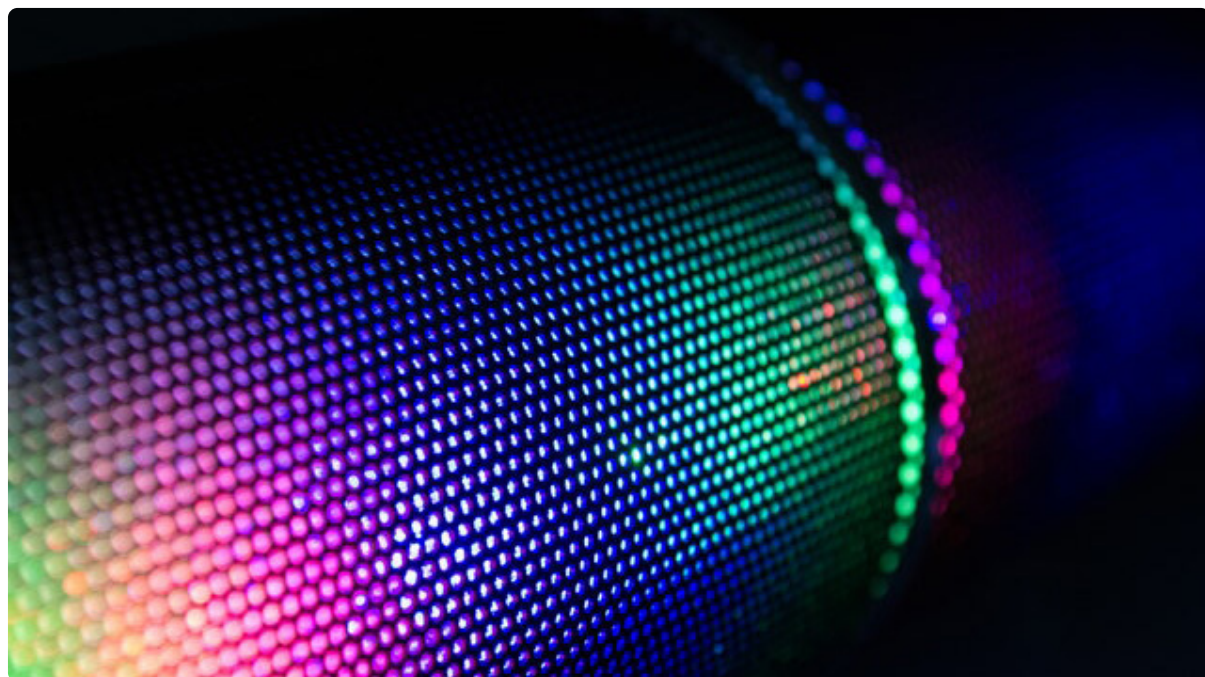


40 Vim的标准和高级操作，配置Vim

更新时间：2019-08-07 20:11:54



“知识是一种快乐，而好奇则是知识的萌芽。”

——培根”

内容简介

1. 前言
2. 标准操作（复制，粘贴，撤销等等）
3. 高级操作（分屏，合并，查找等等）
4. 配置 Vim
5. 总结

1. 前言

上一课是 [带你玩转Linux和Shell编程 | 第五部分第一课：Vim岂是池中物，宝剑锋从磨砺出](#)，我们初步认识了 Vim 这个极为强大的编辑器。

这一课我们继续学习 Vim，好将 Vim 这把宝剑磨得更锋利。

2. 标准操作（复制，粘贴，撤销等等）

我们已经了解了 Vim 的基本操作。虽说基本操作不是很难，但是对用惯了一般文本编辑器的朋友来说也是会有点不适应。所以花时间来练习是必要的。

现在，我们继续深入。你将会发现 Vim 非常方便快捷，甚至部分“吃瓜群众”可能会吃惊到连瓜皮都掉了。

我们会在 Vim 的交互模式下做大部分操作，因此如果你还不是在交互模式下，请按 Esc 键。

x：删除字符

在交互模式下，将光标定位到一个你想删除的字符上，按下字母键 `x`（小写的 `x`），你会发现这个字符被删除了。效果和插入模式下用 `BackSpace` 键（退格键）来删除字符一样。

我们也可以一次性删除多个字符，只需要在按 `x` 键之前输入数字即可。比如我要删除从光标处字符开始到后面的 4 个字符，我可以先按下键盘上的数字键 4，然后再按 `x`。你会发现，4 个字符被删除了。如果输入 12，再按下 `x`，那么从光标处开始往后的 12 个字符都会被删除。

d：删除单词，行等等

我们用字母键 `d`（`d` 是 `delete` 的首字母，是英语“删除”的意思）来删除单词或者行。其实被删除的内容会被暂存在内存里，就好像“剪切”。被剪切的内容之后是可以被粘贴的。我们之后会学到，要使用字母键 `p`。

先从删除行开始：

dd：删除行

连接两次 `d` 来删除光标所在的那一行。

`dd` 也可以和数字配合，以实现一次性删除多行。例如，先输入 2，再按下 `dd`，就会删除从光标所在行开始的 2 行。

dw：删除一个单词

将光标置于一个单词的首字母处，然后按下 `dw`（`delete word` 的缩写，表示“删除单词”），整个单词就会被删除了。如果光标置于单词中的某个字符上，那么只会删除从当前字符开始到下一个空格前的所有字符。当然，你也可以一次性删除 3 个单词，只需要依次按下 `3dw`。而且如果你把数字放在 `d` 和 `w` 之间也是可以的，例如 `d3w`。

d0 和 d\$：删除行首或行末

还记得 `0` 键和 `$` 键吗？之前我们请大家在 Vim 中尽量用这两个键来实现跳转到行首和行末，而不要用 `Home` 和 `End` 键。

- 按下 `d` 键，再加 `0` 键，就会删除从光标处到行首的所有字符。
- 按下 `d` 键，再加 `$` 键，就会删除从光标处到行末的所有字符。

yy：复制行到内存中

按两次 `y` 会把光标所在行复制到内存中。和 `dd` 类似，`dd` 用于“剪切”光标所在行到内存中，而 `yy` 是“复制”。

`yw` 会复制一个单词，`y$` 是复制从光标所在处到行末的所有字符，`y0` 是复制从光标所在处到行首的所有字符。

`y` 是 `yank` 的首字母，是英语“拔出，抽出”的意思。在 Vim 中，`yank` 就是复制（`copy`）的意思。就好像把文本拔（`yank`）起来，之后要粘贴就是放（`put`）上去。

p：粘贴

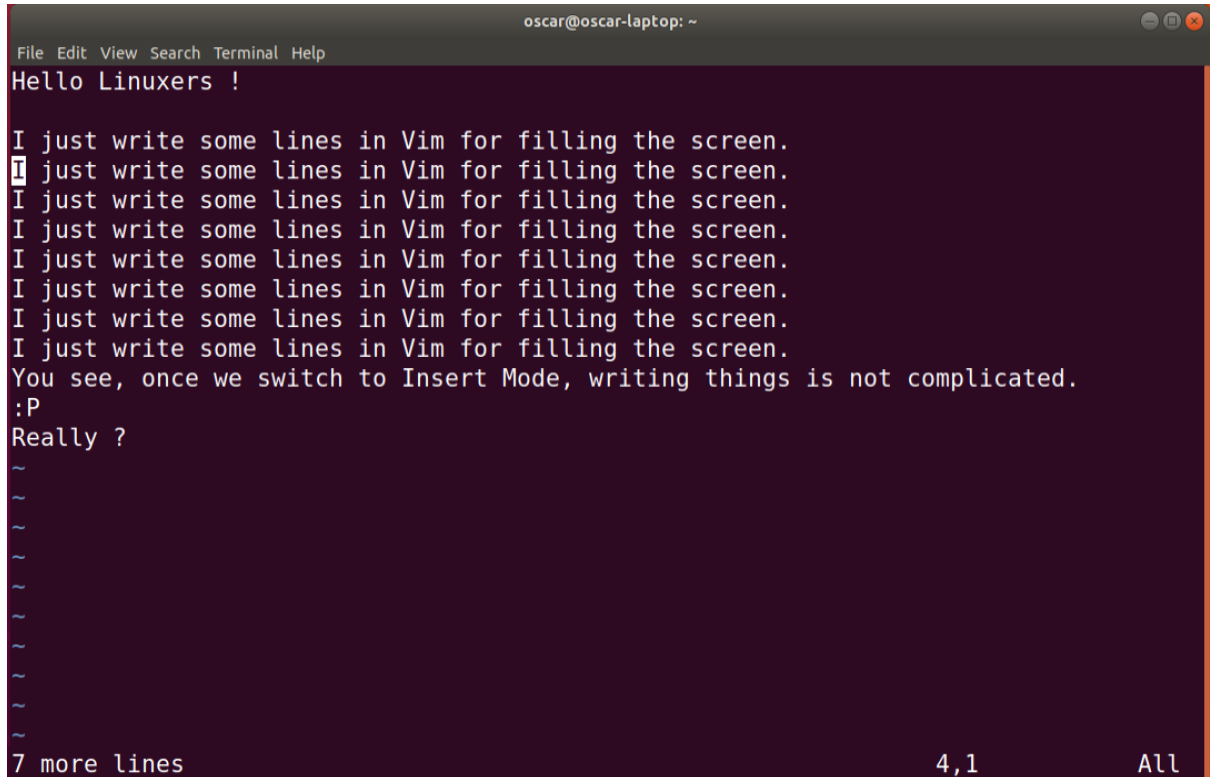
如果你之前用 `dd` 来剪切过一行，或者用 `yy` 来复制过一行，或者是同类操作（比如 `y$`，`dw`，`y0`，`d0`，`d$` 等等），那么可以使用 `p` 键来粘贴这些内容。

`p` 是英语 `paste` 的首字母，表示“粘贴”。

注意：用 p 来粘贴时，内容会被粘贴到光标之后。

如果你用 yy 复制了一行，再用 p 来粘贴，那么这一行会被粘贴到光标所在行的下一行处。

我们也可以将同样的内容粘贴多次，只需要在 p 前面加上次数。例如 7p，表示粘贴 7 次。如下图所示：



```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
Hello Linuxers !
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
~
~
~
~
~
~
~
~
7 more lines                                4,1                                All
```

可以看到左下角显示“7 more lines”，是英语“多了 7 行”的意思。

r：替换一个字符

如果你输入文本时不小心输错了一个字符，你可以用替换来解决。

在交互模式下，将光标置于想要替换的字符上，按下 r 键（r 是 replace 的首字母，是英语“替换”的意思），接着输入你要替换的字符。例如，rs 表示替换当前字符为 s。

如果你用大写的 R，那就是切换到替换模式了。左下角会显示 -- REPLACE --。在替换模式下，你可以一次性替换多个字符。例如下图中，我把 I jus 替换为了 Hello，注意左下角的字样：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
Hello Linuxers !  
  
I just write some lines in Vim for filling the screen.  
Hello! write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
You see, once we switch to Insert Mode, writing things is not complicated.  
:P  
Really ?  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
-- REPLACE --
```

4,6 All

要退出替换模式很简单，按 Esc 键即可。

u： 撤销操作

这是 Vim 中使用频率很高的按键。

如果要撤销最近的修改，只需要按下 u 键（u 是 undo 的首字母，是英语“撤销”的意思）。同样的，如果想要撤销最近四次修改，可以按下 4，再按下 u。

可以看到，Vim 中“数字+指令”是很通用的方式。

为了取消撤销，也就是重做之前的修改，只需要按下 Ctrl 键 + r 键（r 是 redo 的首字母，是英语“重做”的意思）。

g： 跳转到指定行

g 是 go 的首字母，是英语“去”的意思。

Vim 编辑的文件中，每一行都有一个行号。行号从 1 开始，逐一递增。

在 Vim 中，我们可以注意到右下角有类似这样的字样：

4,6

[illegible]

4 表示行号，6 表示列号。因此 4,6 表示当前光标位于第四行，第六列。

如果我们要跳转到第 7 行，我们可以按下 **7G**。注意，这里的 G 是大写。因此，按下 7，再按下 Shift + g，或者 7 + gg 也行（就是先按 7，再按两次 g）。

- 要跳转到最后一行，按下 G（大写的 G，Shift + g）
- 要跳转到第一行，按下 gg（按两次 g 键）
- 跳转到指定行：行号 + G 或 行号 + gg

3. 高级操作（分屏，合并，查找等等）

到目前为止，我们已经见识了 Vim 的常用操作。现在是时候学习一些更复杂的操作了。例如：合并文件，查找，替换，分屏等等。

所有这些操作都是在交互模式下进行。

/ : 查找

如果你按下 /（斜杠）键，那么就进入了查找模式。

这时，你会在左下角看到一个斜杠符号，而且光标会转到斜杠右边，意思是让你输入要查找的字符串：

```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
Hello Linuxers !

I just write some lines in Vim for filling the screen.
Hello! write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
~
~
~
~
~
~
~
~
/
```

此时输入你要查找的字符串，然后按下回车。光标就会转到文件中下一个查找到的匹配处。

如果字符串不存在，那么会显示“Pattern not found”，表示“没有找到匹配项”。如下图中我要查找“boy”，我在 / 之后输入了 boy，再按回车。但是文件中没有这个字符串：

```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
Hello Linuxers !

I just write some lines in Vim for filling the screen.
Hello! write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
~
~
~
~
~
~
~
~
~
E486: Pattern not found: boy 1,1 All
```

如果要查找下一个匹配项，只需要按 n 键（n 是 next 的缩写，表示“下一个”）。如果要反向查找，需要按 N（大写的 n，也就是 Shift + n）。

用斜杠来进行的查找是从当前光标处开始，向文件尾搜索。


```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
Hello Linuxers !

I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
~
~
~
~
~
~
~
~
4 substitutions on 4 lines                                6,1      All
```

- `:%s/旧字符串/新字符串/g`：替换文件中所有匹配的字符串（应该是最常用的吧）。例如 `:%s/just/love/g` 会“替换文件中所有 just 为 love”。

```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
Hello Linuxers !

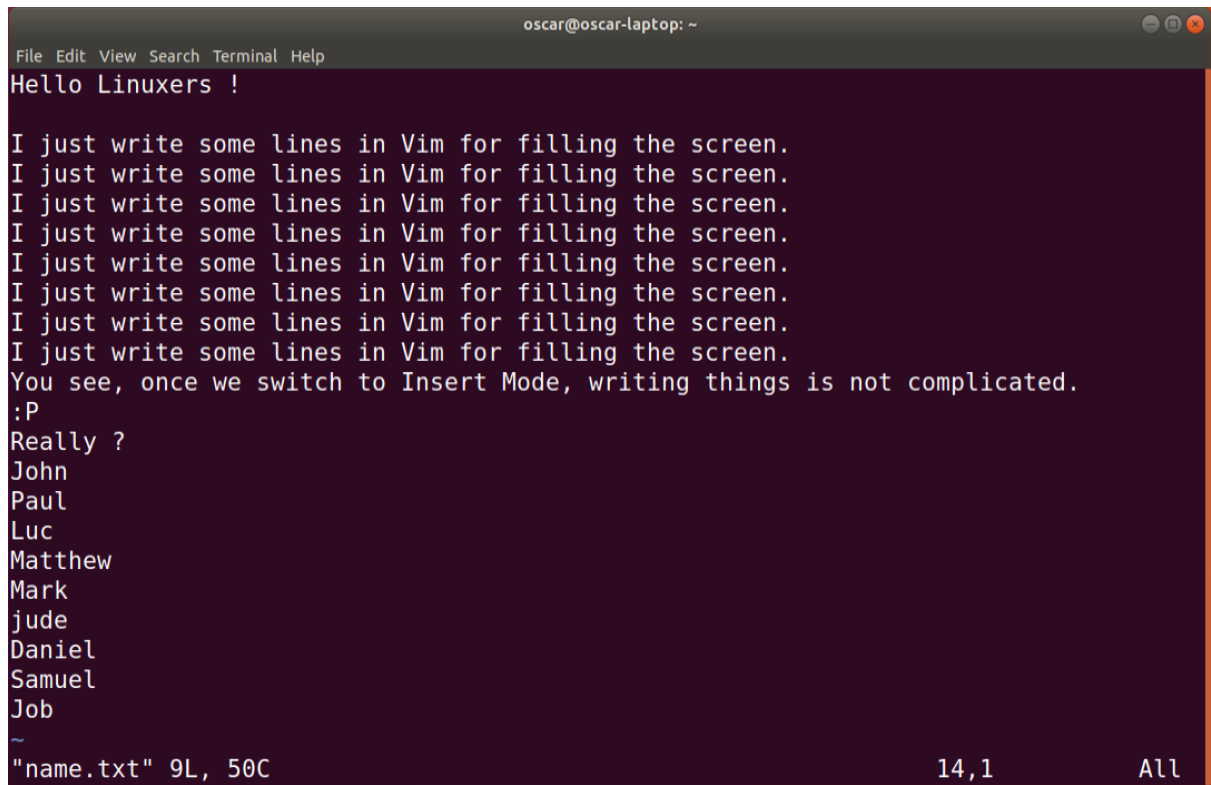
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
I love write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
~
~
~
~
~
~
~
~
8 substitutions on 8 lines                                10,1     All
```

`:r`：合并文件

我们可以用 冒号 + r (`:r`) 实现在光标处插入一个文件的内容。例如：

```
:r 另一个文件名
```

可以用 Tab 键来自动补全另一个文件的路径。比如我插入了之前我们创建的 name.txt 文件的内容：



```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
Hello Linuxers !

I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
John
Paul
Luc
Matthew
Mark
jude
Daniel
Samuel
Job
~
"name.txt" 9L, 50C                               14,1      All
```

分屏

Vim 有一个特别便捷的功能：可以分屏，可以同时打开好几个文件。

是否想起了我们之前讲分屏操作那一课（讲了 screen 命令和 Terminator 软件）？

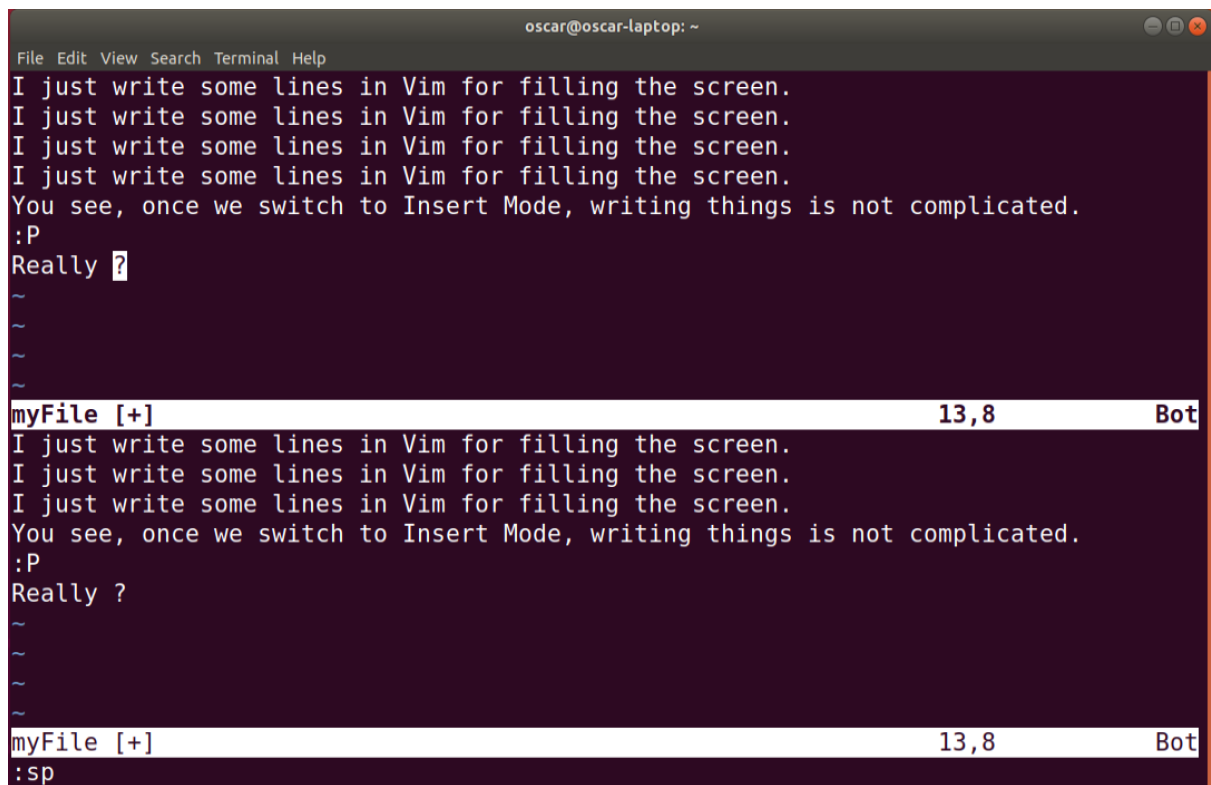
分屏之后，屏幕的每一块被称为一个 viewport，表示“视口”。

`:sp`：横向分屏

首先，我们来学习横向分屏。只要输入 `:sp`，然后回车即可。

sp 是 split 的缩写，是英语“分割”的意思。默认是横向分割。

如下图，屏幕被分成了上下两块：



```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
~
~
~
~
myFile [+] 13,8 Bot
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
I just write some lines in Vim for filling the screen.
You see, once we switch to Insert Mode, writing things is not complicated.
:P
Really ?
~
~
~
~
myFile [+] 13,8 Bot
:sp
```

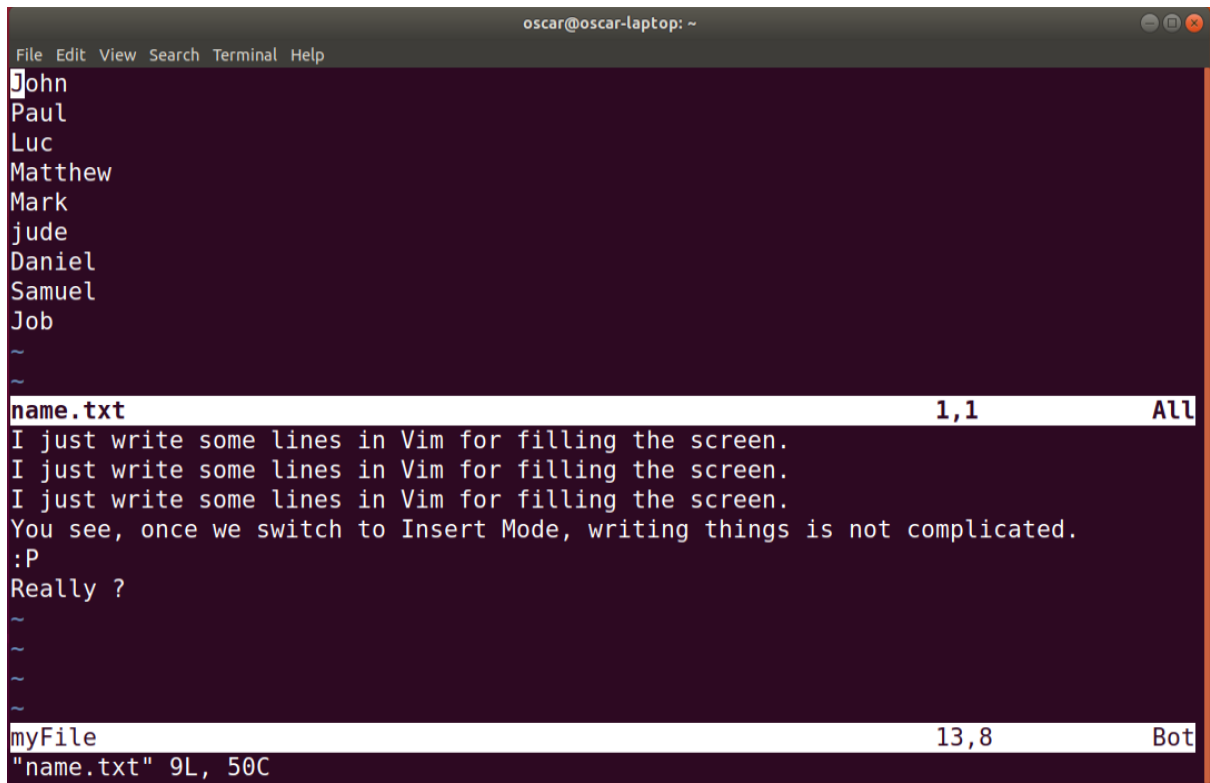
可以看到，myFile 文件又被打开了一次。这时两个 viewport 的内容是同一个文件：myFile。

当然，我们也可以在两个分开的屏幕中分别打开不同的文件。只需要在输入 `:sp` 之后空一格，再输入要打开的另一个文件名：

```
:sp 另一个文件名
```

可以用 Tab 键来自动补全另一个文件的路径。

例如我用 `:sp name.txt` 来打开了 name.txt 这个文件，可以看到现在变成上下两个不同的 viewport，分别打开的是 name.txt 和 myFile 这两个文件：



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
John  
Paul  
Luc  
Matthew  
Mark  
jude  
Daniel  
Samuel  
Job  
~  
~  
name.txt 1,1 All  
I just write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
I just write some lines in Vim for filling the screen.  
You see, once we switch to Insert Mode, writing things is not complicated.  
:P  
Really ?  
~  
~  
~  
~  
myFile 13,8 Bot  
"name.txt" 9L, 50C
```

你可以再输入一次 `:sp`，来把屏幕分成 3 块。再输入一次，屏幕会被分成 4 块，以此类推。

`:vsp`：垂直分屏

如果你想垂直分屏，那就使用 `:vsp`。

`vsp` 是 `vertically split` 的缩写，`vertically` 表示“垂直地”，所以 `vsp` 表示“垂直分割”。

当然，横向分屏和垂直分屏是可以组合的。如下图：



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
John  
Paul  
Luc  
Matthew  
Mark  
jude  
Daniel  
name.txt 1,4 Top  
John  
Paul  
Luc  
Matthew  
Mark  
jude  
Daniel  
name.txt 1,1 Top  
I just write some lines in Vim for filling the screen.  
You see, once we switch to Insert Mode, writing things is not complicated.  
:P  
Really ?  
~  
~  
~  
~  
myFile 13,8 Bot
```

分屏模式下的主要快捷键

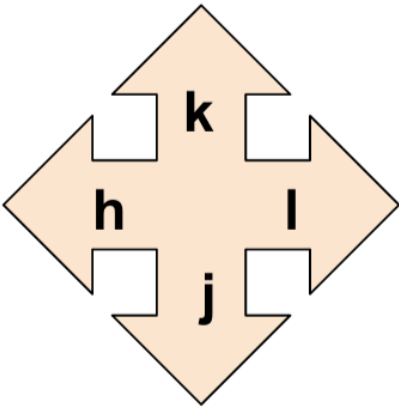
下面列出在分屏模式下的一些实用的快捷键：

Ctrl + w 然后再按 Ctrl + w ： 从一个 viewport 移动光标到另一个 viewport。

Ctrl + w 然后按 j ： 移动光标到下方的 viewport，如果是 h、k、l，那么是分别对应移动到如下表所示的 viewport：

按键	作用
Ctrl + w 然后按 h	移动光标到左边的 viewport
Ctrl + w 然后按 j	移动光标到下边的 viewport
Ctrl + w 然后按 k	移动光标到上边的 viewport
Ctrl + w 然后按 l	移动光标到右边的 viewport

我们上一课讲过 h、j、k、l 在 Vim 中的作用：



Ctrl + w 然后按 + ： 扩大当前 viewport。

Ctrl + w 然后按 - ： 缩小当前 viewport。

Ctrl + w 然后按 = ： 重新均匀分配各个 viewport 的占比。

Ctrl + w 然后按 r ： 调换各个 viewport 的位置。用 R 的话是反向调换。

Ctrl + w 然后按 q 或按 c ： 关闭当前 viewport。输入 `:quit` 或 `:close` 也有一样效果。q 是 quit 的缩写，表示“退出”。c 是 close 的缩写，表示“关闭”。

Ctrl + w 然后按 o ： 只保留当前所在 viewport，关闭其他 viewport。输入 `:only` 也有一样效果，o 是 only 的缩写，表示“仅仅，只”。

有了以上的快捷键组合，你应该可以“任意驰骋”于分屏之间。

`:!` ： 运行外部命令

在 Vim 中可以运行一些终端命令。只要先输入 `:!` ，然后接命令名称。

例如，`:!ls`，就是 Vim 打开的文件所在目录运行 ls 命令，就会列出当前目录包含的文件：

对于 Vim 的插件，我们就不花时间来讲解了，你感兴趣的话可以安装所需插件。但是有些选项参数倒是很有激活的必要。

选项参数功用

在 Vim 被启动后，我们可以运行一些指令来激活一些选项参数。但是这些选项参数的配置在你退出 Vim 时会被忘记。

如果你希望你所做的配置是永久性的，那么需要在你的家目录创建一个 Vim 的配置文件 `.vimrc`。这和我们之前用 `.nanorc` 来配置 Nano 编辑器是一样的原理。

以命令模式激活选项参数

前面提到的“短暂性”配置选项参数的方法是通过运行一些命令。一旦 Vim 被打开后，要激活一个选项参数，只需要输入：

```
:set 选项名
```

`set` 是英语“设置”的意思。

而不激活（取消）一个选项参数，只需要输入：

```
:set no 选项名
```

因此，不激活某个选项参数就是在选项名前加上 `no`。`no` 是英语“不，否”的意思。

有些选项参数需要被指定一个值，像这样：

```
:set 选项名=数值
```

如果想了解一个选项参数的状态，只需要输入：

```
:set 选项名?
```

在选项名后加一个问号（?）即可。

在配置文件中配置选项参数

如果你不喜欢“短暂性”的用命令来配置选项参数的方法，那么可以用一个配置文件，在里面写入所需选项参数的配置。而这也是我们比较推荐的方法，毕竟没有几个人愿意每次打开 Vim 重新配置一次选项参数。

目前在我们的用户家目录还没有 `.vimrc`（注意前面有一点 `.`，表示是隐藏文件）这个 Vim 的配置文件。但是在 `/etc/vim` 目录中有一个文件叫做 `vimrc`（注意前面没有一点 `.`）。

我们把 `/etc/vim` 目录下的 `vimrc` 拷贝到我们家目录中，并且重命名为 `.vimrc`。如下：

```
cp /etc/vim/vimrc ~/.vimrc
```

现在我们来打开 `.vimrc` 这个文件，用什么来打开呢？当然用 Vim 啦。

```
vim .vimrc
```

我们可以看到 `.vimrc` 的文件开头是类似这样的：

```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
" All system-wide defaults are set in $VIMRUNTIME/debian.vim and sourced by
" the call to :runtime you can find below. If you wish to change any of those
" settings, you should do it in this file (/etc/vim/vimrc), since debian.vim
" will be overwritten everytime an upgrade of the vim packages is performed.
" It is recommended to make changes after sourcing debian.vim since it alters
" the value of the 'compatible' option.

" This line should not be removed as it ensures that various options are
" properly set to work with the Vim-related packages available in Debian.
runtime! debian.vim

" Vim will load $VIMRUNTIME/defaults.vim if the user does not have a vimrc.
" This happens after /etc/vim/vimrc(.local) are loaded, so it will override
" any settings in these files.
" If you don't want that to happen, uncomment the below line to prevent
" defaults.vim from being loaded.
" let g:skip_defaults_vim = 1

" Uncomment the next line to make Vim more Vi-compatible
" NOTE: debian.vim sets 'nocompatible'. Setting 'compatible' changes numerous
" options, so any other options should be set AFTER setting 'compatible'.
"set compatible

".vimrc" 61L, 2469C                               1,1                               Top
```

以双引号（"）开头的行是注释，我建议你读一下（如果你英语还可以的话，或者也可以用 Google 翻译），注释还是很重要的。

现在，让我们来激活一些很有用的选项参数。

可以看到一些选项参数的配置行前面有双引号，表示这个选项参数还没有被激活，那么你可以删除行首的双引号来激活这个选项参数。

每个配置选项，你可以先用 `/选项参数` 来搜索，看看 `.vimrc` 文件里是否已经有这个配置选项：

- 可能这个选项参数已经存在也已经被激活（前面没有 "（双引号））。
- 可能这个选项参数已经存在但没有被激活（前面有 "（双引号）），那你可以去掉前面的双引号来激活它。
- 可能这个选项参数还没存在，那你添加即可。

syntax: 配置语法高亮

首先我们来激活语法高亮这个选项参数。这样，Vim 就会根据你打开的文件类型不同，来对文件的文本进行对应的语法高亮。

Vim 支持很多种编程语言类型的文件，例如 C、C++、Python、Java、Ruby、Bash、Perl等等。要激活语法高亮，只需要保证：

```
syntax on
```

这一句前面没有双引号即可，`syntax` 是英语“语法”的意思。

注意：修改了 `.vimrc` 这个配置文件后，须要保存此文件，退出，再重新运行 Vim，以使用最新的配置。

在新版 Ubuntu 中，Vim 默认是开启语法高亮的：

默认地，在 Vim 中搜索文本时，是区分大小写的，比如我搜“just”，那么就搜不到“JUST”，"Just"等等。

如果要 Vim 忽略大小写，那么可以设置：

```
set ignorecase
```

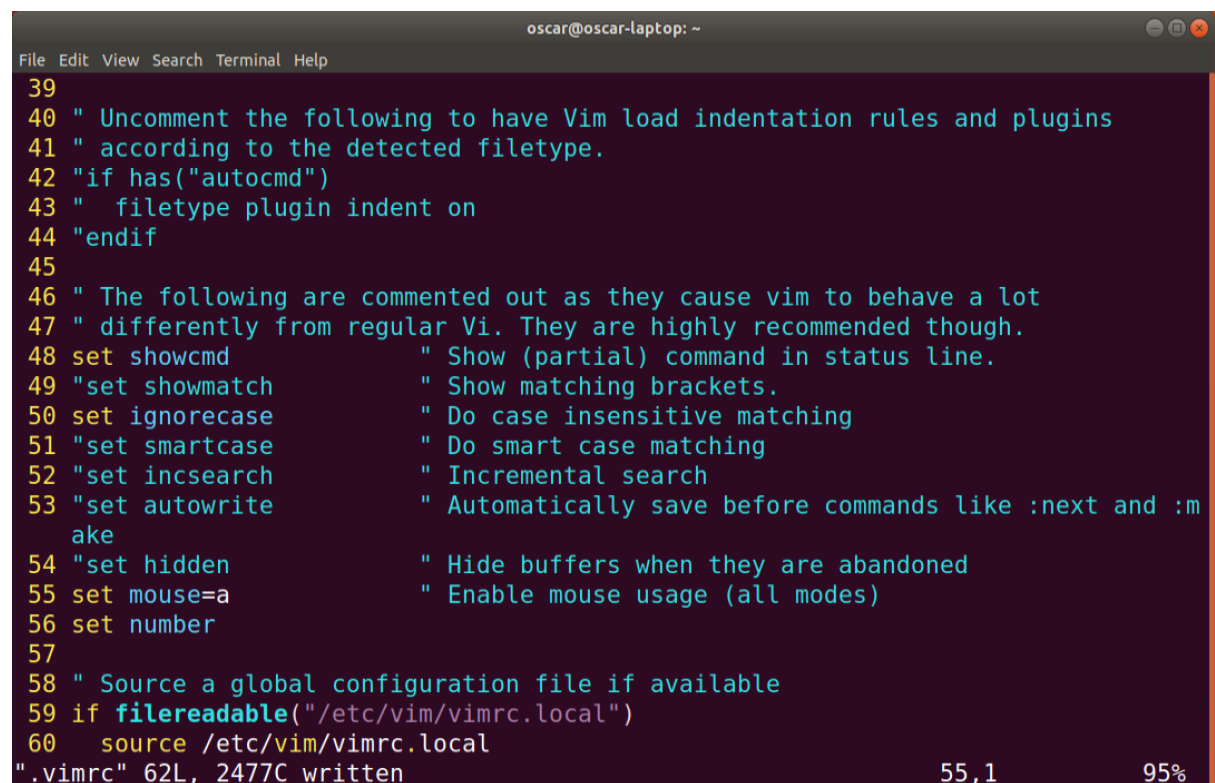
ignore 是英语“忽略”的意思，case 是英语“情况”的意思，因此 ignore case 是“忽略大小写”的意思。

mouse: 鼠标支持

mouse 是英语“老鼠，鼠标”的意思。

是的，即使在终端命令行模式中，仍然可以使用鼠标。激活鼠标很简单，只要：

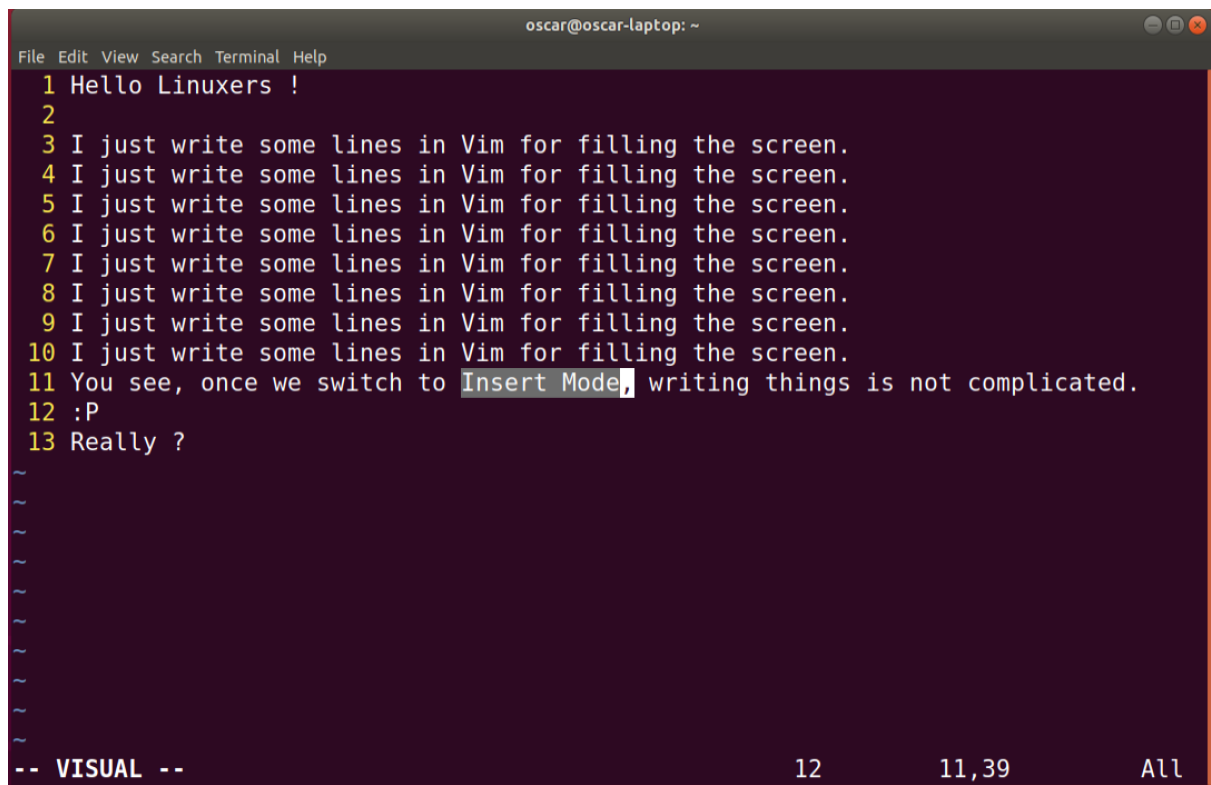
```
set mouse=a
```



```
39
40 " Uncomment the following to have Vim load indentation rules and plugins
41 " according to the detected filetype.
42 "if has("autocmd")
43 "   filetype plugin indent on
44 "endif
45
46 " The following are commented out as they cause vim to behave a lot
47 " differently from regular Vi. They are highly recommended though.
48 set showcmd           " Show (partial) command in status line.
49 "set showmatch         " Show matching brackets.
50 set ignorecase         " Do case insensitive matching
51 "set smartcase         " Do smart case matching
52 "set incsearch         " Incremental search
53 "set autowrite         " Automatically save before commands like :next and :m
ake
54 "set hidden           " Hide buffers when they are abandoned
55 set mouse=a           " Enable mouse usage (all modes)
56 set number
57
58 " Source a global configuration file if available
59 if filereadable("/etc/vim/vimrc.local")
60   source /etc/vim/vimrc.local
".vimrc" 62L, 2477C written                    55,1          95%
```

激活鼠标之后，你就可以用鼠标（或触摸板）来点击文件中任意位置，使光标移动到那里。你甚至可以用鼠标上的滚轮来滚动文件。

你也可以用鼠标来选择一段文本，就会切换到 VISUAL 模式，也就是“可视”模式（visual 是英语“可视的”的意思）。在这个模式下，你可以用 x 来删除刚才选中的文本，用大写的 U 来把文本转为大写，u 来把文本转为小写等等。



```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
1 Hello Linuxers !
2
3 I just write some lines in Vim for filling the screen.
4 I just write some lines in Vim for filling the screen.
5 I just write some lines in Vim for filling the screen.
6 I just write some lines in Vim for filling the screen.
7 I just write some lines in Vim for filling the screen.
8 I just write some lines in Vim for filling the screen.
9 I just write some lines in Vim for filling the screen.
10 I just write some lines in Vim for filling the screen.
11 You see, once we switch to Insert Mode, writing things is not complicated.
12 :P
13 Really ?
~
~
~
~
~
~
~
~
~
~
-- VISUAL --          12          11,39      All
```

经过我们上面一系列的配置，你的 .vimrc 目前大概是类似如下：

```

" All system-wide defaults are set in $VIMRUNTIME/debian.vim and sourced by
" the call to :runtime you can find below. If you wish to change any of those
" settings, you should do it in this file (/etc/vim/vimrc), since debian.vim
" will be overwritten everytime an upgrade of the vim packages is performed.
" It is recommended to make changes after sourcing debian.vim since it alters
" the value of the 'compatible' option.

" This line should not be removed as it ensures that various options are
" properly set to work with the Vim-related packages available in Debian.
runtime! debian.vim

" Vim will load $VIMRUNTIME/defaults.vim if the user does not have a vimrc.
" This happens after /etc/vim/vimrc(.local) are loaded, so it will override
" any settings in these files.
" If you don't want that to happen, uncomment the below line to prevent
" defaults.vim from being loaded.
" let g:skip_defaults_vim = 1

" Uncomment the next line to make Vim more Vi-compatible
" NOTE: debian.vim sets 'nocompatible'. Setting 'compatible' changes numerous
" options, so any other options should be set AFTER setting 'compatible'.
"set compatible

" Vim5 and later versions support syntax highlighting. Uncommenting the next
" line enables syntax highlighting by default.
if has("syntax")
    syntax on
endif

" If using a dark background within the editing area and syntax highlighting
" turn on this option as well
"set background=dark

" Uncomment the following to have Vim jump to the last position when
" reopening a file
"if has("autocmd")
" au BufReadPost * if line("'\"") > 1 && line("'\"") <= line("$") | exe "normal! g'\"" | endif
"endif

" Uncomment the following to have Vim load indentation rules and plugins
" according to the detected filetype.
"if has("autocmd")
" filetype plugin indent on
"endif

" The following are commented out as they cause vim to behave a lot
" differently from regular Vi. They are highly recommended though.
set showcmd " Show (partial) command in status line.
"set showmatch " Show matching brackets.
set ignorecase " Do case insensitive matching
"set smartcase " Do smart case matching
"set incsearch " Incremental search
"set autowrite " Automatically save before commands like :next and :make
"set hidden " Hide buffers when they are abandoned
set mouse=a " Enable mouse usage (all modes)
set number

" Source a global configuration file if available
if filereadable("/etc/vim/vimrc.local")
    source /etc/vim/vimrc.local
endif

```

网上也有很多高手配置的 .vimrc ，可供下载。你只需要稍加修改即可成为自己的 Vim 的配置了。

例如，Github 上就有很多 Linux 用户分享自己的 Vim 配置的，例如这个 Star 数很多的 Vim 配置：
<https://github.com/amix/vimrc> 。

你可以用 Github 的搜索框，搜索“vimrc”来查找：<https://github.com/search?q=vimrc> 。

学完这两课，是否有种“人生苦短，我用 Vim”的感觉？
相信我，随着你用 Vim 来编程，这种感觉会越来越强烈。
当然了，Emacs 也是不错的，和 Vim 不相伯仲。

5. 总结

要用好 Vim，需要记住不少的快捷键，这也是 Vim 入门比较花时间的原因。但一旦过了“磨合期”，那简直快得飞起；

我们可以修改 `.vimrc` 文件来配置 Vim。可以在网上搜索一些 Linux 用户分享的 `vimrc` 文件，稍加修改，成为自己的 Vim 配置。

今天的课就到这里，一起加油吧！

← 39 Vim 岂是池中物，宝剑锋从磨
砺出

41 一入 Shell 深似海，酷炫外壳惹
人爱 →