成直到那个功能为止的操作，例如，拷贝一个词，输入 yw
- 你可以扩充上面的表格为另外一个其它操作。例如，为了拷贝到屏幕的中间行，使用 yM 键。
- 这个概念对于其它的操作也使用，例如，它不只是对于拷贝/粘贴。

技巧26：在拷贝行/词/其它之后或之前粘贴

键	描述
p (小写的 p)	立即粘贴到当前光标位置之后
P (大写的 P)	立即粘贴到当前光标位置之前

如果你执行几次的删除操作，如果你希望粘贴这些删除的字符，使用下面的方法。

首先，查看寄存器，使用下面的命令：

```
:reg
```

最近的删除内容会显示在 0-9 寄存器中，记下你想粘贴的删除词语的寄存器位置。

如果你想粘贴寄存器标号为 3 的词组，执行下面命令：

```
"3p
```

例如，你可以看到所有的寄存器 0-9 包含先前被删除的文本。

```
:reg
```

如果你希望粘贴寄存器 7 中内容（上面的高亮位置）到你当前文档中，执行下面：

`"7p`

技巧27：删除单个字符、词或者行

删除类似于拷贝，然而你必须使用 `d` 替代 `y`。

键	描述
<code>x</code>	删除当前字符
<code>dw</code>	删除当前词
<code>dj</code>	删除当前行和下一行

技巧28：从剪切板缓冲区插入内容

如果你从一个网络浏览器或者其它应用中拷贝文本，你可以直接粘贴它到 Vim 编辑器当前打开文件中，如下所示。

剪切板中拷贝	描述
<code>SHIFT-INSERT</code>	粘贴剪切板中内容到编辑器中（确保在插入模式下执行）。
<code>"*p</code>	在正常模式下粘贴剪切板内容到编辑器

技巧29：从文件中插入内容到剪切板

你希望把当前文件的文本插入到剪切板中。当你的文本传输到剪切板上，你可以粘贴它到另外一个应用中。

剪切板拷贝	描述
<code>:%y+</code>	拷贝整个文件到剪切板
<code>:y+</code>	拷贝文件中当前行到剪切板上
<code>:N,My+</code>	拷贝文件中指定范围的行到剪切板中

为了拷贝可视化选择的行到剪切板上，首先可视化选择行，`:y+`会显示为:`'</'>y+`。

在拷贝之后，你可以使用传统的`<CTRL+V>`操作粘贴这些内容到任意其它应用中。

技巧30：写文件的部分内容到另一个文件

为了写文件的一部分到新的文件中，你可以使用下面方法中的任意一种。

方法 1: 在可视化模式下选择特殊的几行。进入可视化模式（使用 `v` 或者 `V`），导航到希

望的行上，然后执行：

```
:w newfilename
```

方法 2： 写入文件的部分到另外一个文件，你可以指定范围，如下所示。它写入当前文件的 5 到 10 行到一个新的文件。

```
:5,10w newfilename
```

技巧31：交换相邻的字符

如果你遇到一个简单输入错误，错误放置相邻的字符，你可以使用 **xp**。例如你输入 **the** 替代 **te**，导航到 **e**，按下 **xp**，它会自动修正输入。

xp

实际上，**xp** 不是真的修正输入：

- **x** – 删除当前的 **e** 字符，也移动光标到下一个字符 **h** 处；
- **p** – 在当前的字符 **h** 之后粘贴前一个删除的字符 **e**。
- **xp** 的记忆是“位置交换”。

技巧32：.(点)命令的强大功能

.(点)命令是简单的，也是强大的。命令重复最后一次文件内容影响命令。下面的例子说明命令的使用。

- 1 在文件中搜索字符串，使用： **/john<enter>**
- 2 使用 **json** 替换 **john**，使用： **cwjason<ESC>**
- 3 搜索下一个 **john** 出现地方，使用： **n**
- 4 使用 **json** 替换 **john**，使用： **.(点号)**

在上面的例子中，第 4 步，你不需要再次输入 **cwjason**。替代的，简单的输入.(点号)，它会执行最后一次改变命令，也就是 **cwjason**。

技巧33：可视化模式命令

下面是可视化模式下集中不同的类型：

可视化模式类型	描述
v 小写的 v	开始正常的可视化模式 在可视化模式下使用箭头导航选择文本

v (大写)	开始行可视化模式
CTRL-V	开始可视化块模式

下面的截图显示在这三种可视化模式下的区别。

正常可视化模式

在这个例子中，整个第一行和第二行的部分被选择。这可以通过按下 **v** (小写)，使用箭头导航到一行的指定字符上。

```
100 Jason Smith Developer
200 John Doe Sysadmin
300 Sanjay Gupta QA
400 Ashok Sharma DBA
~
~
```

图：正常可视化模式

行可视化模式

在这个例子中，整个第一行和第二行被选择。这可以通过按下 **V** (大写)，使用箭头选择。在这种模式下，当你按下箭头 (或者 **j** 键)，它会选择整行。

```
100 Jason Smith Developer
200 John Doe Sysadmin
300 Sanjay Gupta QA
400 Ashok Sharma DBA
~
~
```

图：行可视化模式

块可视化模式

在这个例子中，只有第二列 (员工名称) 被选择。它可以通过按下 **CTRL-V**，使用箭头选择列来实现。

```
100 Jason Smith Developer
200 John Doe Sysadmin
300 Sanjay Gupta QA
400 Ashok Sharma DBA
~
~
```

图：块可视化模式

可视化模式命令	描述
<ESC>	退出可视化模式
d	仅删除高亮的文本 例如，如果只是选择一行的部分，它只是删除该行上选择的部分。

D	删除高亮文本下的行。 例如，如果只选择行的部分，它会删除整行。
y	仅拷贝（yank）高亮的文本
Y	拷贝高亮文本所在的行
c	删除高亮文本，进入插入模式
C	删除高亮文本所在的行，进入插入模式

技巧34：使用:g 编辑

下面是一些极好的例子，显示:g 的力量：

例子	描述
<code>:g/^\$/d</code>	删除掉文件中所有空行
<code>:g/^\s*\$/d</code>	删除掉文件中所有空行
<code>:g/^\$/,./-j</code>	减少多个空行为单个空行
<code>:g/pattern/d</code>	删除掉指定模式的行
<code>:g/pattern/.w>>filename</code>	提取指定模式的行，写入到另外一个文件中
<code>:g/^/m0</code>	反转文件
<code>:g/^\s*PATTERN/exe "norm! /* \<ESC>A */\<ESC></code>	添加一个 C 风格的注释（/*文本*/）到所有符合模版的行上。

使用:g!或:v 取消操作

取消操作会匹配除了模版之外的所有事情，说明如下：

创建下面 employees.txt 文件：

```
$ vim employees.txt
Emma Thomas:100:Marketing
Alex Jason:200:Sales
Madison Randy:300:Product
Development Sanjay Gupta:400:Support
Nisha Singh:500:Sales
```

删除所有包含 Sales 的行：

```
:g/Sales/d
```

删除所有不包含 Sales 的行：

```
:g!/Sales/d
（或者）
:v/Sales/d
```


第七章 专家级文本操作

技巧 35：拷贝多行到命名缓冲区随后使用

你可以拷贝（yank）多行到命名缓冲区，你可以随后使用它，如下所示。

可用的命令缓冲区：a 到 z（总共 26 个可用命名缓冲区）

"ayy	拷贝当前行到缓冲区 a
"a5yy	拷贝 5 行到缓冲区 a
"ap	粘贴缓冲区 a 中的拷贝行到光标之后的位置
"aP	粘贴缓冲区 a 中的拷贝行到光标之前的位置

技巧 36：转换插入文本到正常模式的命令

你是否遇到过在插入模式下输入一个正常模式命令的错误？该技巧对这种情况非常有用，如下所示：

- 假设再 Vim 编辑器中有下面的文本——john
- 你希望把 john 修改为 Jason。
- 你忘记你在插入模式，并且输入下面的文笔——cwjasonjohn
- 现在你可以简单的输入<F2>功能键，这里撤销你先前的插入，并且使用它作为正常模式下的命令，这种情况下改变单词 john 为 Jason。

为了完成这个技巧，你应该增加下面一行到你的.vimrc 文件中。

```
$ cat ~/.vimrc
inoremap <F2> <ESC>U@.
```

注意：上面一行中是一个@后面跟.点号。

技巧 37：简写和全写

在下面的例子中，当你定义下面的简写的时候，不管什么时候输入 US，它会解释为“United States”。

为了暂时简写一个词，在命令行模式下执行 abbr 命令，如下所示：

```
:abbr US United States
```

为了永久简写一个词，你可以把它放入到.vimrc 中，如下所示：

为了暂时删除一个简写的定义，在命令行执行 `noabbr` 命令。

```
:noabbr US
```

为了永久的删除掉一个简写的定义，从 `.vimrc` 文件中删除掉它。

如果你需要经常输入你的网址 URL 或者电子邮件地址，可以创建一个简写，如下所示：

```
:iabbrev tgs tgs http://www.thegeekstuff.com  
:iabbrev myemal Ramesh.thegeekstuff@gmail.com
```

在完成上面之后，不管你在什么时候输入 `myemail`，它都会自动展开你的电子邮件地址。

你也可以在 `iabbrev` 值中插入指定的键，例如，你可以添加运输返回键 `<CR>`，如下所示：

```
:iabbrev TRR Thanks,<CR>Regards,<CR>Ramesh Natarajan
```

在这个例子中，不管你什么时候输入 `TRR`，它会扩展为：

```
Thanks,  
Regards,  
Ramesh Natarajan
```

技巧 38：自动拼写更正

`Autocorrect.vim` 插件有一个所有类型拼写错误以及正确拼写的收集。

插件的作者，Anthony Panozzo 描述插件如下：

“当你输入的时候，改正普通的打字错误和拼写错误”

下面是几个来自于 `autocorrect.vim` 插件中的例子：

```
ia Britian Britain  
ia Brittish British  
.  
.  
ia Acceptible Acceptable  
ia accessories accessories
```

安装和配置 autocorrect.vim 插件

从 `vim.org` 网站下载插件：

```
$ cd -  
$ wget -O autocorrect.tar http://www.vim.org/scripts/download\_script.php?src\_id=10423
```

```
$ tar xvf autocorrect.tar
```

安装 `autocorrect.vim` 插件。从 Vim 执行 `:source /path/to/the/autocorrect.vim`，如果你需要永久的使用，添加下面一行到 `~/vimrc` 中。

```
$ vi ~/.vimrc
:source ~/autocorrect.vim
```

完成之后，当你有在 `autocorrect.vim` 列表中的词拼写错误时，它会被自动的更正。

```
$ vim test-typo.txt
This is acceptable
```

注释：上面行呗自动改变为 “this is acceptable”。

当你希望对一个特定的词停止扩展或者拼写错误更正时，你可以在 vim 中进行下面操作。这是暂时的不简写。如果你希望这是永久有效，那么从 `~/vimrc` 或者 `~/autocorrect.vim` 中删除这个词。

```
$ vim test-typo.txt
:una US
```

技巧 39：记录和使用宏

该技巧使用一个例子来解释在 Vim 中如何执行记录和使用宏。

前面几步用于在 vim 中记录和使用宏。

- 第一步：键入 `q` 开始记录宏，后面跟一个小写字母是宏的名称。
- 第二步：在 Vim 编辑器中执行任何类型的编辑操作，它们将会被记录。
- 第三步：按下 `q` 停止记录宏。
- 第四步：通过按下 `@` 后面跟宏名称来使用记录的宏。
- 第五步：为了多次重复宏，按下 `:NN@` 宏名，`NN` 为重复次数。

这个例子说明如何执行相同的记录，使用不同的输入。例如：使用不同的参数来执行相同的命令。

- 1 打开 `change-password.sql`，它只有一些名称。

```
$ vim change-password.sql
Annette
Warren
Anthony
Preston
Kelly
```

```
Taylor  
Stiller  
Dennis  
Schwartz
```

2 开始记录和保存它在寄存器 **a** 中

qa

- q 表示开始记录宏；
- a 表示存储记录到寄存器 **a** 中

当你输入 **qa** 时，它会在屏幕的底部显示消息 “recording”。

3 进入插入模式并且输入 **ALTER USER**

I “ALTER USER ”

放置光标在第一行的任意位置，然后按下 **I**（大写的 i），它会进入到这一行的第一个字符处，输入 **ALTER<space>USER<space>**。

4 拷贝下一个词（例如，名称）

<ESC>w yw

- 按下<ESC>，然后按下 **w** 到相爱一个词（名称）
- **yw**，拷贝当前的词（名称）

5 移动到结束并且输入 **IDENTIFIED BY**

<ESC> A “ IDENTIFIED BY `”

- 按下 **ESC** 和 **A** 移动光标到行的结尾，然后输入空格
- 输入 **IDENTIFIED BY ‘**

6 粘贴拷贝的名称

<ESC> p

按下 **ESC**，然后输入 **p** 粘贴名称，它是在第四步中拷贝的。

7 在结尾完成引号

<ESC> A’

按下 **ESC**，然后输入 **A** 移动到行尾，输入’。

8 跳转到下一行并且停止宏记录

<ESC> j q

- j 移动到下一行
- q 停止记录宏

注意：显示在屏幕底部的记录消息现在消失，在这个阶段，文件 change-password.sql 将如下所示：

```
ALTER USER Annette IDENTIFIED BY 'Annette';
Warren
Anthony
Preston
Kelly
Taylor
Stiller
Dennis
Schwartz
~
~
```

图：Vim 宏完成记录

9 在相关的行上使用参数重复 宏

8@a

- 现在通过输入 8@a 重复这 8 步操作；
- @a 重复宏 a 一次
- 8@a 重复宏 a 8 次，自动完成剩下的行，如下所示。

```
ALTER USER Annette IDENTIFIED BY 'Annette';
ALTER USER Warren IDENTIFIED BY 'Warren';
ALTER USER Anthony IDENTIFIED BY 'Anthony';
ALTER USER Preston IDENTIFIED BY 'Preston';
ALTER USER Kelly IDENTIFIED BY 'Kelly';
ALTER USER Taylor IDENTIFIED BY 'Taylor';
ALTER USER Stiller IDENTIFIED BY 'Stiller';
ALTER USER Dennis IDENTIFIED BY 'Dennis';
ALTER USER Schwartz IDENTIFIED BY 'Schwartz';
~
~
```

图：Vim 宏使用完成

技巧 40：排序文件内容

从 Vim 版本 7 开始，Vim 内建排序命名可以使用。

Vim 中排序文件内容如下所示：

:sort

文件内容的排序部分如下所示：

- 按下 v 进入到可视化模式；

- 使用箭头选择需要排序的多行；
 - 按下`:`在 Vim 的底部它会显示`:'<,>'`。
 - 在排序选择的最后增加`!sort`
- `:'<,>!sort`

`:sort` Vim 命令有下面一些可用的选项：

<code>:sort</code> 选项	描述
<code>:sort</code>	以升序排序
<code>:sort!</code>	以降序排序
<code>:sort i</code>	忽略大小写
<code>:sort u</code>	删除重复行， <code>u</code> 表示唯一
<code>:sort! ui</code>	也可以组合所有排序命令选项

技巧 41：恢复删除的文本

如果你有与误操作删除文本，你可以恢复它。你可以恢复到 9 个删除文本的片段。

恢复删除	描述
<code>"1p</code>	恢复第一次删除
<code>"2p</code>	恢复第二次到最后一次删除
<code>"3p</code>	恢复第三次到最后一次删除

如果你不确切知道你删除的东西，你可以通过下面的方式浏览所有 9 次删除的缓冲区。当你看到你希望恢复的文笔，只需要在这一步停止。

浏览所有删除的内容，直到你找到正确的一个。

`"1pu.u.u.u.u.`

你也可以使用`:reg` 查看寄存器 0 到 9 中（删除寄存器）的文本，这会告诉你每个寄存器中使什么内容。

技巧 42：给文件使用添加自动头部

让我们学习如何在文件中使用 Vim 的强大 `autocmd` 特征来创建一个头部分（例如，在 C 编程代码中的头）。当你使用 `vi` 打开一个文件时，它会自动包括文件名、创建日期、最后修改时间。

Vim 的 `autocmd` 语法：

`autocmd {event} {pattern} {cmd}`

事件：这里有 40 多个 autocmd 事件，下面是几个简单的 autocmd 事件。

事件	描述
BufNewFile	开始编辑不存在的文件
FileReadPre	在使用:read 命令读文件之前
BufWritePre	在开始写入整个缓冲区到文件时
FileWritePre	在开始写入部分缓冲区到文件时
BufDelete	从缓冲区列表中删除缓冲区之前
BufWipeout	在完全删除缓冲区之前
BufNew	在创建新的缓冲区之后
BufEnter	在输入缓冲区之后
BufLeave	在进入另外一个缓冲区之前
SwapExists	删除已存在的交换文件

许多开发者希望程序使用一些缺省头文件。例如当打开.c 文件时，你通常需要一个带有许多 item 的文件头。在我们打开一个新的.c 文件时，自动装载下面这个模板。你可以通过下面的三步来实现。

```
/* -----
* File Name : 1.c
* Purpose :
* Creation Date : 22-12-2008
* Last Modified : Mon 22 Dec 2008 10:36:49 PM PST
* Created By :
-----*/
```

第 1 步：创建一个模板文件

保存上面模板到一个文本文件中，在第一行执行:insert，后面跟着模版，最后一行为. 号，如下所示：

```
$ cat c_header.txt
:insert
/* -----
* File Name :
* Purpose :
* Creation Date :
* Last Modified :
* Created By :
-----*/
.
```

第 2 步：添加 autocmd 命令到 ~/.vimrc

添加下面几行到 ~/.vimrc 中

第 1 行: 定义模板文件, 这指定用于*.c 文件, 使用/home/jsmith/c_header.txt 模板文件;

第 2 行: 从第一行到第 10 行搜索模板"File Name:", 如果找到, 它会在这一行中写入当前的文件名。

第 3 行: 更新创建日期字段

第 5 行: 在保存文件时使用当前的日期和时间更新最后修改字段

第 4 和 6 行: 当保存文件时, 光标将会移动到“最后修改字段”。如果你希望光标返回原来的位置, 你需要添加第 4 行和第 6 行到.vimrc 文件。

第 4 行: 在更新之前标识当前光标位置;

第 6 行: 恢复光标位置到之前的位置。

最后说明:

- 在 Vim 验证 autocmd 是否启用—在 Vim 中执行:version。如果 autocommand 特征启用, 它会显示+autocmd;
- AutoCommand 帮助—在 Vim 执行:help 获得 Vim autocmd 特征的快速帮助。

第八章 Vim 作为程序员编辑器

下面的技巧对所有的编程语言和 Shell 脚本都可用。

技巧43：让 Vim 智能的高亮你的代码

命令	描述
<code>:syn on</code>	打开语法高亮显示
<code>:syn off</code>	关闭语法高亮显示

下面的屏幕截图显示在关闭和打开语法高亮时不同的状态。

语法高亮打开	语法高亮关闭
<pre>// my first program in C++ #include <iostream> using namespace std; int main () { cout << "Hello World!"; return 0; } ~ ~</pre>	<pre>// my first program in C++ #include <iostream> using namespace std; int main () { cout << "Hello World!"; return 0; } ~ ~</pre>

注：Vim 通过文件名的后缀来识别使用哪种编程语言的语法高亮的模版。

技巧44：智能格式缩进

为了在可视化模式下缩进块，执行下面的命令：

- 使用 CTRL-V 模式选择块
- TODO: Windows 下的 gvim 使用 v 键进入可视化，<CTRL-Q>进入列模式。
- 按>移动到块的最右端，按<移动到左边。

有时你可能在选择块之后希望执行多次缩进。当你执行>或<时，它只会缩进一次，可视化选择就会取消。你必须再一次进行可视化选择来执行缩进。

为了避免这种情况，进行下面的设置：

```
:vnoremap < <gv
:vnoremap > >gv
```

当你重新映射<和>成如上所示时，在你缩进之后，块仍然会被选择。这种方式，你可以继续缩进多次，最后你按<ESC>退出可视化模式。

Vim 将完成下面类型的缩进，这依赖于你的设置：

- 自动缩进：当你创建一个新行时，它从当前的行拷贝缩进模式。
- 智能缩进：类似于自动缩进，但是当遇到{[]时会增加/减少缩进。
- C 格式缩进：启用自动 C 编程缩进格式。

技巧45：从 Vim 中访问 Unix 的函数帮助页

在 Vim 编辑器中，在你想阅读帮助（man）页的函数名上按 K 键。

K

为了访问其他章节的 Man 页按{n}K。例如，为了访问 man 页的第 2 章，执行下面命令：

2K

例如：sleep 是一个命令，也是库中的函数，因此，C 编程时你想查看它的 man 页，那么你需要使用 3K。

定制的帮助页查询（例如，改变到 Perldoc）

为了查找不是 Unix 帮助页中的内容，执行不同的 keywordprg。

例如，如果你是一个 Perl 程序员，你会经常使用 perldoc 命令来阅读有关函数的信息。因此，在 Vim 编辑器设置下面的命令：

```
:set keywordprg=perldoc\ -f
```

在上面设置完之后，当你在函数名上按 K 时，它会打开 perldoc 替代 Unix 的 man 页。

技巧46：跳到变量声明处

为了跳到变量本地声明的地方，执行命令：

gd

为了跳到变量的全局定义处，执行命令：

gD

技巧47：对齐变量的赋值语句

下面是一个例子代码，其中变量没有适当的对齐。

```
$a = 1;
$a_very_long_variable_name_value = 1;
$b = 1;
$my_short_variable_value = 1;
```

安装对齐（Align.vim）插件：http://www.vim.org/scripts/script.php?script_id=294

```
$ vim Align.vba.gz
:so %
:q
```

可视化选择需要对齐的文本，执行下面的命令：

```
:'<,>Align =
```

在执行完对齐命令之后，变量看上去适当的对齐了，如下所示。

```
$a                                = 1;
$a_very_long_variable_name_value = 1;
$b                                = 1;
$my_short_variable_value         = 1;
```

你也可以使用下面的对齐方式：

```
:<range>Align Separator1 Separator2
```

范围可以在可视化模式中选择，或者指定行数，例如：5,10 (第 5 到 10 行)。

技巧48：使用 CTRL 键来增加和减少数量

CTRL 键	描述
CTRL-A	增加数量 在 Vim 编辑器中放置光标到数量上，然后按 CTRL-A，这个数值会增加 1
CTRL-X	减少数量 在 Vim 编辑器中放置光标到数量上，然后按 CTRL-X，这个数值会减少 1

注：CTRL-A 在 Windows 下的 gvim 中不可用，可以使用键映射指定：noremap <C-I> <C-A>。

技巧49：在插入模式下执行一个 Vim 命令

当你在插入模式下，如果你想执行一个单个 Vim 命令，你不需要按下<ESC>切换到命令行模式。

为了在插入模式下执行单个 Vim 命令。

CTRL-O

下面是步骤：

- 在插入模式下输入字符
- 按下 CTRL-O，它会暂时进入到命令模式
- 按下任何 Vim 命令（例如，5j 跳转到第 5 行）
- 在执行单个 Vim 命令之后，自动返回到插入模式。

技巧50：查看当前文件详情

当你编辑一个文件时，按下 CTRL-G 或者 g CTRL-G 来查看文件的详情，如下所示。

查看基本文件详细信息。

CTRL-G

查看高级文件详情。

gCTRL-G

技巧：控制 Vim 的状态栏

你可以在 Vim 编辑器中启用状态栏来显示当前文件的有用信息。

缺省情况下，在 Vim 编辑器中状态栏是关闭的。启用状态栏如下所示。

```
:set laststatus=2
```

下面是一个简单状态行的例子：

```
:set statusline=Filename:%t\ Line:\ %l\ Col:\ %c
```

其它状态行例子请查看:help statusline

- :set statusline=%<%f\ %h%m%r%=%-14.(%l,%c%V%)\ %P
- :set statusline=%<%f%h%m%r%=%b\ 0x%B\ \ %l,%c%V\ %P
- :set statusline=%<%f%=\ [%1*%M%*%n%R%H]\ %-19(%3l,%02c%03V%)%O'%02b'
- :set statusline=...%r%{VarExists('b:gzflag','\ [GZ]')}%h...

下面是一些用在状态行上的主要变量。完整的列表请参考:help statusline。

F – 缓冲区中文件全路径。

M – 修改标识，文本是",+" 或者 ",-".

- R – 只读表示，文本为",RO".
- H – 帮助缓冲区，文本是",HLP".
- Y – 缓冲区文件类型，例如： ",VIM".参见 'filetype'.
- N – 打印页数. (仅工作在'printhead' 选项)
- L – 缓冲区中行号.
- c – 列数.
- P – 显示窗口占文件的百分比。如同'ruler'中描述的百分比。总是 3 个字符长度。

技巧 52：改变大小写

Vim 中使用下面的方法改变文本的大小写。

CTRL 键	描述
~	正常模式： 改变光标下的字符的大小写，移动到下一个字符上 如果你继续按下~，你会继续一个个的改变字符直到行的结束 可视化模式： 这回改变所有高亮显示的文本
5~	改变下 5 个字符的大小写
g~{motion-key}	改变从光标到整个指定移动位置的字符的大小写
g~~	改变整个当前行的大小写
gUU	改变整个行文本为大写

guu	改变整个当前行的文本为小写
gUaw	改变当前词为大写
guaw	改变当前词为小写
U	可视化模式：改变当前高亮文本为大写
u	可视化模式：改变当前高亮文本为小写
guG	改变从当前位置到文件结束的文本为小写
gUG	改变从当前位置到文件结束的文本为大写

技巧 53：拼写检查

下面是拼写检查命令：

拼写检查命令	描述
:set spell	开始拼写检查处理 这会在当前文档中高亮显示拼写错误
]s	跳到下一个拼写错误处
[s	跳到前一个拼写错误处
z=	拼写错误单词的建议 从列表中输入数值来选择指定的建议
zg	添加高亮错误单词为一个可用单词
:echo &spelllang	显示用于拼写检查的语言编码，例如，显示 en 为英文
:set spelllang=code	如果缺省的拼写设置不正确，使用这个方法设置你的语言

技巧 54：设置退出确认

如果你忘记保存改变，执行:q，你会得到下面的消息。在这种情况下，你必须存储文件，然后再次:q 退出文件。

```
:q
```

在退出 Vim 时，为了获得存储确认对话框，执行下面命令：

```
:confirm q
```

注意：使用在技巧 80 中讲解 Vim map 特征映射:q 命令为:confirm q 命令。

技巧 55：使用:up 和避免:w

当你执行:w 时，它会保存文件。:w 的主要问题是当你输入:w 时，不管文件有没有改变，它都会更新文件时间戳。

幸运的是，:up 可以保存文件，并且只是在文件存在改变时修改时间戳。

:up

注意：使用技巧 80 中讲解的 Vim map 特征把:w 命令映射为:up 命令。

技巧 56：编辑当前缓冲区内容

你可以改变打开缓冲区的内容，使用 bufdo 命令，如下所示：

语法：:bufdo! LINERANGE ! CMD

例如，你可以替代一个缓冲区中的文本模式，如下所示。

:bufdo! 1,\$! sed "s/pattern/replace/"

（或者）

:bufdo! 1,\$s/pattern/replace/

技巧 57：tab 和空格

下面是一些重要的关于 tab 和空格的命令。

命令	描述
:set expandtab	自动改变 tab 为空格
:set tabstop=4	如果你希望 Tab 转化为 4 个空格，使用 tabstop 来指定它。
:retab	基于 expandtab 和 tabstop 选项转换文件中所有 tab 为空格
:set ai	在插入模式下，Vim 会自动缩进新的行（你按下返回键时）到当前行一样的缩进级别。 在新行开始使用^D 来减少缩进<shiftwidth>个字符。

第九章 Vim 命令行技巧

你可以查看 Vim 中所有可用的命令行选项，如下所示：

```
$ vim -h
```

技巧 58：在只读模式下打开文件

使用 -R 选项在只读模式下打开文件，如下所示：

```
vim -R filename.txt
```

或者

```
view filename.txt
```

当你不想编辑一个文件时，使用上面的方法中的一个习惯问题。这帮助你避免造成不必要的文件修改。

技巧 59：简单地覆盖 Swap 文件

使用 -r 选项来列出当前目录下的 Swap 文件，~/tmp、/var/tmp、/tmp。

下面的命令显示所有的 swap 文件。

在这个例子中，在当前目录中存在三个 swap 文件，它们与先前编译的 file1.c、file2.c 和 change-password.sql 相关。

```
$ vim -r
```

当一个 swap 文件存在时，如果你试图打开它的原始我呢就，你会得到下面的消息。

```
vim file1.c
```

你通常会只查看上面的消息，原因是：

- 一些人正在编辑文件；
- 一个先前编辑会话崩溃。
- 基于为什么会发生下面 Vim 提示中的一种，如下所示：
- 打开只读文件（查看文件内容）；
- 任何地方进行编辑（编辑文件内容）；
- 恢复（使用交换文件来替代文件内容）；
- 退出或崩溃。

技巧 60：在打开文件时执行任意 Vim 命令

使用选项-c，你可以在打开一个文件时执行任何 Vim 命令。

在下面的例子中，Vim 打开文件之后，它会跳转到第 50 行上。

```
$ vim -c ':50' filename.txt
```

你也可以从命令行上执行多个 Vim 命令：

```
$ vim -c '<command 1>' -c '<command 2>' <filename>
```

技巧 61：执行保存在文件中的命令

当你发现自己经常执行相同的 Vim 命令序列，你可以把它们保存在文件中，使用时候执行它们，如下所示。

```
$ vim -w repetitive_task.txt file_to_edit.txt
```

技巧 62：暂时跳过插件的装载

当打开一个文件，你可以只是在特殊文件编辑期间暂时地停止装载所有插件，如下所示。

```
$ vim --noplugin filename.txt
```

技巧 63：进入 Vim 的限制模式

你可以进入 Vim 中限制模式，使用下面方法其中一种。

```
$ vim -Z filename
```

（或）

```
$ rvim filename
```

来自:help -Z

限制模式（-Z）：所有使用扩展 Shell 的命令都不可用。这包括暂停使用 CTRL-Z、:sh、过滤、系统函数、反向扩展等。

第十章 gVim 技巧

gVim 是一个基于 X-Windows 接口的 Vim 编辑器。

技巧 64：显示和隐藏 gVim 菜单和工具栏

有些时候为了获得更多屏幕大小，你可能希望取消 gVim 的菜单栏、工具栏、滚动条或其它的可视化组件。

这可以通过使用 `:set guioptions` 来完成。

例如，为了取消工具栏执行下面的命令，请注意这里在=等号之前有一个-减号。

```
:set guioptions-=T
```

你可以操作 gVim GUI 元素，通过下面的命令：

UI 元素代码	描述
<code>:set guioptions+=TmrlRL</code>	显示所有 gVim 的 GUI 元素
<code>:set guioptions-=TmrlRL</code>	隐藏所有 gVim 的 GUI 元素
<code>:set guioptions-=T</code>	隐藏 gVim 工具栏
<code>:set guioptions-=m</code>	隐藏 gVim 菜单栏
<code>:set guioptions-=r</code>	隐藏 gVim 的左边滚动条
<code>:set guioptions-=l</code>	隐藏 gVim 的右边滚动条
<code>:set guioptions-=R</code>	在窗口垂直分割时隐藏 gVim 的右边滚动条显示
<code>:set guioptions-=L</code>	在窗口垂直分割时隐藏 gVim 的左侧滚动条显示

技巧 65：添加用户菜单或者菜单项到 gVim

你可以添加自定义菜单和菜单项到 gVim 用于定制操作。

在已存在的菜单下面增加菜单项。

Hide Tool Bar（隐藏工具栏）——这将隐藏工具栏

View Tool Bar（查看工具栏）——一旦你隐藏工具栏，使用这个显示工具栏

为了添加 Tools->Hide Tool Bar 菜单，执行下面命令：

```
:amenu Tools.&Hide-Tool-Bar :set guioptions-=T<cr>
```

为了添加 Tools->View Tool Bar 菜单，执行下面命令：

```
:amenu Tools.&View-Tool-Bar :set guioptions+=T<cr>
```

如果你按 **Alt+T** 来下拉 **Tools** 菜单，你可以看到 **H** 和 **V** 高亮。这是因为我们在定义菜单项的时候放置**&**符号在 **H** 和 **V** 之前。

因此，你也可以使用下面的快捷方式调用你定义的用户菜单项：

ALT+T H 隐藏工具栏

ALT+T V 显示工具栏

新建顶级工具栏

下面的例子会添加一个新的菜单栏，称为 **BookMark** (**Alt+K** 键) 和一个菜单项，称为 “**Windows Explorer**(使用 **ALT+K E**)”，它用于打开 **Windows** 浏览器。

```
:amenu <silent>BookMar&K.Windows\ &Explorer :!explorer<cr>
```

注意：上面的 **mingle** 是一个命令，你应该在一行上输入。在**&Explorer** 之后应该有一个空格。

技巧 66: *gVim* 中改变字体

你可能不喜欢 **gVim** 的缺省字体，你可以使用下面两种方法来修改它。

方法 1:

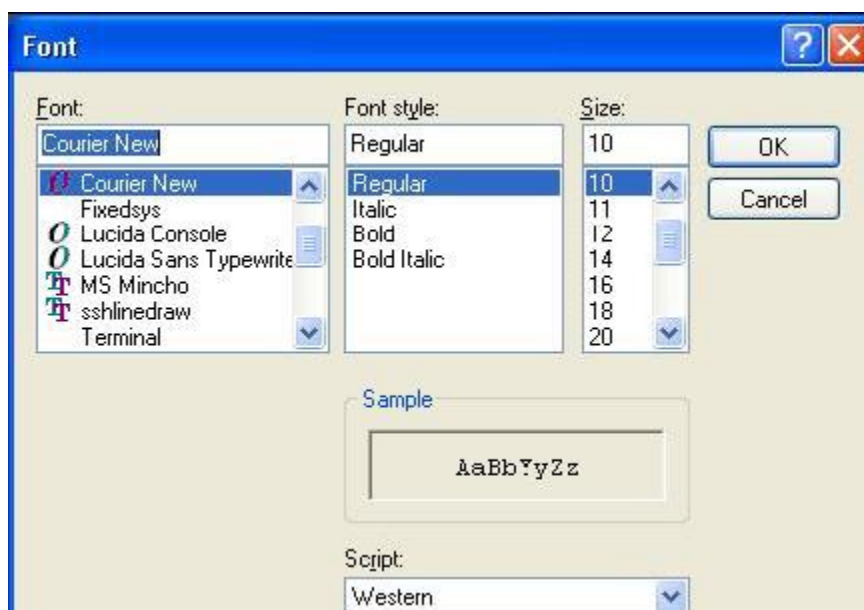
该例子设置字体类型为 **Courier New**，大小为 **10**。对于这种方式，你应该已经知道字体的名称。

```
:set guifont=Courier\ New:h10
```

方法 2:

下面的方法将打开一个字体选择 **UI**，在对话框中你可以选择字体类型和大小。

```
:set guifont=*
```



图：Windows gVim 的字体选择器

第十一章 Vim 外观、标签和窗口

技巧 67：水平和垂直分割窗口

分割当前的窗口

为了分割当前的窗口为两个水平窗口，执行下面命令：

```
:split
```

为了分割当前的窗口为两个垂直窗口，执行下面命令：

```
:vsplit
```

为了关闭分割创建的窗口，输入:q 来关闭窗口

```
:q
```

在另外一个窗口中打开不同的文件

如果你查看/etc/passwd 的时候，你也想使用一个分割水平视图打开/etc/group，执行下面命令：

```
:split /etc/group
```

如果你查看/etc/passwd 的时候，你也想使用一个分垂直视图打开/etc/group，执行下面命令：

```
:vsplit /etc/group
```

在窗口之间切换

为了在窗口之间切换，执行下面的命令：

```
CTRL-W {导航键 -j, k, h, l}
```

例如：跳转到上一个窗口，执行下面指令：

```
CTRL-W k
```

跳转到下一个窗口，执行下面指令：

```
CTRL-W j
```

缩放分割窗口

当再 Vim 编辑器中存在分割窗口时，你可以扩大或者缩小当前窗口的大小，如下所示：

CTRL 键	描述
CTRL-W +	在分割模式下增加当前窗口的大小
CTRL-W -	在分割模式下减少当前窗口的大小

设置窗口的大小

为了打开一个垂直分割含有 25 列的窗口，执行下面命令：

```
:25 vsplit filename.txt
```

为了打开一个水平分割含有 3 行的窗口，执行下面命令：

```
:3 split filename.txt
```

技巧 68： 改变窗口标题

为了改变显示在窗口上的标题，如下：

```
:set title titlestring=My\ Favorite\ File
```

上面的命令会把 Vim 窗口上的标题栏文字修改为 “My Favorite File”。

注：在你执行:help titlestring 时，下面的例子显示在 Vim 文档中。

```
:set title titlestring=%<%F%=%l/%L-%P titlelen=70
```

%F - 当前窗口中的文件名称

%l - 现在光标所在的行数

%L - 文件的总行数

%p - 文件的百分比，例如，如果光标在文件的中间，显示 50%。

技巧 69： 改变 Vim 颜色

首先查看在 Vim 编辑器中所有可用的颜色方案。

```
:!ls $VIMRUNTIME/colors
```

例如：如果你看到在列表中有 blue.vim 或者 evening.vim，你可以改变这些颜色方案，

如下:

```
:colorscheme evening
```

(或者)

```
:colorscheme blue
```

你也可以下载其它颜色方案, 并且把它放置到\$VIMRUNTIME/colors 目录下。链接地址:

http://www.vim.org/scripts/script_search_results.php?keywords=&script_type=color+scheme&order_by=rating

在下载之前, 你也可以查看 <http://code.google.com/p/vimcolorschemetest/>网站上 Vim 颜色方案的截图。

技巧 70: 在标签中编辑多个文件

在一个单个 Vim 会话中编辑多个文件的一个有效的方式是使用标签。

从命令行中打开多个文件。

```
$ vim -p file1 file2 file3
```

下面的屏幕截图显示在标签中打开三个文件。

```
$ vim -p helloworld.cc employee.txt /etc/passwd
```



```
helloworld.cc employee.txt /e/passwd
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World!";
    return 0;
}
~
~
```

图: 以标签方式打开三个文件

当使用标签方式打开文件时, 你可以使用下面的任意一个标签命令。

标签命令	描述
<code>:tabedit FILENAME</code>	在当前 Vim 会话中的新标签中打开另外一个文件
<code>:tabe FILENAME</code>	

:tabs	列出所有打开的标签
:tabn N	转到第 N 个标签上
:tabclose	关闭当前的标签
:tabc	
:tabdo CMD	在所有标签上执行一个命令
:tabn	到下一个标签
:tabp	到前一个标签

第十二章 Vim 编辑器的其它特征

技巧 71：重复一个操作 N 次

“操作操作”能力可以用于所有 Vim 操作，例如，一次向下移动 10 行，你可以输入 10j，如下所示：

```
10j
```

下面有一些重复操作命令：

重复操作命令	描述
@@	重复先前执行的宏
n	在相同的方向上重复搜索
N	在相反的方向上重复搜索
.	重复上一次编辑命令
@:	重复最后一个命令行

技巧 72：撤销和重做操作

单个撤销：

为了撤销一次改变，执行下面命令：

```
u
```

多次撤销：

为了撤销多次改变，执行下面命令。下面的例子会撤销 5 次改变。

```
5u
```

全部撤销：

为了全部撤销所有的改变（在当前行上），执行下面命令：

```
U
```

重做操作：

如果你执行一个错误的撤销，你可以重做这次改版，如下所示：

```
:redo
```

或者

CTRL-R

技巧 73：打开在光标下面的文件

这个技巧在下面这些情况下是有用：

为了确认配置文件给的文件名是有效的；

当编辑一个文本文档时，如果你希望跳转到另外一个文件，它的名称在文档中指定；

当编辑一个源代码，为了访问本地文件，它通过文件名引用来包含或者导入。

打开一个名字在当前的光标下文件（在相同的窗口中）。

Gf

打开一个名字在当前光标下的文件（在新的窗口中）。

CTRL-W f

打开一个名字在当前光标下的文件（在新的标签中）。

CTRL-W gf

如果在光标下的文件名不包含一个全路径或相对路径，Vim 将在当前目录下搜索文件。

对于某些文件，Vim 将打开那些没有全路径文件，Vim 知道这些文件位于哪里，例如：

- C 程序包含的头文件。
- Perl 程序包含的 Perl 模块。

技巧 74：使用传统的方式编辑多个文件

使用这个技巧，你可以在单个 Vim 会话中编辑多个文件。

从命令行上打开多个文件：

```
$ vim file1 file2 file3
```

在 Vim 会话中已经打开另外一个已经打开的文件。

```
:e another_file
```

在当前 Vim 会话中列出所有打开的文件：

```
:ls
```

进入上面:ls 输出中的第 N 个文件:

```
:e N
```

在两个文件之间切换:

```
CTRL-^
```

当编辑多个文件时在文件中移动

在多个文件打开时移动到下一个文件:

```
:next
```

在多个文件打开时移动到前一个文件:

```
:previous
```

技巧 75: 自动保存文件

当你试图切换缓冲区或者文件时, 如果你没有保存改变时 Vim 通常会给出错误消息。

启用切换缓冲区/文件时自动写入文件, 命令如下:

```
:set autowrite
```

使用单个命令写入所有文件 (这可能在宏中非常有用)

```
:wall
```

技巧 76: 在 Vim 加密文件

保存和加密当前文件:

```
:X
```

当你使用:X 加密一个文件时, 下一次打开这个文件, Vim 会提示你输入加密关键词。

技巧 77: 存储和恢复 Vim 会话

当你在 Vim 会话中编辑文件时, 如果你必须执行一些其它的任务, 你不得不关闭所有的文件和 Vim 会话。然而, 在你希望回来并且继续在你先前离开的 Vim 会话中工作。

在这种情况下，你应该保存 Vim 会话，它会保存你当前的设置，例如：

- 缓冲区
- 窗口大小
- 自定义选项
- 折叠
- 当前目录

当你保存会话时，所有上面的信息会被存储。通过使用存储会话命令，你也可以自定义和决定哪些选项你推荐保存的。

为了保存当前的会话，如下：

```
:mksession
```

当你有 N 个打开的文件，折叠，多选项设置，不同的目录，窗口大小定制，mksession 会保存所有这些东西。

为了打开一个保存的会话，执行：

```
: vim -S Session.vim
```

会话命令	描述
:mksession	在当前目录下以缺省文件名 Session.vim 创建一个新的会话
:mksession filename	在当前目录下使用指定的文件名保存会话
\$ vim -S	打开缺省保存的会话，例如，打开当前目录下的 Session.vim
\$vim -S filename	用当前目录下会话文件名打开一个会话
:source Session.vim	在 Vim 编辑器中时，对一个特定的会话文件应用所有的会话设置

技巧 78: Vim 中执行 Unix Shell 命令

为了从 Vim 编辑器中执行一个 Unix Shell 命令，如下：

```
:!unix-command  
:!ls  
:!date
```

你也可以使用下面的方法传递文件名作为参数到 Unix 命令中。

当你在 Vim 编辑器中打开 /etc/sysctl.conf 文件时，假设你正在运行下面的命令。

命令	描述
!echo %	%会传递当前的文件名到 Unix 命令中
sysctl.conf	例如，:ls -l % - 它会在 Vim 编辑器中的当前文件中执行 ls -l

```
:!echo %:p          %:p 会传递文件的全路径名到 Unix 命令中
/etc/sysctl.conf

:!echo %:e          %:e 会传递文件的扩展名到 Unix 命令行中。
conf
```

技巧 79：使用 vimdiff 查看文件之间的不同

与 Unix diff 命令一样，vimdiff 用于显示文件之间的不同的地方。不像 Unix 的 diff 命令，vimdiff 有更多的颜色和用户友好性。

Unix diff 命令文本输出：

```
$ diff employee.txt new-employee.txt
```

Vimdiff 可视化输出：

在下面的例子中，很容易可视化的看到两个文件中哪些地方文本被改变和添加。

```
$ vimdiff employee.txt new-employee.txt
```

vimdiff 命令	描述
\$ vimdiff file1 file2	使用垂直窗口分割显示文件的不同
\$ vim -d file1 file1	
\$ vimdiff -o file1 file1	使用水平窗口分割显示文件的不同
\$ vim -d -o file1 file1	
\$vim file1	如果你已经打开一个文件，使用:diffsplit 以水平方式加载的不同点
:diffsplit file1	
\$vim file1	如果你已经打开一个文件使用:vert diffsplit 以垂直方式加载的不同点
:vert diffsplit file1	
[c	移动到 vimdiff 中下一个改变处
]c	移动到 vimdiff 中前一个改变处

技巧 80：Vim 的 map（映射）命令

使用 Vim 映射特征，你可以映射一个键到特别的功能上，这样你可以重复的执行它。

Vim 中创建一个映射

在下面的例子中，任何时候你输入:write，它会编译当前打开的*.c 程序文件，并且如果编译成功，执行./a.out。

```
:map :write :!cc % && ./a.out
```

`:map` - Vim 创建映射的命令

`:write` - 映射的名称（映射名）

`!cc % && ./a.out` - 当映射名调用时执行的命令

执行映射

为了执行映射，调用映射的名称。在上面显示的例子中，`:write` 是映射的名称。

当你输入`:write`时，它会使用`!cc && ./a.out`来替换，并且 Vim 将使用`$SHELL`编译 C 程序，并且执行 `a.out`。

其它映射例子

你可以映射`:w`命令为`:up`命令，如下所示：

```
:map :w :up
```

你可以映射`:q`命令为`:confirm q`，如下所示：

```
:map :q :confirm q
```

显示定义的映射

输入`:map`来显示所有定义的映射，如下所示：

```
:map
```

技巧 81：使得 Bash 脚本如 Vim 编辑器一样工作

当你开始熟悉 Vim 快捷方式时，你希望 Unix 命令行提示符使用 Vim 一样的快捷方式。

缺省 Bash 命令行使用 emacs 键值完成编辑。。

为了使用 Vim 键值编辑命令行，设置如下：

```
$ set -o vi
```

在你执行上面命令之后，bash 将虚拟成 Vi 的插入模式。按 ESC 来进入命令行模式。在哪里你执行 `vi` 命令来完成命令行的编辑。

确保永久的改变，在`.bashrc`中设置这个选项。

```
$ cat ~/.bashrc
set -o vi
```

执行下面停止 Vim 模式，并且回到 emacs 模式。

```
set -o emacs
```

技巧82：设置 Vim 选项

在 Vim 编辑器中有一些 set 选择可用，下面罗列出其中的一些键值：

选项	描述
<code>:set nu</code>	显示数值
<code>:set ic</code>	搜索时忽略大小写
<code>:set ro</code>	停止 Vim 文件写入
<code>:set wm=n</code>	设置右边忽略的列数，Vim 可以得到最佳换行
<code>:set ai</code>	打开自动缩进
<code>:set all</code>	显示所有的 Vim 会话设置
<code>:set list</code>	显示不可见字符，例如^I 为 Tab 键，\$为行结束
<code>:set hlsearch</code>	高亮显示匹配的模式
<code>:set incsearch</code>	激活增量搜索模式

技巧83：不设置（取消）Vim 选项

在 set 选项之前添加“no”可以取消 Vim 选择设置，下面是一些关闭的特殊选项：

```
:set on<OPTION>
```

为了显示行号：

```
:set nu
```

为了不设置显示行号选项：

```
:set nonu
```

技巧84：缺省寄存器及其使用

任意时刻删除、拷贝或者替换文本，它在你所访问的寄存器中是可用的。

这里有一些存储信息的缺省寄存器，解释如下：

寄存器名称	描述
<code>%</code>	当前文件名称
	可选的文件名称
<code>:</code>	最近执行的命令行

/	最后搜索的模式
"	最后使用的寄存器

为了粘贴寄存器中的内容，执行下面：

```
"<Register Name>p
```

例如，在文件中以文本的方式粘贴文件名，在正常模式下如下实现：

```
"%p
```

技巧85：数字寄存器和恢复删除

下面是一些寄存器相关的重点：

这里有 10 个寄存器可用，数字为 0 到 9；

最近拷贝存储在寄存器 0 上；

最近的删除存储在寄存器 1 上。

对于每个删除，Vim 移动以前的内容从 1 到 2，2 到 3，等等。

技巧86：Vim 目录操作

你可以使用 Vim 编辑器作为文件管理器来导航文件系统和执行各种操作，它在本技巧中进行说明：

```
vim DIRNAME
```

下面的命令在 Vim 内显示/etc/下所有目录和文件的列表。

```
vim /etc/
```

你可以在 Vim 内部执行下面的操作进行文件浏览：

键	描述
<ENTER>	打开当前光标下的文件，进入到光标下的目录
D	删除光标下的文件
R	重命名光标下的文件
X	执行光标下的文件
O	打开一个水平分割的窗口

你也可以从 Vim 内部登陆 Vim 文件浏览器，使用下面的方法：

键	描述
:Ex (数字:浏览)	在 Vim 文件浏览器中打开当前目录
:Ex ~/etc/	在 Vim 文件浏览器中打开指定目录
:Sex	在 Vim 文件浏览器中在水平分割窗口中打开当前目录
:Vex	在 Vim 文件浏览器中在垂直分割窗口中打开当前目录
Tex	在 Vim 文件浏览器中在新的标签下打开当前目录

第十三章 搜索的力量

技巧 87：利用搜索导航

执行 / 搜索术语来搜索从当前位置开始的第一个出现的关键词。

导航键	描述
/	向前搜索（也用于找到下一个）
?	向后搜索（也用于查找前一个）

当你使用 / 搜索术语时，它会从当前位置搜索搜索词语第一次出现的地方。之后，你只需要输入 `n` 或 `N` 来再次搜索相同的查询词语。这类似于许多应用中的“Find Next”和“Find Previous”。

导航键	描述
<code>n</code>	移动到下一个出现的地方（Find Next）
<code>N</code>	移动到前一个出现的地方（Find Previous）
<code>// (或) ??</code>	重复先前的移动或者反向的搜索

技巧 88：移动到当前词的下一个/前一个出现位置

有时你想搜索你光标所在的关键词的下一个出现地方。

导航键	描述
*	移动到当前光标下关键词的下一个出现的地方。
#	移动到当前光标下关键词的前一个出现的地方。

提示：这与前一个技巧有点不同，你不需要第一次搜索关键词，然后执行 `n` 或者 `N`。替代的，你可以在没有第一次搜索光标所在的关键词的情况下输入 * 或 #。

你也可以完成部分搜索当前光标下的关键词。

导航键	描述
<code>g*</code>	移动到当前光标下关键词出现的下一个位置。
<code>g</code>	移动到当前光标下关键词出现的前一个出现位置。

为了列出光标下的关键词所有出现的位置，使用 `[I`（大写的 i）。

`[I`

技巧 89：在一行中搜索一个字符

使用下面的键在一行中进行导航。

导航键	描述
fX	向前移动到一行中的某个字符 X
FX	向后移动到一行中的某个字符 X
tX	向前移动到一行中的字符 X 前面一个字符处
TX	向后移动到一行中的字符 X 前面一个字符处
;	向前重复最后的 f,F,t 或者 T
,	向后重复最后的 f,F,t 或 T。

技巧 90：12 个有用的查找和替换例子

在本技巧中，我们学习如何在 Vim 编辑器中执行基本和高级的文本和模式替换特征。这些特征会使用 12 个非常实用和强大的文本替换的例子来说明。

Vim 编辑器中文本替换命令的语法如下：

```
:[rang]s[bustitue]/{pattern}/{string}/[flags] [count]
```

下面是一些可用的替换标识(flags)：

[c] 每次替换之前确认

[g] 替换在每行上出现的词

[i] 忽略模式的大小写

例子 1：在整个文件中使用一个文本替换另一个文本的所有出现过的地方

在 Vim 中，这是一个非常普通的文本替换特征的用法。当你希望使用一个指定的文本替换掉整个文件中所有出现过的另外一个文本，那么你就用下面的命令：

```
:%s/old-text/new-text/g
```

%s 执行所有的行，用%指定范围标识在整个文件中进行替换

g 标识每行上所有出现的地方，如果 g 标识没有使用，那么只有第一个行上出现的地方会被替换掉。

例子 2：在单行上使用一个文本替换掉另一个文本

缺少指定范围意味着只是体会当前行商的文笔。使用 i 标识，进行大小写不敏感搜索。

```
:s/hello/Hello/gi
```

例子 3: 在许多行的范围内使用文本替换一个文本

你可以只指定行的范围，在这个范围内会受到替换的影响。指定 1, 10 作为范围限制，替换只会在 1~10 行上进行。

```
:1,10s/l/We/g
```

例子 4: 在行可视化模式下进行文本替换

你通过指定可视化选择行来控制替换的范围，在命令模式下，输入 CTRL-V，使用导航键选择你希望使用的文件部分，然后按下":,"，它自定改变为:','>。当自动转换为'<,>'时，你可以开始输入命令的剩余部分，如下所示：

```
:','>s/helo/hello/g
```

例子 5: 只在下一个行数量中替换文本

在替换命令的最后指定一个计数参数，替换应该应用在下{count}行商。例如，为了替换当前行的下面 4 行，你可以输入：

```
:s/helo/hello/g 4
```

例子 6: 只替代整个词而不是匹配部分

让我们假设你希望在原始文本中只改变整个词“his”为“her”。如果你开始标准的替换，部分从 his 变为 her，它也会改变 This 到 Ther，如下所示。

标准替换：

```
:s/his/her/g
```

整个词替换：

```
:s/\<his\>/her/
```

注意：你应该使用<和>来封闭住单词，它会强迫替换只是搜索整个单词，而不是部分的匹配。

注意 2：新手有时候使用空格替代<和>，没有认识到它不会匹配行开始和结尾的单词，或则带有附近有标点符号的单词。

例子 7: 使用正则表达式让一个新的词替代词 1 和词 2

在下面的例子中，Vim 将使用 **awesome** 来替代所有 **good** 或 **nice** 出现的地方。

```
:%s/\(good\|nice\)\/awesome/g
```

你也可以通过正则表达式进行全部单词替换。下面的例子使用 **hai** 替换 **hey** 或者 **hi**。请注意它不进行更长词的替换，例如 **they** 或者 **this**。

```
:%s/<\(hey\|hi\)\/>/hai/g
```

\<和\>- 次的边界

\| - 逻辑或（在这种情况下是 **hey** 或 **hi**）

例子 8: 在 Vim 编辑器中交互查找和替换

你可以在替换命令中使用 **c** 标识执行一个交互查找和替换，它要求询问确定是否进行替换，如下解释。在这个例子中，Vim 编辑器完成单词 ‘**awesome** 的’ 的全局搜索，并且使用 **wonderful** 进行替换。然而，它只是基于你的输入进行替换，解释如下。

```
:%s/awesome/wonderful/gc
```

y - 将替换当前高亮的单词。在替换之后它会高亮下一个匹配搜索模式的单词。

n - 将不体会当前高亮的单词，但是会自动高亮下一个匹配搜索模式的单词。

a - 将替换所有剩下匹配的单词，不再进行提示。

l - 将只替换当前高亮的单词，并且终止查找和替换的功能。

例子 9: 使用它的行号为每行添加行号

当替换字符串以 ‘\=’ 开始，它会作为表达式进行评估，使用 **Line** 函数，我们可以获得当前的行号。

```
:%s/^/\=line(".")". "/g
```

注意：这与 **:set number** 不一样，后者不会把行号写入到文件中。当你使用这个替换时，这些行号会成为文件实际内容的一部分。

例子 10: 使用一个等效的值替换特殊字符

使用 **\$HOME** 变量的值来替换 **~** 字符。

```
:%s!\~!\=expand($HOME)!g
```

注意：

你使用 `expand` 函数扩展所有预定义变量或者用户自定义的变量

上面使用 `!` 替代 `/`，因为 `$HOME` 值会包含至少一个 `/` 字符，它会使替换困惑。有许多可以用于替代 `/` 的字符。

例子 11: 当插入新项时选择数字列表中的序列值

假设再文本文件中有一个数字列表，如下所示。在这个例子中我们假设你希望添加一个新行到 `Article 2` 后面。为此，你应该相应改变所有其他文档的数值。

第 3 篇文章” `Make Vim as Your Perl IDE Using perl-support.vim Plugin`” 遗漏掉了，因此你希望添加它，此时你希望改变 `Article 3` 为 `Article 4`，`Article 4` 为 `Article 5`，一直到 `Article 12` 为 `Article 13`。

这可以通过下面的 `Vim` 替换命令来实现：

```
:4,$s/\d\+/<=submatch(0)+1/
```

范围: `4,$` - 从第 4 行到最后一行（4 是人为指定的）

搜索模式- `\d\+` - 一个数字字符串

替换模式 - `<=submatch(0)+1` - 获得匹配的模式，并且加 1

标识 - 治理没有标识，缺省情况下它只替换每行第一个出现的位置。

在执行替换语句之后，文件如下所示，这时你可以添加第三篇文章。

例子 12: 转换每个句子的首字符为大写

当格式化一个文档，正确的大写句子是相当有用的，这可以使用下面的替换来实现。

```
:%s/\.\s*\w/<=toupper(submatch(0))/g
```

`\.\s*\w` - 搜索模式，一个 `.` (点号) 后面跟着零或多个空格，然后是单词字符。

`toupper` - 转换给定的文本为大写

`submatch(0)` - 返回匹配模式

替换之前的文本，如下：

Lot of vi/vim tips and tricks are available at thegeekstuff.com. reading these articles will make you very productive. following activities can be done very easily using vim editor.

a. source code walk through,

- b. record and play command executions,
- c. making the vim editor as ide for several languages,
- d. and several other @ vi/vim tips & tricks.

替换之后的文本，如下：

Lot of vi/vim tips and tricks are available at thegeekstuff.com. Reading these articles will make you very productive. Following activities can be done very easily using vim editor.

- a. Source code walk through,
- b. Record and play command executions,
- c. Making the vim editor as ide for several languages,
- d. And several other @ vi/vim tips & tricks.

技巧 91：使用 *vimgrep* 在多个文件中进行搜索

你可以使用 *vimgrep* 命令在多个文件中进行搜索。你可以在 Vim 编辑器中执行 *vimgrep*，如下所示。

下面的例子会在当前目录下所有以.txt 结尾的文件中搜索词语 *jasom*。

```
:vimgrep Jason *.txt
```

实际上，*vimgrep* 会跳转到包含匹配单词的第一个文件，使用:cn 跳转到下一个匹配的地方。

关键词	描述
:vimgrep search-term *	在多个文件中搜索 search-term
:cn	跳转到 vimgrep 搜索的下一个匹配地方
:cN	跳转到 vimgrep 搜索的上一个匹配地方
:clist	查看所有匹配 vimgrep 搜索关键词的文件，不跳转到单个的文件。
:cc number	跳转到指定搜索数字。基于:clist 的输出选择搜索数字。

你也可以使用 *vimgrep* 进行递归的搜索。使用**指定递归搜索，如下所示。

例如，下面会搜索在当前目录和所有子目录下*.html 文件中的搜索词 *table*。

```
:vimgrep table **/*.html
```

技巧 92：颜色高亮搜索结果

当你搜索一个关键词，你可能希望自动高亮所有符合的单词。使用:hlsearch 选择。

为了启用搜索结果高亮

```
:set hlsearch
```

在设置之后，当你使用 `/keyword` 进行关键词搜索时，在当前文件中的所有匹配的关键词自动高亮。

键	描述
<code>:set hlsearch</code>	启用搜索结果高亮显示 注意：添加这个到 <code>~/.vimrc</code> 中永远启用
<code>:set nohlsearch</code>	关闭搜索结果高亮显示
<code>:nohlsearch</code>	清除掉当前的搜索高亮显示

技巧 93: Vim 增量搜索

当你使用增量搜索后，在 Vim 中不能没有它。

为了使用增量搜索

```
:set incsearch
```

增量搜索会在输入的时候就开始搜索关键词。

关闭增量搜索，

```
:set noincsearch
```

技巧 94: :match 的力量

在当前颜色方案中使用 `:match` 来显示所有特殊关键词的实例。例如，如果你希望以红色高亮关键词 `Error`，使用下面的命令：

```
:match ErrorMsg /Error/
```

在上面的例子中：

`:match - match 命令`

`ErrorMsg` - 在 Vim 中可用的预先定义的颜色方案（red）

`/Error/` - 用户定义的搜索模式

下面是一些 Vim 中可用的预定义的颜色方案：

ErrorMsg
WarningMsg
ModeMsg
MoreMsg

你也可以创建自己的颜色方案，如下所示：

```
:highlight custom-color-scheme ctermbg=COLOR ctermfg=COLOR
```

当创建预定自定义颜色方案之后，你可以使用它，如下所示：

```
:match custom-color-scheme /\cPATTERN/
```

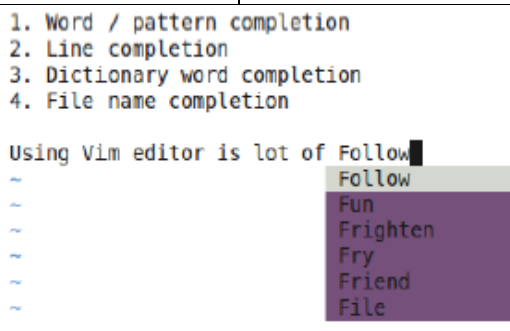
第十四章 自动完成

技巧 95：自动补全单词

在 Vim 中使用 CTRL-X 你可以在插入或者添加模式下执行自动补全单词功能。通过输入词的前几个字符，你可以从字典、同义词字典或者编辑文件中已经出现的关键词中获得整个单词。

你可以使用下面的 Vim 快捷键来选择指定关键词的已有的扩展形式。

键	描述
CTRL-X CTRL-N	关键词的自动完成-向前
CTRL-X CTRL-P	关键词的自动完成-向后

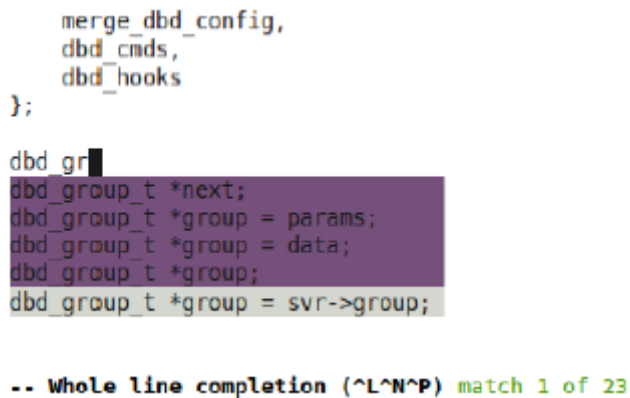


图：在 Vim 中使用 CTRL-X CTRL-N 进行关键词补全

技巧 96：自动行补全

如果你希望插入一个已有行的拷贝，输入行的前几个词/字符串，然后输入 Vim 快捷键“CTRL-X CTRL-L”，它会显示所有符合模式的行。

CTRL-X CTRL-L

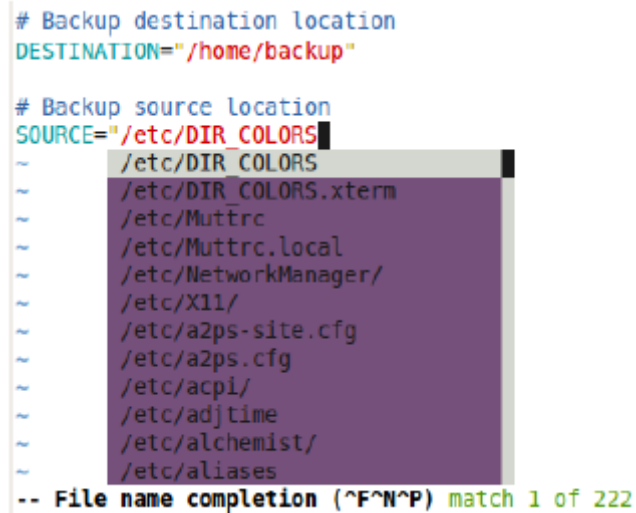


图：Vim 使用 CTRL-X CTRL-L 进行整行补全

技巧97：自动文件名补全

使用快捷键“CTRL-X CTRL-F”插入当前用户可以看到的 Linux 系统中任何文件名。

CTRL-X CTRL-F



```
# Backup destination location
DESTINATION="/home/backup"

# Backup source location
SOURCE="/etc/DIR_COLORS"
~ /etc/DIR_COLORS
~ /etc/DIR_COLORS.xterm
~ /etc/Muttrc
~ /etc/Muttrc.local
~ /etc/NetworkManager/
~ /etc/X11/
~ /etc/a2ps-site.cfg
~ /etc/a2ps.cfg
~ /etc/acpi/
~ /etc/adjtime
~ /etc/alchemy/
~ /etc/aliases
-- File name completion (^F^N^P) match 1 of 222
```

图：Vim 使用 CTRL-X CTRL-F 进行文件名补全

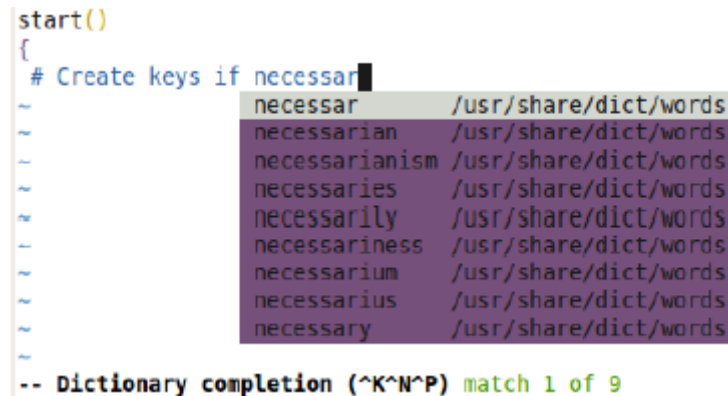
技巧98：字典补全

通过添加下面几行到 ~/.vimrc 文件中，启用 Vim 的字典。

```
$ cat ~/.vimrc
set dictionary+=/usr/share/dict/words
```

当你用于更正输入单词的拼写错误时，这是一个很好的特征。在输入前几个字符之后，输入 Vim 快捷键 CTRL-X CTRL-K 显示符合字典中的单词。

CTRL-X CTRL-K



```
start()
{
# Create keys if necessary
~ necessar /usr/share/dict/words
~ necessarian /usr/share/dict/words
~ necessarianism /usr/share/dict/words
~ necessities /usr/share/dict/words
~ necessarily /usr/share/dict/words
~ necessariness /usr/share/dict/words
~ necessarium /usr/share/dict/words
~ necessarius /usr/share/dict/words
~ necessary /usr/share/dict/words
-- Dictionary completion (^K^N^P) match 1 of 9
```

图：Vim 使用 CTRL-X CTRL-K 进行字典单词补全

技巧 99：同义词字典单词补全

该技巧解释如何使用 Vim 有效的启用一个同义词字典，下面三个步骤：

第 1 步：定义一个同义词字典文件

许多同义词可以罗列在单行上，每个通过空格分隔单词，或者如果一些同义词是多个单词成语则使用逗号分隔。例如，你可以创建一个你自己的同义词字典文件，“important” 单词的同义词如下所示：

```
$ vim /home/jsmith/mythesaurus.txt
important,valuable,substantial,significant
```

第 2 步：在 ~/.vimrc 中指定同义词字典的文件位置

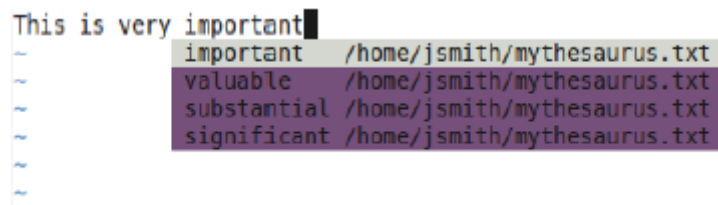
```
$ vim ~/.vimrc
set thesaurus+=/home/jsmith/mythesaurus.txt
```

第 3 步：编辑文档时使用 CTRL-X CTRL-T 来使用同义词字典

Vim 中，如果你希望使用一个可选的单词，在插入模式下输入 CTRL-X CTRL-T。

例如，当你输入单词 “important” 时，输入 CTRL-X 和 CTRL-T，会出现一个可选单词 “valuable”、“substantial” 和 “significant” 的弹出菜单，这些词来自于 /home/jsmith/mythesaurus.txt 文件中，如下所示：

CTRL-X CTRL-T



图：使用 CTRL-X CTRL-T 从 Vim 中导入同义词字典

下载和使用预定义的同义词字典

下载和使用著名的预定义的巨大的同义词字典替代定义自己的同义词字典，如下所示：

```
$ wget http://www.gutenberg.org/dirs/etext02/mthes10.zip
$ unzip mthes10.zip
Archive: mthes10.zip
inflating: aaREADME.txt
inflating: roget13a.txt
```

```
inflating: mthesaur.txt
```

使用 `mthesaur.txt` 作为同义词字典文件。它是一个非常大的文件，每个单词可以获得 50 个以上的相关单词。

```
$ vim ~/.vimrc
set thesaurus+=/home/jsmith/mthesaur.txt
```

Vim 中程序员如何使用同义词字典特征？

这对程序员是非常有帮助的。例如，一个 PHP 程序员使用下面几行创建一个 php 函数文件，在 `~/.vimrc` 中指定这是一个同义词文件。

```
$ vim /home/jsmith/php-functions.txt
math abs acos acosh asin asinh atan atan2 atanh
base_convert bindec ceil cos errors debug_backtrace debug_print_backtrace
error_get_last error_log error_reporting
restore_error_handler
```

添加 `php-functions.txt` 到 `.vimrc` 文件中指定同义词字典文件的位置。

```
$ vim ~/.vimrc
set thesaurus+=/home/jsmith/mythesaurus.txt
set thesaurus+=/home/jsmith/mthesaur.txt.txt
set thesaurus+=/home/jsmith/php-functions.txt
```

现在，当你在 PHP 文件中输入 `math`，按下 `CTRL-X` 和 `CTRL-T`，所有 PHP 数值函数将会显示。当然，请注意你可以定义多个同义词函数，如上所示。

技巧 100：自动打开一个弹出菜单用于补全

从 `vim.org` 上下载 `autocomplpop.vim` 插件，如下所示：

```
$ mkdir -p ~/.vim/plugin
$ cd ~/.vim/plugin
$ wget -O autocomplpop.zip http://www.vim.org/scripts/download\_script.php?src\_id=11538
```

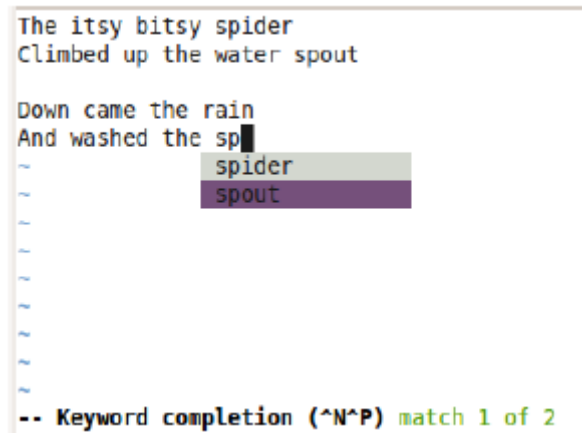
在 `~/.vimrc` 文件中增加下面一行来启用插件：

```
$ vim ~/.vimrc
filetype plugin on
```

例子 1：弹出可变单词选择菜单

安装完插件之后，你不需要输入一个命令序列来激活它。它会自动激活。你输入两个字

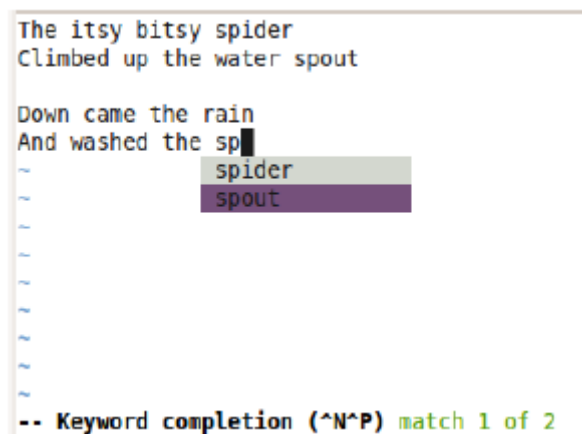
符的时候，它会显示以这两个字符开始的可用单词的选项。在这个例子中，当输入 `sp` 时，它会在弹出对话框中显示 `spider` 和 `spout`。



图：匹配单词的自动补全菜单

例子 2：文件名补全弹出菜单

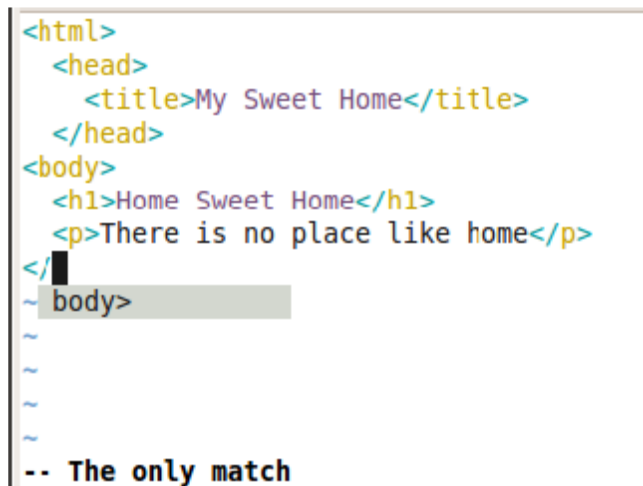
当你在程序内输入文件名时，它会自动显示适当选项的弹出菜单。如下所示：



图：匹配文件名的自动补全弹出菜单

例子 3：标签的全部补全

插件可以对 HTML XHTML CSS Ruby Python 语言进行全部补全。在下面例子中，当你输入 `</` 时，它会自动列出可用的标签。



图：HTML 文件全部补全弹出菜单

技巧 101：输入时自动提供单词补全

当你在 Vim 编辑器中输入一些东西，word_complete.vim 插件可以显示只有一个相关单词，如果你需要显示的单词，你可以按下 TAB 键来接受这个建议。

这是一个完全没有打扰和有效的插件。

安装和配置 Vim 单词补全插件。

从 [vim.org](http://www.vim.org) 网站上下载 word_complete.vim 插件，如下所示：

```
if the directory does not exist already
$ mkdir -p ~/.vim/plugin
$ cd ~/.vim/plugin
$ wget http://www.vim.org/scripts/download\_script.php?src\_id=6504
```

启用插件，添加下面一行到 ~/.vimrc 文件中。

```
$ vim ~/.vimrc
filetype plugin on
```

两种方式启用自动完成插件

方法 1： 在需要的时候启用

这可能是更好的方法，在你需要的时候启用自动补全方法。只要你打开一个文件，执行:call DoWordComplete()来暂时的启用该插件。

```
$ vim mystory.txt
: call DoWordComplete()
```

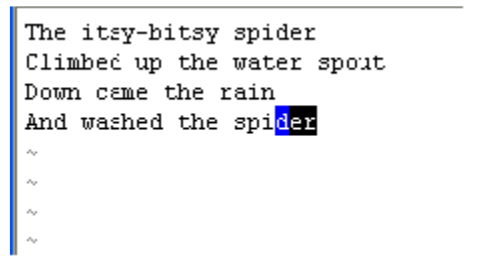

当你缺省就启用这个插件，如果你想暂时停止该插件，执行下面命令：

```
$ vim mystory.txt  
:call EndWordComplete()
```

在下面显示的例子中，当你输入 **spi** 时，它自动填充单词 **spider**。

如果你希望接受这个建议的单词，按下 **TAB** 键；

如果你不想接受这个建议的单词，只需要继续输入；



```
The itsy-bitsy spider  
Climbed up the water spout  
Down came the rain  
And washed the spider  
~  
~  
~  
~
```

图：自动补全插件提示单词 **spider**。

第十五章 额外技巧

额外技巧1：为项目的列表添加项目符号风格

假设你喜欢在 Vim 编辑器中转换下面的项目到项目符号。

The Geek Stuff article categories:

Vi / Vim Tips and Tricks
Linux Tutorials SSH Tips and Tricks
Productivity Tips
HowTo & FAQ
Hardware Articles
Nagios Tutorials
MySQL and PostgreSQL Tips

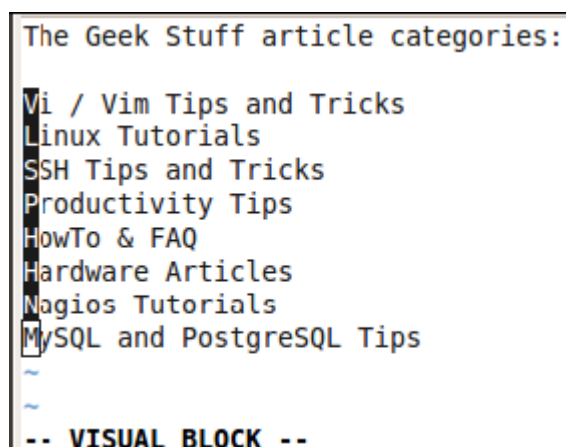
下面几步可以把上面文本转换为项目符号。

第 1 步：移动到需要转换为项目符号的第一项的第一个字节，从哪里开始输入 CTRL-V 进入可视化模式。

CTRL-V

注：在 gVim 中使用 v 进入可视化模式，CTRL+Q 进快选模式。

第 2 步：在可视化模式下选择项目符号的第一个字符，如下所示：



```
The Geek Stuff article categories:
Vi / Vim Tips and Tricks
Linux Tutorials
SSH Tips and Tricks
Productivity Tips
HowTo & FAQ
Hardware Articles
Nagios Tutorials
MySQL and PostgreSQL Tips
~
~
-- VISUAL BLOCK --
```

图：可视化块模式下选择

第 3 步：按下 I 键（大写的 i）

第 4 步：按下 TAB 键

第 5 步：插入*并且按空格，在这个阶段将添加项目符号到第一个项目。

第 6 步：通过按下 ESC 键 2 次对所有项目重复它。

ESC ESC

在上面六步之后，你可以看到项目编号如下所示：

The Geek Stuff article categories:

- *Vi / Vim Tips and Tricks
- *Linux Tutorials SSH Tips and Tricks
- *Productivity Tips
- *HowTo & FAQ
- *Hardware Articles
- *Nagios Tutorials
- *MySQL and PostgreSQL Tip

额外技巧 2：设置 Vim 作为通用缺省编辑器

如果你是系统管理员，你可能希望设置 Vim 作为整个系统的缺省编辑器。如下使用 update-alternatives 选项，这在所有基于 Debian 的系统上可以使用。

```
update-alternatives --set editor <PATH of VIM>
```

额外技巧 3：是的 Vim 作为缺省编辑器

你可以设置 Vim 为缺省编辑器，如下所示：

```
$ export EDITOR=vi
```

许多 Unix 有用程序（例如，crontab）引用该 EDITOR 变量来检查所使用的编辑器。因此，最好把它设置在 bashrc 中。

额外技巧 4：格式化段落

为了格式化段落，如下所示：

```
gqap
```

额外技巧5：编辑可重用的宏

在你记录一个宏之后，如果你发现宏里面有一个错误，你可以有以下两个选择：

- 再次记录宏
- 编辑宏，只是更正错误的地方。

下面三步说明如何编辑宏和更正错误（替代重新记录宏）。

第一步：从记录宏的寄存器中粘贴宏

“ap

第二步：编辑宏，在这个例子中，FOR 应该是 BY，如下所示：

ALTER USER ... identified BY ...

第三步：拷贝宏到寄存器中

“ayy

额外技巧6：缩进代码块

这里有两种方法来缩进代码，如上所示。

方法 1：

- 移动光标到{或}处
- 按下>i{缩进位于{和}之间的代码

方法 2：

- 移动光标到{之后的第一行，例如 printf
- 按下 v 启用可视化模式
- 使用箭头选择在{和}之间的行
- 按下>来缩进位于{和}之间的代码。

注：使用>进行右缩进，使用<进行左缩进。

额外技巧7：合并的力量

你可以通过编辑命令与导航命令一起使用产生强大的结果。

例如，dj 会一行行的删除，d`a 将删除到标签 a 的位置。

键	描述
d<Navigation key>	删除到导航键指定的位置
dw	删除一个词
d\$	删除到行尾
d0	删除到行开始
dG	删除到文件的结尾
dgg	删除到文件的开始
dk	删除当前行和前一行
dj	删除到当前行和下一行
dM	删除到屏幕的中间
dH	删除到屏幕的中间
dL	删除到屏幕的底部
y<Navigation key>	拷贝到导航键指定的位置
c<Navigation key>	改变直到导航键指定的位置

额外技巧 8：标识文件的改变

在打开文件之后，你可以使用:changes 来标识所有文件上做的修改，如下所示：

```
:changes
```

额外技巧 9：刷新屏幕

当你的屏幕由于一些原因可视化不正常，你可以使用 CTRL-L 重画它。

```
CTRL-L
```

额外技巧 10：插入非键盘字符

你可以插入非键盘的字符到文件中，使用:digraphs

```
:digraphs
```

例如，插入版本符号，在插入模式下执行：

```
CTRL-K Co
```

来自于:help digraphs

Digraph 用于输入一个普通键盘不能正常输入的字符。这些最原始的字符有 8bit 集合。Digraphs 比十进制数还容易记住他们。可以输入 CTRL-V。（参考\i_CTRL-V|）。

额外技巧 11: Vim ex 模式

从正常模式下，按下 Q 进入到 Vim ex 模式。

当你希望执行命令时可以进入 ex 模式，与:mode 一致。

当你按下 Q，你会留在 ex 模式(:mode)一直到你决定退出 ex 模式。

进入 Ex 模式:

Q

退出 Ex 模式

:vi

额外技巧 12: 放置光标在匹配的最后

当你在 Vim 内使用 /pattern 搜索时，缺省光标位于匹配的开始。

但是，如果你希望光标放置在匹配的结尾，你可以使用 /pattern\zs。

光标在模式的开始位置:

/pattern

光标在模式的结尾处:

/pattern\zs

额外技巧 13: 查看字符的 ASCII 值

由于一些原因，如果你希望知道一个字符的十进制、十六进制或者八进制的值，移动你的光标到字符处，按下 ga，它会显示 ASCII 值，如下所示:

字符 n 的 ASCII 值:

ga

<n> 110, Hex 63, Octal 156

额外技巧 14: 在 Vim 编辑器中编辑二进制文件

为了在 Vim 编辑器中编辑二进制文件，使用选项 -b 到 Vim 命令行上，如下所示:

```
$ vim -b binaryfile
```

额外技巧 15：换行—仅查看代码要求的部分

自动换行

为了启用基于缩进的换行，如下设置：

```
:set foldmethod=indent
```

在这样设置之后，你的代码将会基于缩进换行，如下所示：

你也可以操控换行使用下面的键：

换行键	描述
za	切换在光标下的折行
zR	不进行所有的折行
zM	再次进行所有折行

手动折行

启用手动折行，如下所示：

```
:set foldmethod=manual
```

你可以操作手动折行，使用下面的键：

换行键	描述
zf<Navigation-key>	通过导航键选择折行的行
zf/pattern	通过搜索模式选择折页的行
:range fold	通过范围指定折页的行

你也可以存储所有你的折行，如下所示：

```
:mkview
```

打开视图查看所有保存的定制折行：

```
:loadview
```

反馈和支持

我希望你发现 Vim 101 技巧的电子书是有用的，我诚挚的感谢 www.thegeekstuff.com 博客上所有读者给予的支持，他们以自己的方式鼓励我。

请使用这个回复地址 <http://www.thegeekstuff.com/contact> 给我发送本书中提到的 101 条技巧的反馈、问题或者更正。