

The
Pragmatic
Programmers

Pragmatic
Express



tmux

Productive
Mouse-Free
Development



Brian P. Hogan

Edited by Susannah Davidson Pfalzer

目錄

本书简介	0
致谢	1
前言	2
第1章 基础知识	3
第2章 配置 tmux	4
第3章 脚本定制 tmux 环境	5
第4章 文本和缓冲区	6
第5章 使用 tmux 结对编程	7
第6章 工作流	8
附录 配置文件	9

《tmux: Productive Mouse-Free Development》中文翻译

原书封面



PDF 版的电子书请自行搜索下载。

翻译进度

本书已初步翻译完毕，如果您在阅读过程中发现任何不妥，请在 [github](#) 项目中提出 issue，我会尽快解决，感谢您的阅读。

- [x] [致谢](#)
- [x] [前言](#)
- [x] [第1章 基础知识](#)
- [x] [第2章 配置 tmux](#)
- [x] [第3章 脚本定制 tmux 环境](#)
- [x] [第4章 文本和缓冲区](#)
- [x] [第5章 使用 tmux 结对编程](#)
- [x] [第6章 工作流](#)
- [x] [附录 配置文件](#)

翻译贡献

修正、精校：[pityonline](#)

GitHub 地址

[Github 链接](#)

GitBook 地址

[GitBook 链接](#)



致谢 (Acknowledgments)

起初我是打算就把这本书自行出版的，但是之前我和 Pragmatic Bookshelf 的几位朋友有过一段愉快的合作经历，因此我想试试看他们是否还愿意和我合作。十分感谢 Dave 和 Andy 能让我们再次合作。

感谢我的编辑 Susannah Pfalzer，使你现在看到的这本书比之前好得多了。她的指导使得本书能够始终如一地专注于核心内容，让这本书的内容循序渐进，可读性更好。

那几位自愿抽出它们的时间来校正这本书的朋友们让我看到了最有趣、最有思想的批注。感谢他们让本书有了众多美妙的想法。十分感谢 Jeff Holland, Austen Ott, Kevin Gisi, Tony Collen, Harley Stran, Chris Johnson, Drew Neil, Darcy Laycock, Luke Chadwick, Jeff Carley, Marc Harter 和 Nick LaMuro 给出的中肯建议。

我要特别感谢 Chris Warren, Mike Weber, Aaron Godin, Emma Smith, Erich Tesky 和我的其他商业合作伙伴，感谢他们对我的支持，还要感谢 Chris Johnson 让我首次使用了 tmux。

最后，我要极大地感谢我的妻子 Carissa 长久以来对我的不断支持，她辛勤地工作以及对我们女儿的关怀使我能够抽出时间来撰写本书。

前言

你的鼠标正在拖慢你的速度。

鼠标的发明为我们与计算机的交互创造了一种新的方式。我们可以单击、双击、三连击，甚至还能猛击某个应用。鼠标和图形交互界面的产生，使得使用计算机对普通用户来说变得更加容易。但是对于程序员来说，鼠标有很大的负面影响。

编译、构造软件时，我们会使用多个程序一起工作。比如一个 web 开发者可能会同时运行一个数据库控制台，一个 web 服务端和一个文本编辑器。用鼠标在这些窗口之间来回切换不仅会浪费宝贵的时间还会打断你的思路。这看起来似乎没什么，但是你的手需要从键盘移开，再放到鼠标上，然后定位，最后再完成鼠标操作，这个过程非常容易让人分心。

使用 tmux，你可以创建一个如图1（使用 tmux 作为开发环境）所示的工作环境。使用 tmux 的窗口，你可以非常轻松地在一个非常简单的环境中管理文本编辑器、数据库控制台、本地 web 服务器。你还可以把 tmux 窗口分割为多个区域，让多个程序并排显示或运行。这意味着你可以在一个窗口里让文本浏览器，irc 聊天客户端，或自动化测试与你的主编辑器同时显示、运行。

最棒的是，你仅仅通过键盘快捷键就可以非常快速地在这些窗口和面板之间互相移动，这样会极大地提高你的注意力和生产效率。

在这本书中，你可以学到如何配置、使用并自定义 tmux。你会学习到如何同时管理多个程序，编写脚本来创建自定义的环境，还能学会如何使用 tmux 与其他人远程工作。使用 tmux，你可以创建一个几乎纯键盘操作的工作环境。

```

1 <h1>Dashboard#index</h1>
2 <p>Find me in app/views/dashboard/index.html.erb</p>
~
~
~
~
~
~
~
~
~
~
~/devproject/app/views/dashboard/index.html.erb" 2L, 78C

1,1      All
Dashboard#index      Devproject
Find me in app/views/dashboard/index.html.erb

OK
Session: devproject 1 3      1:Main* 2:console 3:server-      31 Jan 13:37
  
```

```

Rails 3.2.1: http://goo.gl/rXbj8 | Rails 3.1.3:
ess.net] has joined #RubyOnRails
13:37 < necromancer> smookil: do you have
resources :pools set up?
13:37 < sdwrage> simple_format
13:37 < sdwrage> is that one too?
13:37 < shinobi> danneu: i took your advice for
that selection box, the params
are coming back correctly, but
it still doesn't select the
item in the hash lol
13:37 < smookil> necromancer: sure
13:37 -!- p0y [~p0y@125.212.56.217] has quit
[Remote host closed the connection]
[13:37] [bphogan_(!)] [2:freemove/#r [Act: 1]
[#rubyonrails]

# No reason given
# ./spec/helpers/dashboard_helper_spec.rb:14
Project requires a name
# Not yet implemented
# ./spec/models/project_spec.rb:4
dashboard/index.html.erb add some examples to
(or delete) /Users/brianhogan/devproject/spec/vi
ews/dashboard/index.html.erb_spec.rb
# No reason given
# ./spec/views/dashboard/index.html.erb_spec
.rb:4
Finished in 0.08671 seconds
4 examples, 0 failures, 3 pending
  
```

图1 - 使用 tmux 作为开发环境

什么是 tmux?

tmux 是一个终端复用器 (terminal multiplexer)。它让我们可以使用单一环境就可以登录多个终端或窗口，每个终端或窗口都运行着独立的进程或程序。例如，我们可以打开 tmux 然后运行 Vim 编辑器。然后可以新建一个窗口运行一个数据库控制台，然后在这些程序之间来回切换，这一切都是在一个会话 (session) 中进行的。

如果你使用了一个现代操作系统并且终端有标签页功能的话，这听起来并不新鲜。但是同时运行多个程序只是 tmux 的特性之一。我们可以将窗口 (window) 划分为水平或垂直面板 (pane)，也就是说可以在同一个屏幕上并排显示或运行两个或多个程序。这些操作都不使用鼠标。

我们还能从一个会话中分离出来，让整个工作环境都在后台运行。如果你以前用过 GNU-Screen，那你对这个特性一定感到很熟悉。tmux 与 GNU-Screen 有许多的相似之处，但是 tmux 的功能更多，而且 tmux 的配置更容易。由于 tmux 使用了 client-server 模型，因此可以在一个中央位置控制窗口和面板，甚至可以从一个终端窗口就实现多个会话之间的切换。这个 client-server 模型还可以让我们创建 tmux 脚本并与其他窗口或应用程序交互。

在本书中，我们会探讨以上所有这些特性等等。

谁应该读这本书

本书的目标是帮助 Mac 或 Linux 程序员在使用终端时更加得心应手。

如果你是一个软件开发人员，你会看到如何使用 tmux 构建一个开发环境，通过它你可以轻而易举地同时使用多个终端会话工作。并且如果你已经能非常熟练地使用 Vim 或 Emacs，你会看到 tmux 是如何进一步提高你的工作效率。

如果你是一个系统管理员或偶尔与远程服务器打交道的开发人员，你可能会对如何利用 tmux 持久地监测服务器状态感兴趣。

本书内容

本书将会向你展示如何把 tmux 融入到你的工作中，包括 tmux 的基本特性以及如何在每天的工作场景中应用这些特性。

在第 1 章，基础知识。你会学到关于 tmux 的基本特性，包括创建会话，面板，窗口以及学习如何执行基本的操作。

在第 2 章，配置 tmux。你会学习到如何重定义许多 tmux 默认的快捷键和外观配置。

在第 3 章，脚本定制 tmux 环境。你会学习到如何编写你自己的开发环境脚本，包括使用命令行界面，配置文件和 tmuxinator 工具。

之后，在第 4 章，处理文本和缓冲区。你会学习到如何通过键盘快捷键在缓冲区之间移动文本，如何选中并复制文本，以及如何工作于多个粘贴缓冲区。

接下来，在第 5 章，使用 tmux 结对编程。你会学习到如何配置 tmux，使你 and 同事可以在不同的计算机上使用 tmux 基于同一份代码一起工作。

最后，在第 6 章，工作流。本章涵盖了更多管理窗口、面板和会话，并向你展示如何使用 tmux 更进一步地高效工作。

你需要什么

要使用 tmux，你需要有一台运行 Mac OS X 或 UNIX/Linux 操作系统的计算机。不幸的是，tmux 不支持 Windows 系统，但 tmux 在 Linux 系统的虚拟机、VPS 或共享主机环境中依然可以良好运行。

尽管不是必须的，使用文本编辑器例如 Vim 或 Emacs 的经验将会对学习使用 tmux 非常有帮助。tmux 的工作机制和它们类似，而且 tmux 有一些预定义的键盘快捷键会让使用过这些编辑器的读者们感到熟悉。

约定

tmux 是一个由键盘驱动的工具。你会在本书中遇到大量的键盘快捷键。由于 tmux 提供了大小写敏感的键盘快捷键，因此你可能会对本书中提到的快捷键感到理解不是很清晰。

为了尽可能地简洁，下面是本书中所使用的一些约定。

- `CTRL - b` 表示“同时按下 `CTRL` 键和 `b` 键”。
- `CTRL - R` 表示“同时按下 `CTRL` 键和 `R` 键（大写 R，你可能需要同时按下 `SHIFT` 键和 `r` 键，我不会在这些按键中特别说明需要按下 `SHIFT` 键）”。
- `CTRL - b d` 表示“同时按下 `CONTROL` 键和 `b` 键，然后松开它们，然后再按下 `d` 键”。
在第 1 章，基础知识，你会学习关于命令前缀（command prefix），使用命令前缀，`CTRL - b d` 可以表示为 `PREFIX d`。
- 最后，我会在本书中展示一些终端命令，比如：

```
$ tmux new-session
```

美元符号只是终端会话的提示符，在终端输入命令时并不需要输入这个符号。

在线资源

本书的网站是一个交互式的论坛，同时你也可以在上面反馈本书的错误之处。网站上有本书中所建立的配置文件和脚本文件，你可以在网站上直接下载这些源代码。

使用 tmux 让我工作效率大增，我也很高兴能和大家分享我得经验。让我们开始吧，我们先来安装 tmux，了解它的基本特性。

第 1 章 基础知识

一旦你熟悉了 tmux 后，它就像一个加速器一样加速你的工作效率。在这个章节你会了解到 tmux 的基本特性，包括使用会话，窗口和面板管理你的程序。这些简单的概念正是组成 tmux 的重要基石，是它们让 tmux 成为开发人员和系统管理员的手中利器。

在学习如何使用这些特性之前，你需要确保 tmux 已被安装。

1.1 安装 tmux

你可以通过两种方式安装 tmux：使用包管理器安装，或者从源码编译构建安装。

无论选择哪种方式，都要确保安装的 tmux 是 1.6 或更新版本，本书使用的是 tmux 1.6 版。早期的版本可能不支持我们将要学习的一些特性。

通过包管理器安装

tmux 可以在多种包管理器中安装。在 OS X 系统，你可以通过 Homebrew 或 Macports 安装 tmux。如何安装包管理器不在本书的范围内，请为你选择的管理器查阅相关资料。无论你选择了哪种包管理器，可能都会用到 Xcode，可以通过 Mac 电脑的随机光盘安装 Xcode，或是通过 Mac App Store 安装。

如果你使用的是 Homebrew，可以通过以下命令安装：

```
$ brew install tmux
```

如果你使用的是 MacPorts，可以通过以下命令安装：

```
$ sudo port install tmux
```

对于 Ubuntu 用户，可以通过以下命令安装：

```
$ sudo apt-get install tmux
```

在终端中执行以下命令来确认 tmux 版本号及安装是否正确：

```
$ tmux -V
```

tmux 1.6

由于包管理器的原因，你可能无法安装最近的 tmux 版本，这意味着你需要通过源码编译安装。下面我们看看如何安装。

从源码安装

在 Mac OS X 与 Linux 编译安装 tmux 的过程是一样的，两者都需要 GCC 编译器。

Mac 用户需要安装 Xcode，可以在随机光盘或 Mac App Store 中获取。

Linux 用户，自带的包管理器往往已经内置了 GCC 编译器。对于 Ubuntu 系统，只需要执行以下命令就可以安装好所有编译器：

```
$ sudo apt-get install build-essential
```

tmux 还依赖 libevent 和 ncurses，请事先安装好。对于 Ubuntu 系统，可以使用以下命令安装：

```
$ sudo apt-get install libevent-dev libncurses5-dev
```

编译器与软件包依赖安装后，下载 tmux 源码，执行以下命令解压并安装：

```
$ tar -zxvf tmux-1.6.tar.gz
$ cd tmux-1.6
$ ./configure
$ make
$ sudo make install
```

你可以执行以下命令验证安装是否成功，该命令会返回当前 tmux 版本：

```
$ tmux -V
```

tmux 1.6

现在 tmux 已经安装完成了，下面我们开始通过一个基础的会话展开对 tmux 核心功能的探索。

1.2 开始使用 tmux

在终端中只要执行以下命令就可以启动 tmux：

```
$ tmux
```

然后在屏幕上会出现一个类似图 2 的界面，这就是一个新的 tmux 会话。使用起来就像在普通终端中一样，在里面执行任何终端命令都和普通终端差不多。

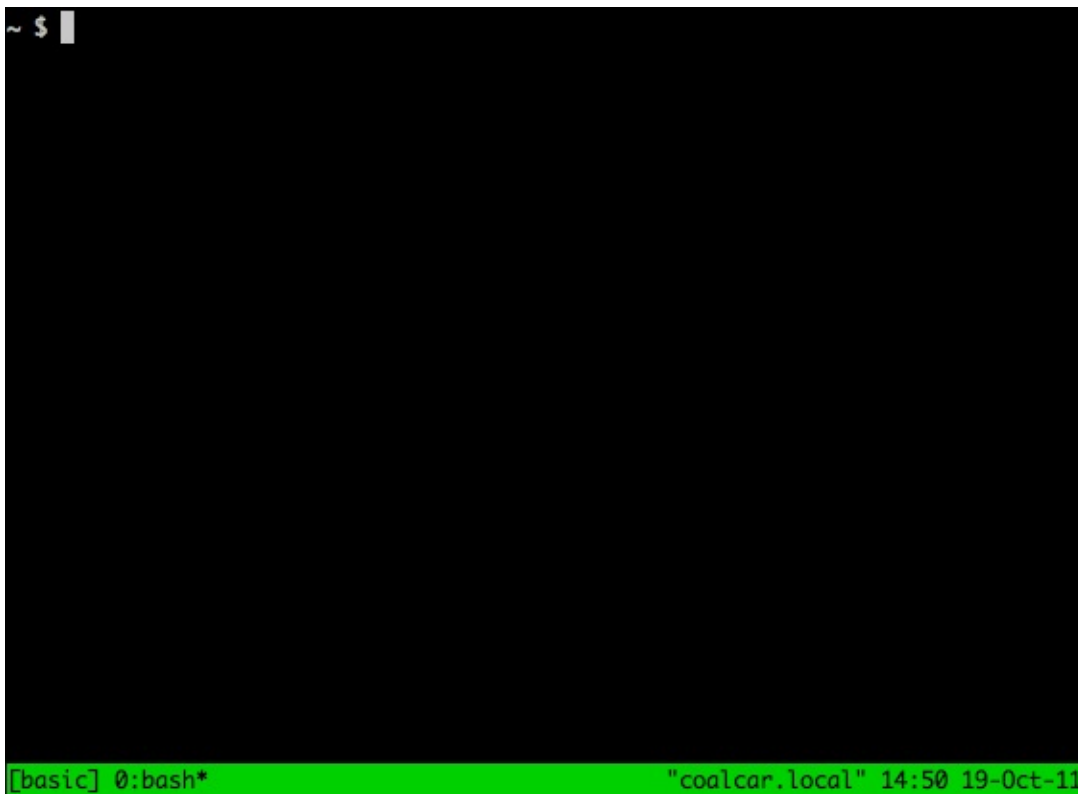


图2 - 新的 tmux 会话

要关闭 tmux 会话，只需在会话中输入：

```
$ exit
```

这样就可以关闭 tmux 并返回到标准终端中。

但是，除非你只是很短暂地用一下 tmux，否则我们不建议你使用这种方式来使用会话。我们在后面会说明如何创建一种“命名会话（named session）”来取代这种方式。

创建命名会话

我们可以在一台计算机上创建多个会话，并且通过为每个会话指定一个唯一的名称来管理它们。我们现在就来创建一个叫做 basic 的会话：

```
$ tmux new-session -s basic
```

这个命令可以简化为：

```
$ tmux new -s basic
```

这种简化命令会在本书的后面一直使用。

执行这条命令后我们会进入到一个全新的会话中，但是我们并不会感到与之前进入 tmux 会话的方式有何不同之处。如果我们输入 `exit` 命令，也会返回到标准终端里。

然而命名会话会在我们想让 tmux 在后台运行而又不想关闭 tmux 时变得非常方便，我们接下来会讨论这部分内容。在继续之前，请输入命令来离开 tmux：

```
$ exit
```

1.3 分离和连接会话

tmux 的一个最大优势就是启动 tmux 环境并执行各种程序或进程时，可以通过从会话“分离（detaching）”让 tmux 在后台运行。

如果我们关闭了一个普通的终端会话，那么这个会话中的所有程序都会被杀死。但是从一个 tmux 会话分离时，实际上并没有关闭 tmux。在这个会话中运行的程序仍然在运行。然后可以在任何想要的时候再“连接（attaching）”上这个会话，你会发现所有的界面（现场）就和你分离会话时一模一样。下面对这个功能的演示中，我们会创建一个命名会话，启动一个程序，然后从这个会话中分离。首先创建这个 basic 会话：

```
$ tmux new -s basic
```

然后，在这个 tmux 会话中执行一个 `top` 命令，它用来监测计算机的内存和 CPU 使用情况，就像这样：

```
$ top
```

现在终端里就有与图3（tmux 中运行的 top 命令）类似的界面了。这时在键盘上同时按下 `CTRL-b` 键，然后再按下 `d` 键，这样就从 tmux 会话中分离出来，返回到标准终端界面里了。在后面我们会学习如何快速地返回到 basic 会话中，但是首先我们来谈谈命令前缀（command prefix）。

```
Processes: 93 total, 2 running, 91 sleeping, 446 threads      22:42:55
Load Avg: 0.66, 0.60, 0.53  CPU usage: 8.96% user, 13.0% sys, 78.2% idle
SharedLibs: 8260K resident, 4848K data, 0B linkedit.
MemRegions: 23559 total, 996M resident, 23M private, 293M shared.
PhysMem: 248M wired, 1204M active, 472M inactive, 1924M used, 122M free.
VM: 222G vsize, 1041M framework vsize, 10823854(1) pageins, 4231683(0) pageouts.
Networks: packets: 41524634/17G in, 34596097/5325M out.
Disks: 10812312/106G read, 12469324/432G written.
update interval[1]: ^Ad
PID  COMMAND      %CPU  TIME    #TH  #WQ  #PORT  #MREG  RPRVT  RSHRD  RSIZE
80624  Finder        0.0   18:39.34  9    3    272    588    20M    34M    32M
68792  ssh-agent     0.0   00:12.05  2    1     33     58    424K    364K   1124K
63634  quicklookd    0.0   00:00.10  6    2     79     71   1936K   4672K  5844K
63626  tmux          0.0   00:00.00  1    0     15     41   348K    1036K  804K
63600  bash          0.0   00:00.06  1    0     17     25   1236K    760K   1936K
63595  top           10.6   00:08.36 1/1    0     27     33   2004K    264K   2584K
63568  bash          0.0   00:00.08  1    0     17     25   1260K    760K   1960K
63567  tmux          0.0   00:00.06  1    0     8      41   496K     1036K  1056K
63511  cupsd         0.0   00:00.07  3    1     37     57   2152K    244K   3428K
63368- GoogleTalkPl 0.0   00:01.28  8    1    214    169   7164K    6236K   12M
63367- PluginProces 0.0   00:00.09  3    1     81     90   1376K    4688K  5464K
62688- TweetDeck     1.6   05:32.94  9    2    189    997   91M      26M    118M
62611  Preview       0.0   00:04.63  2    1    108    177   7564K     27M    23M
[basic] 0:top* "coalcar.local" 22:42 24-Oct-11
```

图3 - 在 tmux 中执行 top 命令

命令前缀

由于我们的程序是在 tmux 环境里运行的，因此需要一种方式来告诉 tmux 当前所输入的命令是为了让 tmux 去执行而不是 tmux 里的应用程序去执行，这就是 `CTRL-b` 组合键的作用。

当我们想要从 tmux 会话中分离时，可以先按 `CTRL-b` 键，然后再按 `d` 键（`d=detach`，译者注）。在执行 tmux 命令时，每次都要先按下这个 `CTRL-b` 组合键，然后再按下 tmux 命令键（如 `d`），因此我们把这个组合键称为命令前缀。有一点非常重要：并不是把前缀键和命令键一起按下，而是先同时按下 `CTRL` 键和 `b` 键，然后松开这两个键，然后再快速地按下要让 tmux 执行的命令键。

在本书的剩余部分中，我们会使用符号 `PREFIX` 来表示命令前缀，比如 `PREFIX d` 命令表示从一个 tmux 会话分离。在第二章，我们会将前缀重新映射为一个更加方便的组合键，但是目前，我们还是使用默认的组合键。

现在，我们来学习如何回到刚才分离的那个 tmux 会话。在这之前，请将你的终端窗口关闭。

重新连接已有会话

我们之前新建了一个 tmux 会话，在会话中运行了一个程序，然后从这个会话中分离出来，最后甚至关闭了终端窗口，但是这个 tmux 会话依然在运行，而 top 命令也没有停止运行。

在一个新的终端窗口里执行以下命令列出当前存在的 tmux 会话：

```
$ tmux list-sessions
```

这个命令可以简化为：

```
$ tmux ls
```

这条命令会展示目前有一个会话正在运行：

```
basic: 1 windows (created Mon Jan 30 16:58:26 2012) [105x25]
```

要想连接到这个会话，可以使用 `attach` 命令。如果目前只有一个会话在运行，可以用下面这个命令来连接它：

```
$ tmux attach
```

这样就可以重新连接到这个 `tmux` 会话中。如果有 2 个或多个 `tmux` 会话在运行，操作会稍微麻烦一些。我们先用命令 `PREFIX d` 从 `basic` 会话中分离出来。

现在，用这个命令创建一个新的 `tmux` 会话并让它在后台运行：

```
$ tmux new -s second_session -d
```

再列出当前已有的会话列表，就会看见类似如下：

```
$ tmux ls
basic: 1 windows (created Mon Jan 30 16:58:26 2012) [105x25]
second_session: 1 windows (created Mon Jan 30 17:49:21 2012) [105x25]
```

用 `-t` 参数加上会话名称来连接到指定的会话，如连接到 `second_session` 这个会话：

```
$ tmux attach -t second_session
```

我们可以用前面提到的命令从一个会话中分离出来，然后再连接到另一个会话里。在后续章节你会学到在活动会话之间切换的其它方式。但是现在，我们先学习如何杀死一个活动会话。

杀死会话

可以在一个会话中使用 `exit` 命令来杀死这个会话，也可以使用 `kill-session` 命令杀死指定会话：

```
$ tmux kill-session -t basic
$ tmux kill-session -t second_session
```

当一个会话中的程序处于挂起状态时这个命令会很有用。

此时，如果再次列出当前已有会话，会得到这样的信息：

```
$ tmux ls
failed to connect to server: Connection refused
```

这是因为目前没有 tmux 会话在运行，因此 tmux 本身也没有运行，所以它无法处理这个请求。

现在你已经学会了创建并使用会话的基础知识了，下面我们来学习如何在一个会话中同时运行多个程序或命令。

1.4 使用窗口

在一个 tmux 会话中同时运行多个命令或同时执行多个程序的情景非常普遍。可以通过窗口（window）来管理它们，窗口的功能就像现代图形终端模拟器或 web 浏览器的标签页。

一个新的 tmux 窗口被创建时，tmux 环境（environment）会为配置一个初始化的窗口。我们可以创建任意多的窗口，而且在分离与连接会话过程中它们会一直存在。

现在我们来创建一个包含 2 个窗口的新会话。第一个窗口只显示终端提示符，而第二个窗口将会执行 top 命令。可以用以下命令来创建一个叫做 windows 的会话：

```
$ tmux new -s windows -n shell
```

命令中的 `-n` 参数可以让 tmux 把第一个窗口命名为 shell，方便我们辨识这个窗口。

接下来再为这个会话添加一个窗口。

创建并命名窗口

要在当前会话创建一个新的窗口，只需要按下 `PREFIX c` 键。这样创建一个窗口会让 tmux 自动把焦点切换到这个新的窗口里来。在这里可以运行其他程序，在这个窗口里我们运行 `top` 命令。

```
$ top
```


我们已经为第一个窗口命名为 `shell`，但是第二个窗口似乎看起来叫做 `top`。由于在创建这个新的窗口时并没有为它命名，因此这个窗口的名称会随着当前运行的程序变化而变化。我们来给这个窗口取个名吧。

要重命名一个窗口，可以通过按下 `PREFIX ,` 键（前缀 + 逗号键，译者注），然后状态栏就会进入重命名窗口的界面。我们将这个窗口命名为 `Processes`。

可以在一个 `tmux` 会话中创建任意多个窗口。但是一旦创建了 2 个窗口以上，就必须学会如何在窗口之间切换（`move`）。

在窗口之间切换

我们已经在 `tmux` 环境中创建了两个窗口，有多种方法可以在这些窗口之间来回切换。如果只有两个窗口，按下 `PREFIX n` 键（`n=next`，译者注）在已打开窗口之间来回切换。

可以按下 `PREFIX p` 键（`p=previous`，译者注）切换到上一个窗口。

`tmux` 的窗口默认有一个编号，从 0 开始计数。按下 `PREFIX 0` 键快速切换到第 1 个窗口，按下 `PREFIX 1` 键切换到第 2 个窗口。从 0 开始计数的窗口数组并不是一成不变的，在第 2 章，你会学到如何让窗口编号不是从 0 开始，而是从 1 开始。

如果窗口超过了 9 个，可以按下 `PREFIX f` 键（`f=find`，译者注）通过窗口的名称来查找一个窗口（如果窗口已被命名），或者按下 `PREFIX w` 键（`w=window`，译者注）显示一个可视化的窗口列表，然后再选择其中想要的那个窗口。

从这里，我们可继续创建新的窗口并运行程序。从会话中分离出来后，那些窗口会在下次连接时保持依然保持断开连接时候的环境。

如果要关闭一个窗口，可以在命令提示符后输入 `exit`，或者可以按下 `PREFIX &` 键（`&=et=exit`，译者注），后者会在状态栏给出一个确认信息询问是否要关闭当前窗口。如果确认并关闭了当前窗口，那么当前窗口的焦点就会转移到上一个窗口。要想完全退出一个 `tmux` 会话，必须要关闭所有窗口。

在会话中创建窗口这个功能已经非常了不起了，但是我们还能进一步让 `tmux` 把一个窗口分割成多个面板（`pane`）。

1.5 使用面板

你或许认为能够在不同的窗口之间运行不同的程序已经非常棒了。但是 `tmux` 还能把窗口分割成多个面板。

下面我们就来创建一个名为 `panes` 的会话来演示面板的工作机制。

```
$ tmux new -s panes
```

窗口可以以垂直方向或水平方向分割。我们先把窗口在垂直方向分割，然后再在水平方向分割，这样就能得到一个窗口，左边一半是一个大的面板，右边一半是两个并排的小一些的面板，就像图4（一个被切分为多个面板的 tmux 会话）那样。

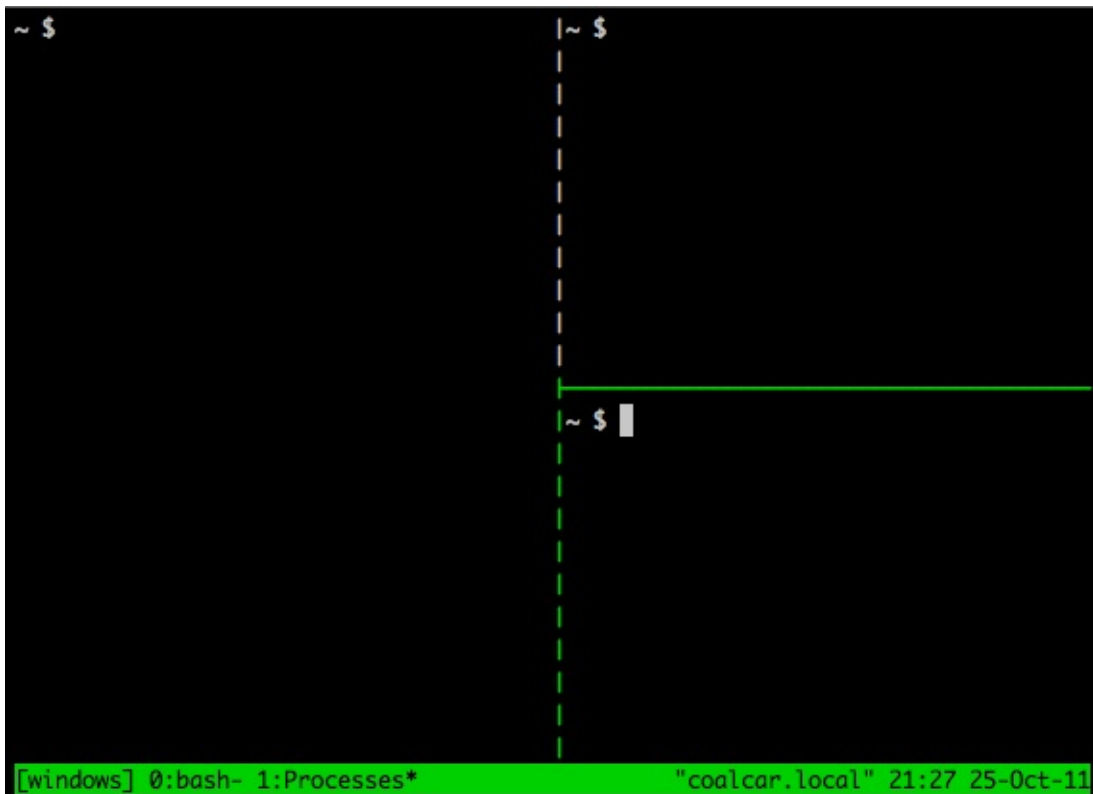


图4 - 一个被切分为多个面板的 tmux 会话

在 tmux 会话中，按下 `PREFIX %` 键（%形似左右各一半，译者注），当前的窗口就会在中间部分从上到下一分为二，在右边的面板里新建第二个面板，而且当前的焦点也会转移到新的面板中。

再按下 `PREFIX "` 键（双引号，译者注），会把这个新的面板再水平地分割为两半。tmux 的默认配置会使得在分割面板时，分割出的两个面板会各占 50% 大小。

要想在这些面板之间来回切换，可以按下 `PREFIX o` 键（o形似循环，译者注）。还可以通过使用 `PREFIX` 前缀键，后面跟随 `UP`、`DOWN`、`LEFT` 或 `RIGHT` 键（上、下、左、右箭头，译者注）在这些面板之间上、下、左、右移动焦点。

通过几次按键，我们已经把一个窗口分割成了带有 3 个面板的工作空间。我们接下来看看如何重新调整这些面板的布局（layout）。

面板布局

可以通过逐步调整或使用模版来调整一个面板的大小。使用默认的快捷键来逐步调整面板的大小不得不说让人觉得恶心。在第 2 章，我们会定义一些调整面板大小更加便捷的快捷键。但是目前，我们还是会使用 tmux 的几个默认面板布局中的一个：

- `even-horizontal` 把所有面板均匀地水平排列，从左到右。
- `even-vertical` 把所有面板均匀地垂直排列，从上到下。
- `main-horizontal` 在顶部创建一个非常大的面板，其余面板变为小面板水平地放在底部。
- `main-vertical` 在左侧创建一个非常大的面板，其余面板变为小面板垂直地放在右侧。
- `tilled` 把所有面板大小均等地在屏幕上显示。

可以通过按下 `PREFIX SPACEBAR` 键（空格键，译者注）来依次轮回使用这些面板布局。

关闭面板

关闭一个面板和关闭终端会话或一个 `tmux` 窗口的方法相同：只需要在面板里输入 `exit` 命令。还可以通过按下 `PREFIX X` 键（大写 `X` 键，译者注）杀死一个面板，如果当前窗口只有这一个面板的话，它还会同时关闭这个窗口。

在杀死指定的面板时，你会收到确认提示。如果某个面板被阻塞了，或者你再也用不着这个面板了，不要犹豫，杀死这个面板。

目前为止，你应该已经学会了如何创建新的会话、窗口和面板，并且能在它们之间来回切换了。在继续深入之前，我们先来了解一些 `tmux` 命令。

1.6 使用命令模式

到现在，我们已经可以使用快捷键来创建 `tmux` 窗口和面板了，这些快捷键都只是 `tmux` 预先定义好的一些组合键。有两种方法来执行 `tmux` 命令：从终端本身执行，或者在 `tmux` 状态栏的“命令区域（command area）”执行命令。在第 3 章我们会学到如何在终端里执行 `tmux` 命令。现在我们来学习如何使用 `tmux` 的命令模式来创建一些新的窗口和面板。

进入命令模式，你需要在一个正在运行的会话里按下 `PREFIX :` 键（冒号键，译者注）。按下快捷键之后状态栏的颜色会改变，并且你会看到一个命令提示符提示你可以输入命令。我们现在就来使用 `new-window` 命令创建一个新的窗口，就像这样：

```
new-window -n console
```

使用这个命令而不是使用快捷键来创建新的窗口的好处就是在创建窗口的同时我们附加一个 `-n` 参数给它取个名字。我们现在更进一步，创建一个窗口并让它执行 `top` 命令。在命令模式里输入这个命令：

```
new-window -n processes "top"
```

当我们敲下 `Enter` 时，新窗口就会显示出来并且可以看到 `top` 命令也在运行，显示了当前计算机正在运行的进程。

在创建 tmux 窗口时给它指定一个初始化的进程是非常便捷的，但是当按下 `q` 键关闭 `top` 程序时，这个 tmux 窗口也会被一起关闭。如果你想要让 `top` 程序关闭后窗口依然存在，可以通过配置文件来搞定这个事情，但是最简单的办法莫过于在创建窗口时不要给它指定一个初始化的程序，在创建完窗口之后再执行你想要的命令。

tmux 可以使用命令模式来创建新的窗口，面板和会话，甚至设置一些其它的环境变量配置。在第 2 章，我们会定制一些快捷键来让这些命令用起来更方便、简单。

1.7 接下来做什么？

在本章，我们学习了关于 tmux 会话、面板、窗口和命令的基本用法，但是还有很多知识我们没有提到。

按下 `PREFIX ?` 键（问号键，译者注），你可以看到一个关于 tmux 预定义的快捷键列表，以及这些快捷键所绑定的命令。

使用 tmux 时，你应该多想想如何为你的工作创建不同的 tmux 环境。如果你在监控服务器，可以使用 tmux 的面板来创建一个监控面板，让它可以同时显示多个监测脚本和日志文件。

经过之前的学习，我们总结了前面出现的一些命令配置，方便以后查阅。

以备查阅

创建会话

命令	描述
<code>tmux new-session</code>	创建一个未命名的会话。可以简写为 <code>tmux new</code> 或者就一个简单的 <code>tmux</code>
<code>tmux new -s development</code>	创建一个名为 <code>development</code> 的会话
<code>tmux new -s development -n editor</code>	创建一个名为 <code>development</code> 的会话并把该会话的第一个窗口命名为 <code>editor</code>
<code>tmux attach -t development</code>	连接到一个名为 <code>development</code> 的会话

会话、窗口和面板的默认快捷键

快捷键	功能
<code>PREFIX d</code>	从一个会话中分离，让该会话在后台运行。
<code>PREFIX :</code>	进入命令模式
<code>PREFIX c</code>	在当前 tmux 会话创建一个新的窗口，是 <code>new-window</code> 命令的简写
<code>PREFIX 0...9</code>	根据窗口的编号选择窗口
<code>PREFIX w</code>	显示当前会话中所有窗口的可选择列表
<code>PREFIX ,</code>	显示一个提示符来重命名一个窗口
<code>PREFIX &</code>	杀死当前窗口，带有确认提示
<code>PREFIX %</code>	把当前窗口垂直地一分为二，分割后的两个面板各占 50% 大小
<code>PREFIX "</code>	把当前窗口水平地一分为二，分割后的两个面板各占 50% 大小
<code>PREFIX o</code>	在已打开的面板之间循环移动当前焦点
<code>PREFIX q</code>	短暂地显示每个面板的编号
<code>PREFIX x</code>	关闭当前面板，带有确认提示
<code>PREFIX SPACE</code>	循环地使用 tmux 的几个默认面板布局

第 2 章 配置 tmux

tmux 默认的快捷键对我们来说并不友好。许多重要而且有用的功能都使用了一些很难操作的组合键或是冗长的命令字符串。而且 tmux 默认的配色方案眼睛看起来也不舒服。在本章节中，我们会为 tmux 构建一个基本的配置文件并在本书的剩余章节一直使用。首先我们会从定制屏幕导航以及创建、调整面板大小开始，然后讨论如何处理一些更高级的设置。确保你的终端配置正确，这样我们设置的一些 tmux 外观属性也能在你的屏幕上正常显示。完成这些之后，你就会对 tmux 的可扩展性有了更深的理解，然后你可以定制一个你独有的 tmux。下面，我们先从如何配置 tmux 开始。

2.1 介绍 .tmux.conf 文件

在默认情况下，tmux 会在两个位置查找配置文件。首先查找 `/etc/tmux.conf` 作为系统配置，然后在当前用户的主目录下查找 `.tmux.conf` 文件（`~/.tmux.conf` 优先级更高，译者注）。如果这两个文件都不存在，tmux 就会使用默认配置。我们并不需要使用系统配置，所以只需要在主目录下创建一个新的配置文件即可（即 `~/.tmux.conf` 文件）。命令如下：

```
$ touch ~/.tmux.conf
```

在这个文件里可以做任何想让 tmux 做的事情，比如定义新的快捷键，配置一个包含多个窗口、面板，运行着程序的 tmux 默认环境等。我们先从一些基本的选项做起，令 tmux 使用更简单。

重新映射大写锁定键（**CAPS LOCK** 键，译者注）

在很多键盘上，CAPS LOCK 键就在 a 键旁边。如果把这个键映射为 CTRL 键，你会在使用快捷键时更加方便。

在 OS X 系统中，可以在偏好设置的键盘选项面板里重新映射 CAPS LOCK 键。你只要按下 Modifier Keys 按钮然后把 CAPS LOCK 键的动作变为 Control。

在 Linux 系统中，这个过程会因你的操作系统而变得稍有些复杂，不过你可以在 Emacs 的维基百科中找到办法。

这个小小的改变为你节省的时间远超你的想象。

[Emacs 维基百科](#)

定义更方便的前缀键

你在之前已经知道 tmux 默认使用 `CTRL-b` 键作为它的命令前缀键。许多使用过 GNU-Screen 的 tmux 用户都是使用 `CTRL-a` 键作为命令前缀键，使用 `CTRL-a` 键是个非常不错的选择因为它更容易同时按下，尤其是如果你把 `CAPS LOCK` 键重新映射为 `CTRL` 键之后，`CTRL-a` 就更容易按下了。它能让你的手常位于键盘的主按键区。

在 `.tmux.conf` 文件里，我们使用 `set-option` 命令来设置选项，可以缩写为 `set`。下面我们就通过命令重新定义命令前缀键：

```
set -g prefix C-a
```

在这个例子里，我们使用了 `-g` 选项，也就是全局配置（global），它能让设置的选项在所有创建的 tmux 会话里生效。

尽管不是必须的，我们可以通过 `unbind-key` 命令或 `unbind` 命令移除之前绑定的组合键。可以在配置文件里输入以下内容来释放 `CTRL-b` 组合键：

```
unbind C-b
```

tmux 并不会实时地自动从配置文件读取你所做的修改。因此如果你在使用 tmux 的过程中修改了 `.tmux.conf` 文件，要想让所做的配置修改生效的话，你需要关闭所有的 tmux 会话，或者在 tmux 命令模式输入 `PREFIX : 命令` 然后输入以下内容：

```
source-file ~/.tmux.conf
```

现在就可以使用 `CTRL-a` 键作为命令前缀键了。在剩余章节中，我们还是会继续把它称为 `PREFIX` 键。

修改默认延时

当我们向 tmux 发送命令时，tmux 增加了一个小小的延时（也就是在松开 `PREFIX` 键和按下命令键之间的时间，译者注），这个延时是很有可能妨碍其它程序的运行，比如 Vim 编辑器。可以通过设置这个延时而让 tmux 响应地更快。在配置文件中增加下面的内容，将延时设置为 1ms：

```
set -sg escape-time 1
```

当重新载入配置文件后，我们就可以向 tmux 更快地发送快捷键了。

设置窗口和面板索引

在第一章里，我们讨论了窗口以及在一个会话里创建多个窗口时如何通过索引切换当前的窗口。这个索引是从 0 开始的，感觉似乎有点恶心（程序员貌似还比较习惯从 0 开始的索引 =。=，译者注）。通过在配置文件里添加下面的内容可以让窗口的索引从 1 开始：

```
set -g base-index 1
```

也就是说，我们可以使用 `PREFIX 1` 快捷键来切换到第一个窗口，而不是 `PREFIX 0`。

还可以通过 `pane-base-index` 选项来设置面板的初始索引。我们把下面这行添加到配置文件中，好让窗口和面板的初始索引保持一致：

```
setw -g pane-base-index 1
```

到目前为止，我们已经学会了使用 `set` 命令来配置 tmux 会话。如果需要配置与窗口进行交互的选项，需要使用 `set-window-option` 命令，可以简写为 `setw`。在本书中，为了让配置的例子能够在一行内显示，因此我使用的都是命令的简写，你在使用简写时要特别小心，因为一不小心，你就有可能把 `set` 和 `setw` 搞混了。

现在我们再来创建几个更有用的快捷键来加速你的操作。

2.2 定制键、命令和用户输入

tmux 里的许多默认快捷键无论是在生理上还是在心理上来说都有些过于延展了。拿 `PREFIX %` 键来说，它不光在键盘上不好按，而且如果不看命令参考的话，你几乎记不住它是用来做什么的。

在本节中，我们将会定义或者重定义一些最常用的 tmux 命令。我们先从自定义一个重新加载 tmux 配置文件的快捷键开始。

创建重新加载配置的快捷键

每次我们对配置文件做修改之后，要想让新的配置文件生效，要么关掉所有的 tmux 会话然后重新打开它，要么就要在 tmux 会话里发送一个命令来重新加载配置文件。现在我们来自定义一个快捷键来让它重新加载配置文件。

我们使用 `bind` 命令来定义新的快捷键。先指定要按下的键，后面跟着它要执行的命令。对于我们的第一个快捷键，我们设置为 `PREFIX r`，让它重新加载当前 tmux 会话的

`.tmux.conf` 配置文件，就像这样：

```
bind r source-file ~/.tmux.conf
```


尽管上面的命令里没有 `PREFIX`，但是在使用 `bind` 命令定义快捷键之后，还是需要在实际中先按下 `PREFIX` 键，然后再按下 `r` 键。虽然我们刚才自定义了重新加载配置文件的快捷键，但是在新的配置文件被加载前我们还是不能使用它，因此还需要再使用一次 `PREFIX`：快捷键进入命令模式，然后输入以下命令重新加载配置文件：

```
source-file ~/.tmux.conf
```

重新加载配置文件后，tmux 并不会提示配置是否有所改变，可以使用 `display` 命令让 tmux 在状态栏输出一个消息。我们修改一下刚才定义的快捷键，让它能够在配置文件加载后显示一个消息“Reloaded!”：

```
bind r source-file ~/.tmux.conf \; display "Reloaded!"
```

通过在多个命令之间添加 `\;` 符号可以使一个键可以绑定执行多个命令。通过刚才定义的快捷键，我们就可以在修改配置文件后按下 `PREFIX r` 键让新的配置快速生效。

提问：我能定义一个不需要前缀的快捷键吗？

当前可以！在 `bind` 命令后面添加 `-n` 参数就可以通知 tmux 这个快捷键不需要按下前缀键。例如：

```
bind-key -n C-r source-file ~/.tmux.conf
```

通过上面的配置，你就可以使用 `CTRL-r` 键来重新加载配置文件了。

但是，这么做会让 tmux 会话里的任何程序或命令都禁用组合键，所以使用这种做法时你要特别小心。

发送前缀键到其它程序

我们在前面把命令前缀键重新映射到了 `CTRL-a` 键，但是例如 Vim、Emacs 甚至是 Bash 终端也会经常用到这个组合键。我们需要配置 tmux，把这个组合键发送给需要的程序中。可以定义一个快捷键来发送 `send-prefix` 命令，就像这样：

```
bind C-a send-prefix
```

在配置生效后，你只需要按两次 `CTRL-a` 键就可以把 `CTRL-a` 命令发送给 tmux 里的程序了。

分割面板

tmux 里分割面板的默认键比较难记，所以我们来重新定义为易记的快捷键。我们把水平分割定义为 `PREFIX |` 键，把垂直分割定义为 `PREFIX -` 键：

```
bind | split-window -h
bind - split-window -v
```

乍一看，你可能觉得这两个键搞反了。`split-window` 命令的 `-v` 参数和 `-h` 参数分别代表“垂直（vertical）”分割和“水平（horizontal）”分割，但是对于 `tmux` 来说，垂直分割表示在当前面板之下创建一个新的面板，所以垂直分割之后的两个面板会上下叠在一起。而水平分割表示在当前面板旁边创建一个新的面板，因此两个面板时左右并排显示在屏幕上的。因此，要想垂直地分割窗口，就要使用“水平”分割；要想水平地分割窗口，就要使用“垂直”分割。

使用新的快捷键会让我们联想起来更加符合视觉感受。如果想要分割窗口，只需现在脑袋里想象你要把窗口分割成什么样子然后按下快捷键。

重新映射移动键

使用 `PREFIX o` 键在面板之间移动有点让人讨厌，但是要是使用箭头键（上下左右键，译者注）就意味着必须把手从键盘的主操作区拿开。如果你使用过 `Vim` 编辑器，你可能会比较熟悉使用 `h`，`j`，`k` 和 `l` 键来回移动。可以把 `tmux` 里的移动键映射为 `Vim` 的操作方式。

```
bind h select-pane -L
bind j select-pane -D
bind k select-pane -U
bind l select-pane -R
```

另外，还可以定义 `PREFIX CTRL-h` 键和 `PREFIX CTRL-l` 键在窗口之间循环切换：

```
bind -r C-h select-window -t :-
bind -r C-l select-window -t :+
```

如果你已经把 `CAPS LOCK` 键重新映射到 `CTRL` 键，你现在就可以在窗口和面板之间来回切换而不必把手移出键盘的主操作区。

调整面板大小

要调整面板大小，可以进入命令模式然后输入 `resize-pane -D` 命令让一个面板向下调整一行的距离。我们可以通过在调整方向的后面添加一个数字来指定要调整的大小，比如 `resize-pane -D 5`。但是这个命令有些啰嗦，我们来定义几个快捷键使得调整面板大小更方便些。

我们在之前使用了 `Vim` 的移动键来调整窗口大小。现在我们来使用 `PREFIX H`，`PREFIX J`，`PREFIX K` 和 `PREFIX L` 键调整面板的尺寸：

```
bind H resize-pane -L 5
bind J resize-pane -D 5
bind K resize-pane -U 5
bind L resize-pane -R 5
```

注意，上面的配置使用的是大写字母。tmux 是对大小写敏感的，大写字母和小写字母分别表示不同的快捷键。你需要使用 `SHIFT` 键来触发大写字母的快捷键。

使用这样的移动键会帮助我们始终跟随窗口移动的方向。比如，如果有一个分割为两个水平面板的窗口，就像这样：

```
-----
|               |
|   Pane 1      |
|               |
|-----|
|               |
|   Pane 2      |
|               |
|-----|
```

如果想要增加 Pane 1 的大小，那么就可以把光标移动到 Pane 1 里然后按下 `PREFIX J` 键，那么水平分割线就会向下移动。如果按下 `PREFIX K`，就会让水平分割线向上移动。

我们通过逐渐增加的方式来调整面板大小，也就是说每次想要调整面板大小的时候，就需要先按下前缀键。但是如果使用了 `-r` 参数，那么就可以让这个快捷键变为“可重复的（repeatable）”，这意味着只需要按下前缀键一次，然后就可以在最大重复限制范围内持续地按下定义的命令键。修改后的配置如下：

```
bind -r H resize-pane -L 5
bind -r J resize-pane -D 5
bind -r K resize-pane -U 5
bind -r L resize-pane -R 5
```

默认的最大重复限制为 500 毫秒，可以通过设置 `repeat-time` 选项把这个时间改为更大的数值。

现在我们把注意力转移到 tmux 如何与鼠标配合使用。

处理鼠标

虽然 tmux 的设计目标是纯键盘操作，但是你会发现有时候使用鼠标更加方便。如果你的终端配置支持使用鼠标的向前单击和滚动切换程序，那么你就可以告诉 tmux 如何处理这些鼠标事件。

有时你可能需要使用鼠标滚轮在终端的缓冲区里向上滚屏，或者你刚开始使用 tmux 时想用鼠标选择窗口和面板。要想在 tmux 里使用鼠标，需要打开鼠标模式：

```
setw -g mode-mouse on
```

还可以配置 tmux，让它能够使用鼠标选择一个面板、调整面板大小或者让我们在窗口列表里选择一个窗口。我们需要配置与刚才类似的配置，就像这样：

```
set -g mouse-select-pane on
set -g mouse-resize-pane on
set -g mouse-select-window on
```

把这些配置添加到配置文件里非常方便，但是你要记住，在 tmux 里使用鼠标会让你的操作速度变慢。尽管使用滚屏和单击功能似乎是个好主意，但是你还是应该学会通过相应的键盘操作切换面板或是在缓冲区里向前或向后移动。所以，在我们的配置文件里会把鼠标选项禁用。可以像这样明确地指明：

```
setw -g mode-mouse off
set -g mouse-select-pane off
set -g mouse-resize-pane off
set -g mouse-select-window off
```

或是简单地禁用全部鼠标选项：

```
setw -g mode-mouse off
```

注：在新的版本里这一块被重构了，所以从 2.1 开始只有一个开头，之前的那些开头都没有用了。如下：

```
set -g mouse on
```

这样设置可以防止不小心使用鼠标选中了终端窗口而导致的误操作，而且它会使我们更加专注于键盘操作。

tmux 提供的高扩展性配置系统可以高度自定义我们与界面交互的方式，还可以配置它的外观，让它看起来更舒适一些，而且在一些情境中，它还能展示给我们更多的信息。

2.3 视图风格

tmux 提供了相当多的方法来定制外观。在本节中，我们会讨论如何配置这些选项，包括配置状态栏和其它面板。我们先配置不同的颜色，然后把乏味的状态栏变成能展示重要信息的工具。

配置颜色

为了让 tmux 具有最佳的视觉体验，首先要确保终端和 tmux 都运行在 256 色模式里。我们先来配置终端。

可以使用一个简单的 Perl 脚本来测试终端的色彩模式。命令如下：

```
$ wget http://www.vim.org/scripts/download_script.php?src_id=4568 -O colortest
$ perl colortest -w
```

如果你的配置正确的话，你应该看到终端类似于图5（终端正确显示 256 色）所示。

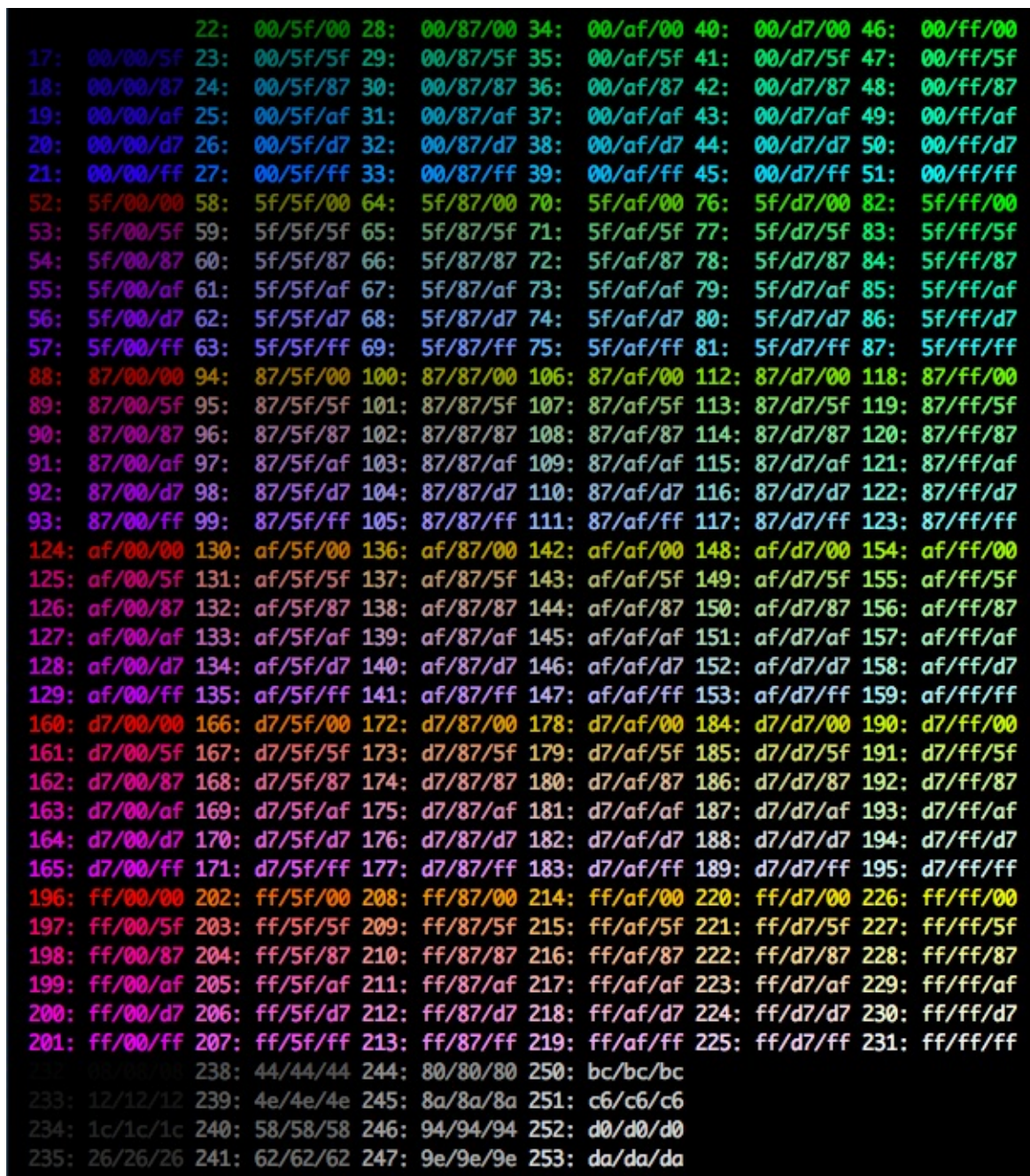


图5 - 终端正确显示 256 色

如果你使用的是 Linux 系统，你可能需要把下面内容添加到你的 `~/.bashrc` 文件里才能运行一个 256 色的终端：

```
[ -z "$TMUX" ] && export TERM=xterm-256color
```

这个条件语句会确保 `TERM` 变量只在 tmux 外运行，因为 tmux 会自己设置它所在的终端。

如果你使用的是 Mac，你应该知道在雪豹系统里终端只会显示 16 色。你需要另外安装一个终端，比如 iTerm2 才能得到全色彩支持。

如果无法看见颜色的正确编号，你需要配置你的终端使用 xterm 的 256 色模式。在 iTerm2 终端里，你可以编辑默认的配置文件找到这个选项，然后把终端模式设置为 xterm-256color，就如图6（为 tmux 配置 iTerm2）所示：

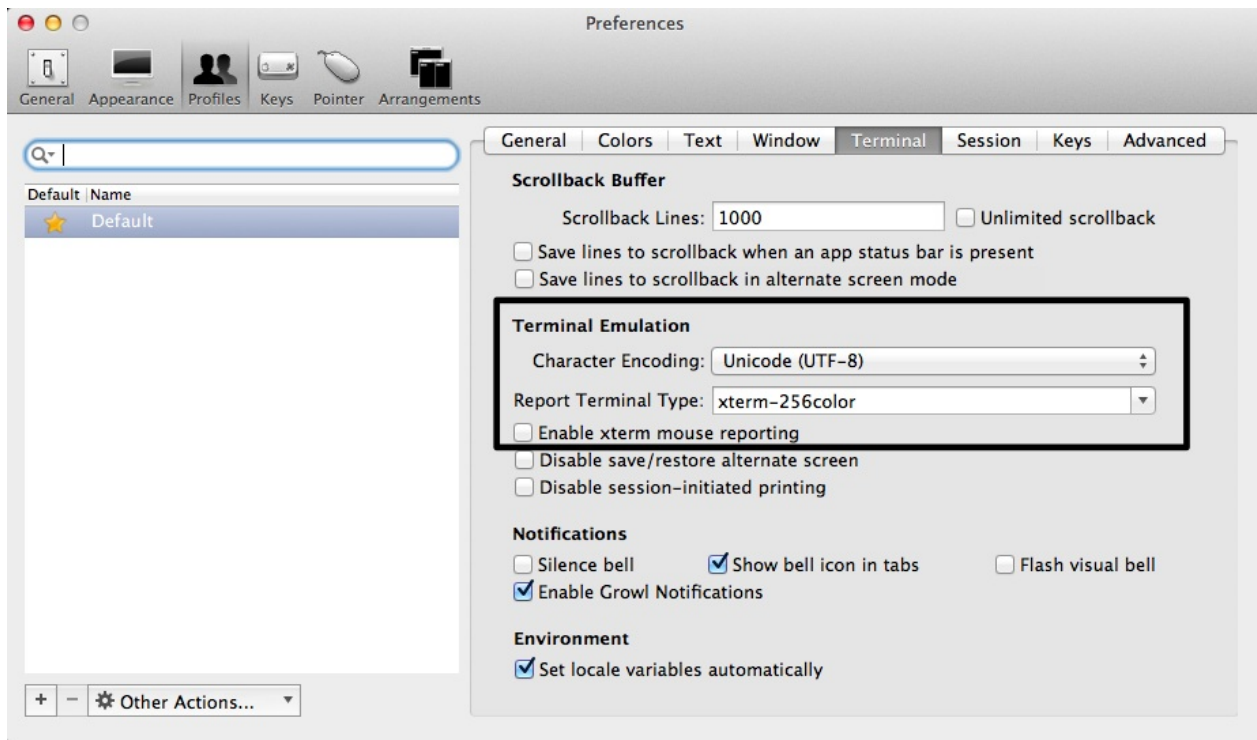


图6 - 为 tmux 配置 iTerm2

另外，你还要保证你的终端模拟器支持显示 UTF-8 字符，只有这样才能让譬如面板分割线的可视元素显示为虚线。

为了让 tmux 以 256 色模式显示内容，需要把下面内容添加到我们的 `.tmux.conf` 文件里：

```
set -g default-terminal "screen-256color"
```

当色彩配置正确配置后，你会发现在 tmux 里运行例如 Vim 之类的程序更加方便了，尤其是如果语法高亮使用了更加丰富的配色方案。你可以在图7（Vim 里 16 色和 256 色对比）里看出差别来。现在就可以来配置 tmux 组件的外观了，我们先从配色开始。

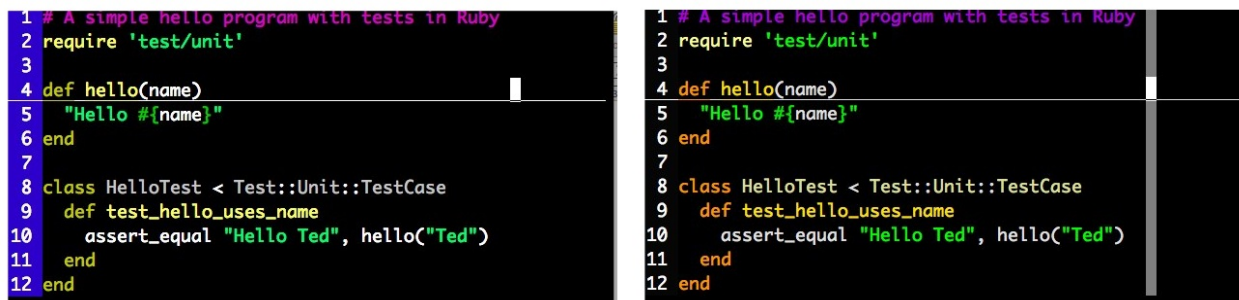


图7 - Vim 里 16 色和 256 色对比

改变配色

我们可以改变 tmux 交互界面的多个部分的颜色，包括状态栏，窗口列表，命令区域甚至是面板分割线。

tmux 提供了特定颜色的变量，包括 `black`（黑，译者注），`red`（红，译者注），`green`（绿，译者注），`yellow`（黄，译者注），`blue`（蓝，译者注），`magenta`（品红，译者注），`cyan`（蓝绿，译者注），或 `white`（白，译者注）。我们可以使用在 256 色调色板中的 `colour0` 到 `colour255`。如果你再看看 `colortest` 程序的输出结果，你会看到这些颜色的编号。你还可以运行这个简单的 shell 脚本来选择你需要的颜色：

```
for i in {0..255} ; do
    printf "\x1b[38;5;${i}mcolour${i}\n"
done
```

tmux 有特定的配置选项可以更改每个组件的前景色和背景色。我们先从定制状态栏的颜色开始探索。

改变状态栏的颜色

状态栏的默认配色是在亮绿色的背景上显示黑色字体。这看起来有些太乏味了。我们让它变成默认为黑色背景上显示白色的字体。

使用 `status-bg` 和 `status-fg` 选项来设置状态栏的背景色和前景色，可以像这样配置：

```
set -g status-fg white
set -g status-bg black
```

在后面我们会定制状态栏内条目（items）的颜色。现在先来配置窗口列表的颜色。

改变窗口列表配色

我们想让当前活动的窗口显示的更加明显可以设置当前活动窗口的颜色为红色，不活动窗口的颜色为蓝绿色。使用 `set-window-option` 选项来配置普通窗口（regular window）的样式，就像这样：

```
setw -g window-status-fg cyan
setw -g window-status-bg default
setw -g window-status-attr dim
```


可以使用 `default` 作为一个默认值那么这个值就会继承状态栏的颜色。要配置活动窗口（active window）的样式，可以使用相似的配置：

```
setw -g window-status-current-fg white
setw -g window-status-current-bg red
setw -g window-status-current-attr bright
```

这样就定制了窗口列表的颜色，还可以定制面板分隔符（pane dividers）的颜色。

改变面板分隔符配色

可以指定面板分隔符的颜色，更棒的是，还可以通过定义颜色让当前活动面板变得更显眼，就像图8（活动面板）所示。



图8 - 活动面板

面板也有前景色和背景色之分，所以我们设置相应的变量：

```
set -g pane-border-fg color
set -g pane-border-bg color

set -g pane-active-border-fg color
set -g pane-active-border-bg color
```

一个面板的前景色就是组成边界的虚线颜色。默认的背景色是黑色，如果让它标记出当前活动面板，就可以让活动面板显得相当突出：

```
set -g pane-border-fg green
set -g pane-border-bg black
set -g pane-active-border-fg white
set -g pane-active-border-bg yellow
```

在修改状态栏前，我们先来润色一下 tmux 命令行。

定制命令行

在 tmux 的命令模式输入 tmux 命令也能看到警告信息，当然也可以定制它的配色。配置方法几乎和配置状态栏相同。

我们把背景色改为黑色，文字颜色设置为白色。当有消息提示时设置它的颜色为亮白色。配置如下：

```
set -g message-fg white
set -g message-bg black
set -g message-attr bright
```

就是这么简单。现在我们来为窗口列表两边的状态栏区域配色。

2.4 定制状态栏

tmux 的状态栏能显示非常多的信息。可以通过执行 shell 命令使用预定义的组件（components）或者创建我们自己的组件。

默认的状态栏显示的信息，看起来像是这样：

```
[development] 0:bash* "example.local" 00:44 02-Nov-1
```

在左侧，先看到 tmux 会话的名称，然后后面跟着窗口列表。窗口列表先是显示当前窗口的索引值，然后跟着窗口的名称。在右侧，可以看到服务器的主机名（或者是本机的主机名，译者注），后面跟着日期和时间。下面我们来自定义状态栏的内容。

配置状态栏条目

状态栏包含 3 个组件：一个左面板，窗口列表和一个右面板。我们可以改变状态栏里左侧或右侧面板的内容，这需要使用一个文本和变量的组合。表1（状态栏变量）列出了状态栏里可能用到的变量。

表1 - 状态栏变量

变量	描述
#H	本地主机的主机名
#h	本地主机的主机名，没有 domain
#F	当前窗口的标签
#I	当前窗口的索引
#P	当前面板的索引
#S	当前会话的名称
#T	当前窗口的标题
#W	当前窗口的名称
##	一个 # 符号
\$(shell-command)	shell 命令的第一行输出
#[attributes]	要改变的颜色或属性

例如，如果想要在左侧显示当前 tmux 会话的名称，就需要使用 `set-option -g status-left` 选项，后面跟着 `#S` 值，就像这样：

```
set -g status-left "#S"
```

还可以通过设置前景色让它显示地更明显，像这样：

```
set -g status-left "#[fg=green]#S"
```

可以向状态栏里添加任何想要的属性和条目。为了便于展示，我们修改了左侧的状态栏，让它显示绿色的会话名称，黄色的窗口编号，以及蓝绿色的当前面板。配置如下：

```
set -g status-left "#[fg=green]#S #[fg=yellow]#I #[fg=cyan]#P"
```

也可以向状态栏里添加任意文字。我们现在添加一些文字，让会话、窗口和面板显示地更突出，像这样：

```
set -g status-left-length 40
set -g status-left "#[fg=green]Session: #S #[fg=yellow]#I #[fg=cyan]#P"
```

我们设置了 `status-left-length` 选项因为指定的输出对默认长度来说太长了，所以我们让那个区域更宽一些。

还可以配置右侧的状态栏。现在我们向它添加当前日期和时间：

```
set -g status-right "[fg=cyan]%d %b %R"
```

这样配置的日期格式是“13-Jan 13:45”，你可以让它显示任意你想要的格式，可以使用许多编程语言通用的 `strftime()` 时间格式化机制。

在状态栏里开启 UTF-8 支持是个不错的注意，尤其是如果你特别喜欢使用这些字符。

```
set -g status-utf8 on
```

还可以更进一步，通过使用 `$(shell-command)` 变量把 shell 命令加入到状态栏中，在状态栏显示该命令的返回结果。在后续章节会详细介绍这个功能。

让状态栏实时更新信息

我们已经把当前时间和一些其它动态信息添加到了状态栏，这时需要告诉 tmux 这些信息的刷新周期。默认配置下，tmux 会每 15 秒刷新一次状态栏。可以通过使用 `set-option -g status-interval` 命令后面加上刷新周期（以秒为单位，译者注）来指定 tmux 的刷新时间，就像这样：

```
set -g status-interval 60
```

这样就会让 tmux 每 60 秒刷新一次状态栏。注意，如果你在状态栏里添加了 shell 命令，这些命令也会在每次状态栏刷新时执行一遍，所以要注意不要加载太多资源密集型的脚本。

让窗口列表居中显示

我们还能控制窗口列表显示的位置。默认的，窗口列表是靠左对齐的，通过简单的配置就可以让窗口列表在左右面板之间居中显示：

```
set -g status-justify centre
```

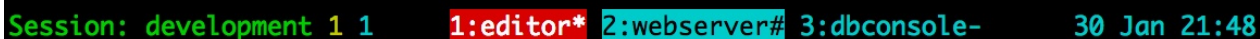
这样配置就会让窗口列表居中显示。创建新窗口时，窗口列表会相应地变换位置，让整个窗口列表显示在状态栏正中间。

窗口活动通知

同样的，我们希望如果当前会话的其他窗口里有一些事件发生时我们能够注意到这些事件，那么我们就可以快速响应那个窗口。可以通过增加一个可视化的通知（visual notification）实现这个功能，像这样：

```
setw -g monitor-activity on
set -g visual-activity on
```

现在呢，如果其它窗口里有一些活动，它就会使用蓝绿色的背景色突出显示，就像这里的 webserver 窗口：

A screenshot of a tmux session header. It shows the session name 'development' followed by window indices '1 1'. The active window is '1:editor*' with a red background. Other windows are '2:webserver#' with a blue-green background and '3:dbconsole-' with a black background. The date and time '30 Jan 21:48' are on the right.

2.5 接下来做什么？

在本章，我们已经构建了一个相当实用的配置文件。你可以在附录 1（本书使用的配置文件）里查看完整的 `~/.tmux.conf` 文件。

在 `~/.tmux.conf` 文件里可以定义许多附加选项。例如，在第 3 章我们会讨论如何使用特定项目的配置文件（project-specific configuration files）设置一个自定义的工作环境。

另外，你可以为你的系统定制一个默认配置，它位于 `/etc/tmux.conf`。如果你配置了一个共享服务器让你的团队成员可以合作，或者只是想让系统的每个用户都有一些预设配置时，这时定制一个系统配置文件最好不过了。

现在我们已经有了一个定义好的配置，让我们回顾一下是如何通过脚本创建我们自己的开发环境的，而且设置好之后一劳永逸，无需每次重新配置。

以备查阅

命令	描述
<code>set -g prefix C-a</code>	设置前缀键为 CTRL-a
<code>set -sq escape-time n</code>	设置 tmux 等待前缀键和命令键之间的时间间隔（毫秒）
<code>source-file [file]</code>	加载一个配置文件。重新加载当前配置文件或以后加入附加配置选项。
<code>bind C-a send-prefix</code>	两次按下 <code>PREFIX</code> 键后向 tmux 发送组合键
<code>bind-key [key] [command]</code>	新建一个快捷键，执行指定的 <code>command</code> 。可简写为 <code>bind</code>
<code>bind-key -r [key] [command]</code>	新建一个可重复的快捷键，就是说只需按下一次 <code>PREFIX</code> 键之后就可以重复地按下命令键。当你想要在元素之间循环移动或调整面板大小时非常有用。可简写为 <code>bind</code>
<code>unbind-key [key]</code>	移除一个定义的快捷键，让它绑定到其它命令。可简写为 <code>unbind</code>
<code>display-message</code> 或 <code>display</code>	在状态消息里显示给定的文字
<code>set-option [flags] [option] [value]</code>	配置会话选项。使用 <code>-g</code> 选项可作为全局配置
<code>set-window-option [option] [value]</code>	配置窗口选项，例如活动通知，光标移动，或其它与窗口和面板相关的元素
<code>set -a</code>	把值添加到当前选项而不是替换选项的值

第 3 章 脚本定制 tmux 环境

在项目上工作时，你可能需要运行一大堆的工具和程序集。如果你在做一个 web 应用，你可能需要一个命令窗口，一个文本编辑器，一个数据库命令窗口，和一个运行着你的自动化测试套件的窗口。这样就有大一堆的窗口需要管理，一大堆的命令需要输入。

想象一下你来到了你的工作站前刚坐下，准备开始为你的项目添加新的特性，然后只需要一个简单的命令就可以把这些程序运行起来，每个程序都运行在一个的 tmux 会话中，拥有它自己的面板或是窗口。我们可以使用 tmux 的客户端-服务器模型（client-server model）来创建一个定制的脚本来自动地构建开发环境、分割窗口并运行程序。我们接下来会先学习如何手动执行，然后深入学习高级的自动化工具。

3.1 使用 tmux 命令创建一个自定义安装

我们已经学习了如何使用 tmux 命令来创建新的 tmux 会话，但是 tmux 命令还有很多其它的参数可以使用。例如选择一个运行中的会话然后把它的窗口分割成面板，改变布局，甚至是在会话内运行程序。

关键参数是 `-t`，也就是 `target`。当我们有一个已命名的 tmux 会话，可以像这样连接到它：

```
$ tmux attach -t [session_name]
```

我们使用这个 `target` 参数来引导命令指向正确的 tmux 会话。所以，如果我们创建了一个新的名为 `development` 的 tmux 会话，像这样：

```
$ tmux new-session -s development
```

然后我们使用 `PREFIX d` 从这个会话分离出来，那么可以使用这个命令让这个会话水平分割窗口：

```
$ tmux split-window -h -t development
```

执行完命令后，如果再连接到这个会话，我们就会发现窗口已经分割成了两个面板。命令：

```
$ tmux attach -t development
```

实际上，并不需要从 tmux 会话里分离出来再发送命令。可以另开一个终端然后把窗口再次分割，但是这次我们使用垂直分割，命令如下：

```
$ tmux split-window -v -t development
```

通过这种方法可以非常简便地自定义当前已存在的工作环境。下面来看看其它的一些命令。

编写一个项目配置脚本

在第一章，我们讨论了譬如 `new-session` 和 `new-window` 的 `tmux` 命令。下面我们来编写一个简单的脚本，创建一个新的 `tmux` 会话，它含有多个窗口，有一个窗口会包含多个面板。最重要的是，我们让每个面板都运行不同的程序。

我们从创建一个新的脚本文件开始，在主目录下创建一个名为 `development` 的脚本。然后赋予这个文件可执行权限，这样就可以运行它了。命令如下：

```
$ touch ~/development
$ chmod +x ~/development
```

在这个新的脚本文件里，首先创建一个名为 `development` 的 `tmux` 会话，内容如下：

```
tmux new-session -s development -n editor -d
```

在创建这个会话的同时我们给它传入了一些附加的参数。首先，创建这个会话并为它命名，使用了 `-s` 参数。然后把初始窗口命名为 `editor`，然后再通过 `-d` 参数立即从这个新的会话中分离。

开始使用会话时，我们希望把工作目录变更到项目目录。我们把项目目录称作 `devproject`。当然，在变更到这个目录之前，最好先创建它，命令如下：

```
$ mkdir ~/devproject
```

目录创建之后，我们把下面一行内容添加到配置文件中，这里使用了 `send-keys` 命令：

```
tmux send-keys -t development 'cd ~/devproject' C-m
```

我们在这行配置的最后添加了一个 `C-m`（Control-M），这样就向 `tmux` 里发送了一个回车符。接下来我们就可以重复使用这条命令，在窗口里打开 `Vim` 编辑器。就像这样：

```
tmux send-keys development 'vim' C-m
```


通过这 3 个命令，我们就已经创建了一个新的会话，变更了工作目录，并打开了一个编辑器。但是我们的工作环境还没有配置完成。我们来分割一下主编辑窗口，这样就能在底部拥有一个小的终端窗口。我们使用 `split-window` 命令来完成这项任务。在我们的脚本里，添加这样一行：

```
tmux split-window -v -t development
```

这样就会让主窗口水平地分割。还可以在分割时指定窗口所占的百分比，像这样：

```
tmux split-window -v -p 10 -t development
```

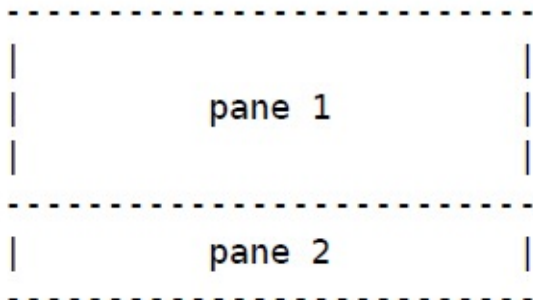
但是我们并不会这样做，我们会把 `split-window` 命令放到一边，然后选择一个 tmux 默认布局——`main-horizontal`——只需要把下面的内容添加到配置中：

```
tmux select-layout -t development main-horizontal
```

我们已经创建了第一个窗口并把它分割为两个面板，但是我们需要让底部的面板需要在打开时就处于项目目录的位置。我们已经知道如何向 tmux 实例发送命令，接下来我们会学习如何向指定的面板和窗口发送命令。

指定目标面板和窗口

通过例如 `send-keys` 这样的命令，不仅可以指定目标会话，也可以指定窗口和面板。在第 2 章的配置文件里，我们指定了初始索引为 1，也就是说窗口从 1 开始编号（而不是从 0 开始）。但是这个初始索引并不会影响到面板的索引编号，这也就是我们为什么也要把面板的初始索引设置为 1 的原因。在下面的例子里当前窗口有两个面板，如图 9（两个面板）所示：



在 Pane 1 里打开了 Vim 编辑器，然后我们想在 Pane 2 里发送一个命令来改变项目路径。可以通过这种格式来指定一个具体的面板 `[session]:[window].[pane]`，这个例子里就是 `development:1.2`，把下面这行配置添加到配置脚本里，就会得到我们想要的效果：

```
tmux send-keys -t development:1.2 'cd ~/devproject' C-m
```

几乎就要完成了。接下来就可以用我们学到的内容来完成配置，添加一些窗口到会话中。

创建并选择窗口

我们想要在会话里有第二个窗口并运行全屏大小的控制台。可以通过 `new-window` 命令来创建新窗口：

```
tmux new-window -n console -t development
tmux send-keys -t development:2 `cd ~/development`C-m
```

创建窗口之后，使用 `send-keys` 命令再次把当前路径改到项目路径。在新窗口里只有一个面板，所以只需指定窗口的编号即可。

启动了新会话后，我们想让第一个窗口显示出来，可以使用 `select-window` 命令：

```
tmux select-window -t development:1
tmux attach -t development
```

可以继续向这个脚本里添加命令，创建更多的窗口和面板、开始到远程服务器的连接、追踪日志文件、连接到数据库控制台、或者在开始工作时运行命令获取最新的代码。但是我们就讲到这里，最后以连接到会话来简单地结束我们的脚本，让我们配置的 `tmux` 会话显示在屏幕上，准备开始工作。全部脚本如下所示：

```
tmux new-session -s development -n editor -d
tmux send-keys -t development 'cd ~/devproject' C-m
tmux send-keys -t development 'vim' C-m
tmux split-window -v -t development
tmux select-layout -t development main-horizontal
tmux send-keys -t development:1.2 'cd ~/devproject' C-m
tmux new-window -n console -t development
tmux send-keys -t development:2 'cd ~/devproject' C-m
tmux select-window -t development:1
tmux attach -t development
```

运行下面这个命令时，我们就会看到如图 10（脚本定制的开发环境）所示的效果：

```
$ ~/development
```

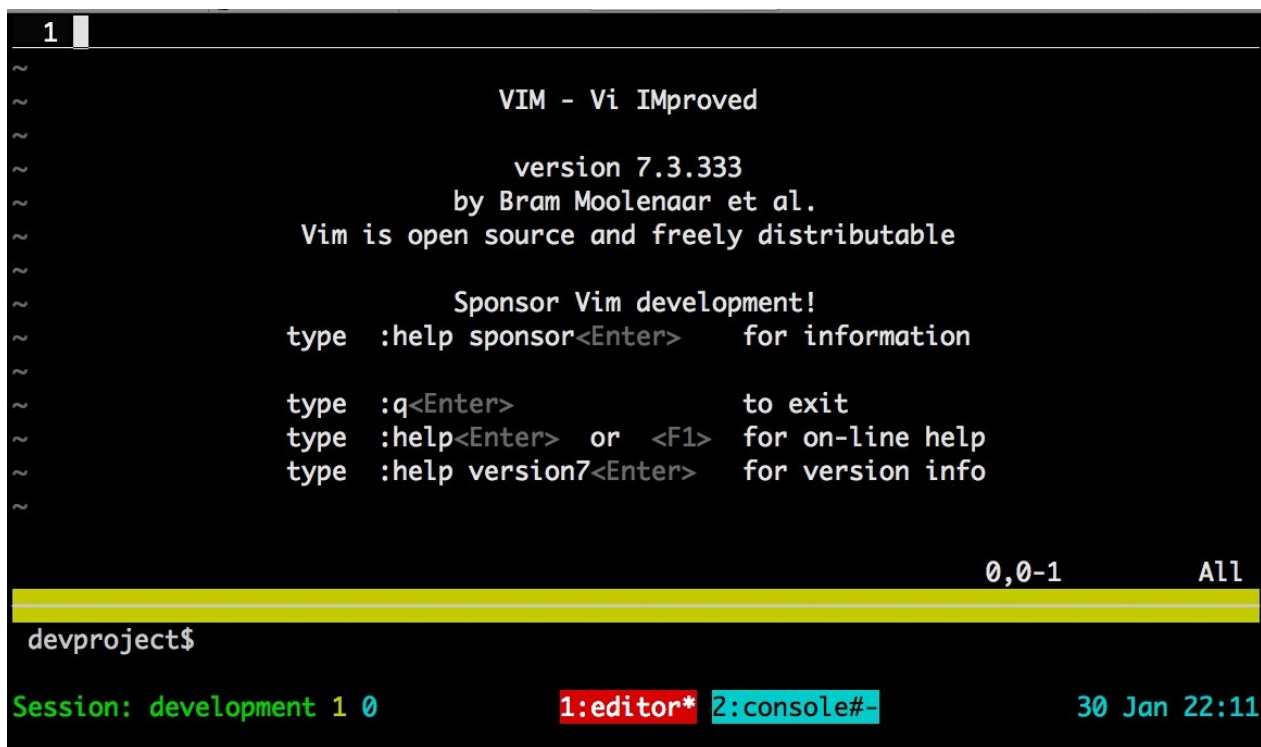


图10 - 脚本定制的开发环境

这种方法的缺点就是这个脚本创建了一个全新的会话。如果你之前已经创建过一个 `development` 会话，然后再次执行这个脚本的话它就无法正常工作。可以修改脚本，检查是否有同名的会话存在来解决这个问题，使用 `tmux has-session` 命令来检查，然后之后只有在这个会话不存在时才创建这个会话，就像这样：

```
tmux has-session -t development
if [ $? != 0 ]
then
    tmux new-session -s development -n editor -d
    tmux send-keys -t development 'cd ~/devproject' C-m
    tmux send-keys -t development 'vim' C-m
    tmux split-window -v -t development
    tmux select-layout -t development main-horizontal
    tmux send-keys -t development:1.2 'cd ~/devproject' C-m
    tmux new-window -n console -t development
    tmux send-keys -t development:2 'cd ~/devproject' C-m
    tmux select-window -t development:1
fi
tmux attach -t development
```

这样做就能使一个项目配置正常工作了。你可以修改这个脚本，让项目名使用一个变量，这样脚本就能变得通用，但是现在我们来看看另外一些管理多个项目的配置方法。

3.2 设置使用 tmux 配置

`.tmux.conf` 文件本身可以包含命令来设置一个默认的环境。如果想让 `tmux` 会话每次都在相同的默认文件夹路径启动，或者是想让它自动地打开一个分割的窗口，可以把这些写进默认配置文件里，只需使用相似的命令。

还可以指定一个配置文件，让 `tmux` 实例在启动时加载它，只需使用 `-f` 参数。这种方式就不需要改变默认的配置文件，并且可以在项目中检查配置文件。还可以设置每个项目对应一个配置选项，比如新的键盘快捷键。

让我们试试这个吧，现在创建一个名为 `app.conf` 的配置文件。在这个文件里，可以使用在之前章节使用的相同命令，由于是在配置文件里而不是一个 `shell` 脚本里，我们就无需特别指定 `tmux` 的每个命令都使用 `tmux` 前缀，配置文件如下：

```
source-file ~/.tmux.conf
new-session -s development -n editor -d
send-keys -t development 'cd ~/devproject' C-m
send-keys -t development 'vim' C-m
split-window -v -t development
select-layout -t development main-horizontal
send-keys -t development:1.1 'cd ~/devproject' C-m
new-window -n console -t development
send-keys -t development:2 'cd ~/devproject' C-m
select-window -t development:1
```

请注意，在配置文件的第一行就导入了原始 `.tmux.conf` 文件。这样就能使用之前定义的所有环境设置，包括快捷键和状态栏设置。这个导入配置文件的命令并不是强制的，但是如果不导入这个文件，就必须使用所有的默认快捷键和默认选项，或者只能在这个文件里重新定制选项。

要使用这个配置文件，必须要传入 `-f` 参数，后面加上配置文件的路径。还必须使用 `attach` 命令来启动 `tmux`，就象这样：

```
$ tmux -f app.conf attach
```

这是因为默认 `tmux` 在启动时总是调用 `new-session` 命令。我们的配置文件已经创建了一个新的会话，所以如果不使用 `attach` 参数的话就会拥有 2 个 `tmux` 会话。

这种方法有很大的灵活性，可以通过使用一个名为 `tmuxinator` 的命令行工具做更多的事情。

3.3 使用 `tmuxinator` 管理配置

`tmuxinator` 是一个简单的工具，可以用它编写并管理不同的 `tmux` 配置。我们使用简单的 `YAML` 格式来定义窗口布局和命令，然后就可以使用 `tmuxinator` 命令登录了。和其他方法不同，`tmuxinator` 为配置文件提供了一个集中的位置和一个更简单的办法来创建复杂的布局。

它还允许在每个窗口创建之前运行特定的命令。

tmuxinator 的运行需要 Ruby 解释器，所以你的系统上需要安装了 Ruby 解释器才能使用 tmuxinator。MAC OS X 的用户已经安装了 Ruby，Linux 用户可以通过包管理器安装 Ruby 解释器。如果你打算不仅仅为了使用 tmuxinator 而安装 Ruby 解释器的话，强烈建议通过 RVM 安装 Ruby，你可以依照 RVM 网站的指引来安装 Ruby。[RVM 网站链接](#)

我们使用 Rubygems 安装 tmuxinator，Rubygems 是 Ruby 的包管理系统。命令如下：

```
$ gem install tmuxinator
```

如果你没有使用 RVM，你需要 root 权限来执行这条命令或者使用 sudo 命令。

tmuxinator 需要预先定义 shell 环境变量 \$EDITOR，所以如果你还没设置的话，可以在 Linux 系统上修改你的 .bashrc 文件，或者在 OS X 系统上修改 .bash_profile 文件。例如，如果你想把 Vim 作为默认的编辑器，你需要在 Bash 配置文件里增加这条语句：

```
export EDITOR=vim
```

现在就可以创建一个新的 tmuxinator 项目了，我们把它称作 development。命令如下：

```
$ tmuxinator open development
```

这会打开我们预定义的 \$EDITOR 变量的编辑器并显示默认的项目配置，就像这样：

```
project_name: Tmuxinator
project_root: ~/code/rails_project
socket_name: foo # Not needed. Remove to use default socket
rvm: 1.9.2@rails_project
pre: sudo /etc/rc.d/mysqld start
windows:
- editor:
  layout: main-vertical
  panes:
    - vim
    - #empty, will just run plain bash
    - top
- shell: git pull
- database: rails db
- server: rails s
- logs: tail -f logs/development.log
- console: rails c
- capistrano:
- server: ssh me@myhost
```

这是一个 Ruby on Rails 开发者使用 Git 工作的一个非常棒的工作环境。它创建了一个含有 8 个窗口的 tmux 会话。第 1 个窗口被分为 3 个面板，使用了 main-vertical 界面布局。剩下的窗口打开了一些服务器连接和控制台，如图 11（使用 tmuxinator 默认配置的 Rails 工作环境）所示：

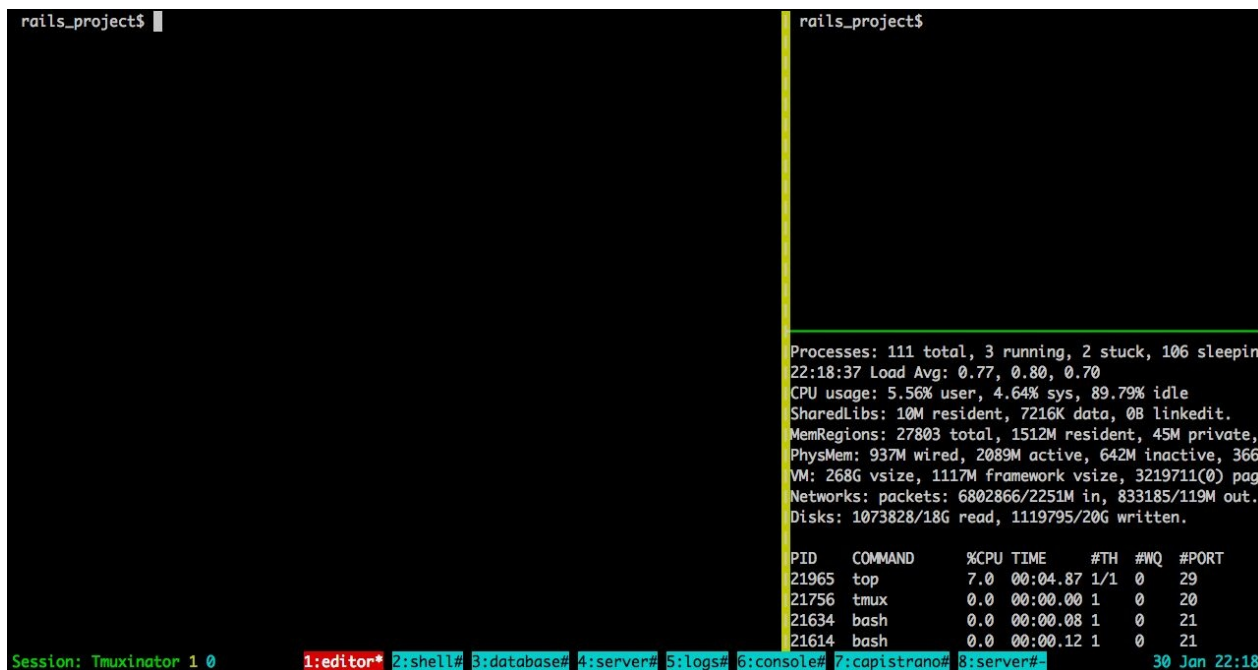


图 11 - 使用 tmuxinator 默认配置的 Rails 工作环境

如你所见，tmuxinator 让定义窗口和面板变得非常简单，而且也很容易让它们执行独立的命令。可以在每个窗口加载时指定它要执行的命令。让我们移除这些配置然后构建一个开发环境，在窗口上部打开 Vim，下面显示终端，初始化时进入 `~/devproject` 路径中。配置如下：

```
project_name: devproject
project_root: ~/devproject
windows:
- editor:
  layout: main-horizontal
  panes:
    - vim
    - #empty, will just run plain bash
- console: # empty
```

`yaml` 文件格式使用的是 2 个空格作为缩进，所以你一定要确保在写入文件时保证文件里的缩进完全正确。

要使用新环境，我们保存刚才的配置文件然后执行下面的命令：

```
$ tmuxinator devproject
```

tmuxinator 就会自动地加载原始 `.tmux.conf` 文件，应用设置，然后使用刚才配置文件指定的窗口和面板设置。如果想再对环境有所修改，只需要再次执行这个命令：

```
$ tmuxinator open devproject
```

默认情况下，tmuxinator 的配置文件位于 `~/.tmuxinator/` 文件夹里，所以你可以在这里找到配置文件、备份配置文件，或是将配置文件与他人一起分享。

其实，tmuxinator 只是一个构造了一个脚本来执行独立的 tmux 命令，就像在之前在脚本文件里写的那样。它提供了一个非常漂亮又好记的语法。但是，要想使用 tmuxinator 你需要先安装一个 Ruby 解释器，所以它可能无法实现在任意环境下都使用 tmuxinator 来管理 tmux 的这种需求。

3.4 接下来做什么？

你可以在 shell 里使用 tmux 的每个命令，这就意味着你可以编写脚本来自自动化 tmux 的几乎所有功能，包括运行会话等。例如，创建一个快捷键来运行一个 shell 脚本文件，这个脚本文件把当前的窗口分割为两个面板并把你的行为写入到 Web 和数据库的生产服务器的日志中。我们会在第 6 章讨论这些技术细节。

我们已经探讨了编写 tmux 脚本的 3 种不同方式，而且已经数次优化了配置。我们现在知道如何设置项目、在面板和窗口之间来回切换以及登录控制台。但是当我们在 tmux 会话里使用应用程序时，程序日志或者是测试结果会开始不停的刷屏。所以在接下来我们会学习如何处理输出缓冲区以及如何复制、粘贴文本。

以备查阅

可脚本化的 **tmux** 命令

命令	描述
<code>tmux new-session -s development -n editor</code>	创建一个名为 development 的会话，将第一个窗口命名为 editor
<code>tmux attach -t development</code>	连接到一个名为 development 的会话
<code>tmux send-keys -t development '[keys]' C-m</code>	向 development 会话的活动窗口或面板发送键盘命令， C-m 相当于按下回车键
<code>tmux send-keys -t development:1.0 '[keys]' C-m</code>	向 development 会话的第 1 个窗口和第 1 个面板发送键盘命令， C-m 相当于按下回车键
<code>tmux select-window -t development:1</code>	选择 development 会话的第 1 个窗口，让它作为活动窗口
<code>tmux split-window -v -p 10 -t development</code>	在 development 会话里垂直地分割当前窗口，把它分为水平的 2 个面板并设置它的高度为总窗口大小的 10%
<code>tmux select-layout -t development main-horizontal</code>	设置 development 会话的布局为 main-horizontal
<code>tmux -f app.conf attach</code>	加载配置文件 app.conf 并连接到由这个配置文件所创建的一个会话
<code>tmuxinator open [name]</code>	在项目名 [name] 路径下使用默认的编辑器打开配置文件。如果不存在就创建一个新的配置文件
<code>tmuxinator [name]</code>	对指定项目加载 tmux 会话。如果会话不存在或无法连接到会话就根据项目的配置文件创建一个新会话
<code>tmuxinator list</code>	列出当前的项目
<code>tmuxinator copy [source] [destination]</code>	复制一个项目配置文件
<code>tmuxinator delete [name]</code>	删除指定的项目
<code>tmuxinator implode</code>	删除当前所有的项目
<code>tmuxinator doctor</code>	使用 tmuxinator 和系统配置查找问题

第 4 章 文本和缓冲区

在日常工作过程中，你使用复制和粘贴的次数远超你的想象。使用 tmux，可以直达你需要终端输出缓冲区屏幕回滚的地方来查看缓冲区里那些已经不在屏幕范围的内容。你还可能需要复制一段文字然后把它复制到一个文件或另一个程序里。本章将会介绍如何管理 tmux 会话里的文本。你会学习到如何使用键盘操作 tmux 的输出缓冲区，如何使用多重粘贴缓冲区，以及如何使用系统剪贴板。

4.1 使用复制模式滚动输出

在终端里使用程序时，程序的输出常由于内容过多而不停滚动以致超出了屏幕范围。使用 tmux，可以通过键盘来向前翻动输出缓冲区，这样就可以看到错过的内容。这个功能在运行测试或是审阅日志文件时尤其有用。

按下 `PREFIX` `[` 键会进入复制模式，然后就可以使用光标移动键在屏幕里移动光标。默认地，光标移动键是箭头键。在第 2 章，我们在配置文件里使用了 Vim 模式的移动键，这样就可以在窗口之间移动以及重绘面板大小时不用把手移出键盘的主操作区。tmux 里有一个使用 vi 模式操作缓冲区的功能。要开启这个功能，请把下面这行代码添加到你的 `.tmux.conf` 文件里：

```
setw -g mode-keys vi
```

这个配置可以使用 `h`，`j`，`k` 和 `l` 键在缓冲区里移动。要离开复制模式，只需按下 `ENTER` 键（回车键，译者注）。但是一次移动一个字符显然是不怎么高效的。那么既然我们开启了 vi 模式，我们也可以使用其它 vi 的快捷键在缓冲区里移动。

例如，可以使用 `w` 键跳到下一个单词，使用 `b` 键回跳一个单词。也可以使用 `f` 键，后面跟随任意字符来跳到当前行的指定字符，使用 `F` 键回跳。

在缓冲区里快速移动

当缓冲区里输出了多个页面时，在滚屏之间使用光标移动并不是很高效。取代单词之间移动或是字符之间移动的一种方式，就是在缓冲区里一页一页的滚动，或者是跳转到缓冲区的顶部或底部。

可以使用 `Ctrl b` 向上翻滚一屏或是使用 `Ctrl f` 向下翻滚一屏，使用 `g` 键跳转到缓冲区历史的最顶部，也可以使用 `G` 键跳转到缓冲区历史的最底部。

查找缓冲区

如果知道要查找什么内容的话我们就不必一页页的在数百行内容之间浏览。在复制模式里按下 `?` 键，可以向上查找短语或关键词。只需按下 `?` 键，输入要查询的短句，然后按下 `ENTER` 键就会跳转到第一个匹配的短句。然后按下 `n` 键跳转到下一个匹配，或者按 `N` 移动到上一个匹配。

要向下查找，你需要按下 `/` 键而不是 `?` 键。按下 `n` 键会跳转到下一个匹配而按下 `N` 键会跳转到上一个匹配。

学习以这种方式在缓冲区之间移动会极大地加快你的速度。输入想要移动到的单词而不是使用箭头来移动会更快，尤其是在查找日志文件的输出时会更明显。

这只是一些使用缓冲区的基本知识。下面我们来学习如何复制一个面板的文本然后把它粘贴到另一个面板中。毕竟，这是 `tmux` 的复制模式。

4.2 复制和粘贴文本

在输出缓存区里移动和查找内容只是程序的一半（意为部分功能，译者注）。我们还会经常需要复制一些文本。`tmux` 的复制模式提供了这个机会，它能让我们选择并复制文本到一个粘贴缓存区，这样就可以把它复制到任何地方。

要复制文本，需要先进入复制模式然后把光标移动到要选择的文本的开始处。然后按下 `SPACE` 键（空格键，译者注）然后移动光标到文本的尾部。当我们按下 `ENTER` 键后，这段被选择的文本就会被复制到一个粘贴缓存区中。

要粘贴刚才捕捉的内容，则切换到粘贴模式然后按下 `PREFIX]` 键。

下面我们来学习一些特定的方式来从主输出缓冲区里复制和粘贴文本。

捕捉面板

`tmux` 有一个非常方便的快捷键可以把整个面板的可视内容全部复制到一个粘贴缓存区里。只需要按下 `PREFIX :` 键进入命令模式然后输入

```
capture-pane
```

接下来就可以使用 `PREFIX]` 键把内容复制到当前焦点会话里。

显示并保存缓存区

显示粘贴缓存区的内容，只需要在命令模式里使用 `show-buffer` 命令，或者是在终端会话里使用如下命令：

```
$ tmux show-buffer
```

使用 `save-buffer` 命令可以把缓存区保存到一个文件里，而且这是实时的保存。事实上，可以捕捉当前缓存区并保存到一个文本文件，就像这样：

```
$ tmux capture-pane && tmux save-buffer buffer.txt
```

或者在命令模式里使用 `capture-pane; save-buffer buffer.txt` 命令。当然，如果你想的话，可以把这个操作定义为一个快捷键。

使用多重粘贴缓存区

tmux 保留了一个粘贴缓存区的栈空间，也就是说可以多次复制文本而无需替换缓冲区里已有的内容。这比操作系统提供的传统剪贴板要便捷得多。

每次我们复制一些新的文本时，tmux 就会创建一个新的粘贴缓存区，把新的缓存放在栈的最顶端。

为了更清楚的描述，下面我们来创建一个新的 tmux 会话然后打开一个文本编辑器例如 Vim 或 Nano。在编辑器里，输入以下句子，每句话占用一行空间：

```
First sentence is first.  
Next sentence is next.  
Last sentence is last.
```

现在复制一些文本到粘贴缓存区里。按下 `PREFIX [` 键进入复制模式。移动到第一个句子的开始处，按下 `SPACE` 键开始选择文本，移动到第一句话的末端，然后按下 `ENTER` 键选择整句话。然后对第 2 句话和第 3 句话重复刚才的操作。

每次我们复制文本时，tmux 都创建一个新的缓存区。可以通过 `list-buffers` 命令查看这些缓存区。如下：

```
0: 22 bytes: "Last sentence is last."  
1: 22 bytes: "Next sentence is next."  
2: 24 bytes: "First sentence is first."
```

按下 `PREFIX]` 键会总是粘贴第 0 个缓存内容，输入 `choose-buffer` 命令来选择一个特定的缓存区然后把它的内容粘贴到焦点面板中。

把当前窗口分为两半，然后在第 2 个面板中打开 Nano 程序，然后进入命令模式输入以下命令：

```
choose-buffer
```

你会看到一个列表显示如图 12（选择一个文本缓存并插入）所示。你可以选择列表中的任意一个，按下 `ENTER`，文本就会被插入到指定的面板。

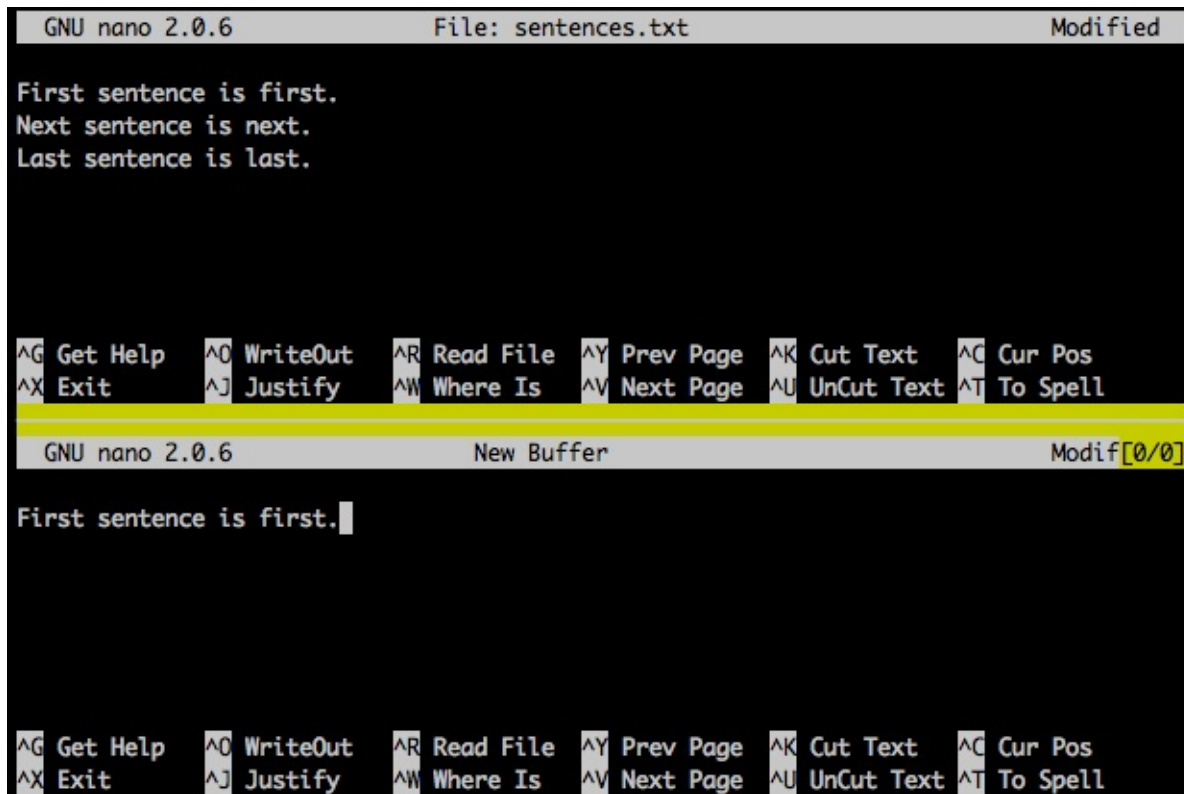


图 12 - 选择一个文本缓存并插入

这种方式来管理多重文本非常棒，当你使用一个字符界面的环境却无法进入剪贴板时尤其有用。

这些缓存会在 tmux 会话的运行过程中被共享，因此我们可以把粘贴缓存中的内容从一个会话复制到另外一个里。

自定义复制和粘贴快捷键

如果你习惯使用 Vim 而且用复制和粘贴命令比较频繁，你可以在你的配置文件里重新定义这两个命令的快捷键。例如，可以把 `ESCAPE` 键（`ESC` 键，译者注）定义为进入复制模式，使用 `y` 键（yank，复制，译者注）把文本复制到缓存区，使用 `v` 键进入视图模式（Visual mode，译者注）选择文本，然后使用 `p` 键（paste，译者注）粘贴文本，配置如下：

```
unbind [  
bind Escape copy-mode  
unbind p  
bind p paste-buffer  
bind -t vi-copy 'v' begin-selection  
bind -t vi-copy 'y' copy-selection
```

如果你经常在窗口和面板之间大量使用复制和粘贴操作的话，这样的配置会极大程度地提高你的工作效率，而且这些快捷键都是你非常熟悉的 Vim 快捷键。

4.3 在 Linux 使用剪贴板

使用 `xclip` 实用工具可以集成 Linux 系统上的剪贴板，这样你就可以在不同程序之间更加便捷地进行复制和粘贴操作。

首先你需要安装 `xclip` 工具。在 Ubuntu 系统上，使用如下命令：

```
$ sudo apt-get install xclip
```

然后通过 `xclip` 来使用 tmux 的 `save-buffer` 和 `set-buffer` 命令。

要把当前缓存区的内容复制到系统剪贴板，我们在 `.tmux.conf` 文件里添加如下命令：

```
bind C-c run " tmux save-buffer - | xclip -i -sel clipboard"
```

这个配置定义了 `PREFIX CTRL-C` 快捷键来捕获当前缓存区的内容然后通过管道输出到 `xclip` 程序里。可以把下面这个命令也添加到配置文件里，这样就可以使用 `PREFIX CTRL-V` 快捷键把系统剪贴板的内容复制到 tmux 会话中来：

```
bind C-v run " tmux set-buffer \"$(xclip -o -sel clipboard)\"; tmux paste-buffer"
```

这个命令会把 `xclip` 的内容输出到一个新的 tmux 缓存区里然后把它粘贴到当前的 tmux 窗口或面板。

4.4 使用 OS X 剪贴板命令

如果你是一个 Mac 用户，你会比较熟悉 OS X 的命令行剪贴板工具 `pbcopy` 和 `pbpaste`。这些简单的工具使得你在使用剪贴板时更加方便。`pbcopy` 命令把文本复制到系统剪贴板，`pbpaste` 命令把内容粘贴出来。例如，你可以一起使用 `pbcopy` 和 `cat` 命令，这样就

可以很方便地把 `.tmux.conf` 文件的配置复制到剪贴板，然后可以把它粘贴到电子邮件或是 Web 上，就像这样：

```
$ cat ~/.tmux.conf | pbcopy
```

这样操作文本确实很方便，但是 tmux 并没有权限访问这些工具，在 tmux 会话里并不能直接使用它们。可以使用一个由 Chris Johnsen 编写的包装程序（wrapper program）来突破这个限制。[Github 链接](#)

要使用这个包装脚本，首先克隆这个项目到本地然后编译它。代码如下：

```
$ git clone https://github.com/ChrisJohnsen/tmux-MacOSX-pasteboard.git
$ cd tmux-MacOSX-pasteboard/
$ make reattach-to-user-namespace
```

然后，把文件移动到 `PATH` 路径下的某个位置，例如 `/usr/local/bin`，代码如下：

```
$ sudo mv reattach-to-user-namespace /usr/local/bin
```

最后，配置 tmux 来使用这个包装程序，把下面这行命令添加到 `.tmux.conf` 文件中：

```
set -g default-command "reattach-to-user-namespace -l /bin/bash"
```

这个配置会让 tmux 的新窗口能够通过包装脚本加载 Bash shell。如果你使用的是其它 shell，你需要在配置里修改它的指定路径或命令。

重新加载配置文件之后，就可以使用 `pbcopy` 命令了。这样还会有个额外的好处，就是可以把当前 tmux 缓存区的内容添加到系统剪贴板里了，命令如下：

```
$ tmux show-buffer | pbcopy
```

或者可以粘贴剪贴板里的内容，像这样：

```
$ tmux set-buffer pbpaste; tmux paste-buffer
```

这就意味着我们可以创建键盘快捷键来做这个操作，就像我们在 4.3 章节里所做的那样。但是很不幸，我们使用的包装程序无法和 tmux 的 `run` 命令一起使用。好在有解决方案，可以在 `pbpaste` 和 `pbcopy` 命令前添加包装程序的前缀。所以，为了支持复制功能，我们在配置文件里要这样定义一个快捷键：

```
bind C-c run "tmux save-buffer - | reattach-to-user-namespace pbcopy"
```

同样，要支持从系统剪贴板粘贴内容，需要把下面这个超级长的命令添加到配置文件里，要注意，这个命令必须都在同一行里：

```
bind C-v run "tmux set-buffer $(reattach-to-user-namespace pbpaste); tmux paste-buffer"
```

这样就用一个简单的办法解决了一个比较复杂的技术问题。

4.5 接下来做什么？

要想在缓存区和剪贴板之间来移动文本，你可以在一个没有剪贴板的环境中创建一个，比如当你登录了服务器的控制台或是一个没有图形界面的终端。能够在一个很长的控制台的输出历史之间来回滚动会帮我们做很多事情。仅仅这个功能，就值得你在服务器上安装 tmux。

现在我们对 tmux 里如何移动，切换窗口或面板有了比较详细的了解，可以在日常中使用 tmux 工作了。对很多开发人员来说，结对编程也是日常工作的一部分。接下来我们会学习如何使用 tmux 和其他开发人员合作。

以备参考

快捷键

快捷键	描述
PREFIX [进入复制模式
PREFIX]	粘贴当前缓存区的内容
PREFIX =	列出所有粘贴缓存区并粘贴选中的缓存内容

复制模式移动键（VI 模式）

命令	描述
h, j, k, 和 l	移动光标，分别表示向左，向下，向上和向右
w	以一个单词为单位向前移动光标
b	以一个单词为单位向后移动光标
f 后跟随任意字符	移动光标到指定字符的下一个匹配位置
F 后跟随任意字符	移动光标到指定字符的前一个匹配位置
CTRL -b	向上翻滚一个屏幕的位置
CTRL - f	向下翻滚一个屏幕的位置
g	跳转到缓存区的顶部
G	跳转到缓存区的底部
?	在缓存区内向后查找
/	在缓存区内向前查找

命令模式

命令	描述
show-buffer	显示当前缓存区内容
capture-pane	捕捉指定面板的可视内容并复制到一个新的缓存区
list-buffers	列出所有的粘贴缓存区
choose-buffer	显示粘贴缓存区并粘贴选择的缓存区内的内容
save-buffer [filename]	保存缓存区的内容到指定文件里

第 5 章 使用 tmux 结对编程

到目前为止，我们已经学习了日常使用 tmux 以及修改配置。而 tmux 最受开发人员欢迎的功能之一就是结对编程。那是我第一次使用 tmux，当我的朋友向我介绍 tmux 的众多功能时我立刻就喜欢上了这个功能。

结对编程有很多好处。和其他开发人员一起工作能够帮你发现许多你无法自己看到的事情，除非你们身处同一位置，否则结对编程就会变得相当困难。使用例如 iChat, Skype 甚至是 GoToMeeting 这种屏幕共享的软件会占用相当多的带宽，而且在网络状况不好时就会变得非常糟糕。在本章，我们会学习使用 tmux 进行结对编程，之后你就可以和另一位开发人员一起远程工作了，甚至可以使用糟糕的宾馆 WiFi 网络。

和远程用户协作有两种方式。第一种方式需要创建一个新的用户账户，你和他人一起共享这个账户。在这个账户下配置 tmux 和开发环境，然后把它作为一个共享的工作空间。第二种办法是使用 tmux 的 sockets 连接，这样你就可以让第二个用户连接到你的 tmux 会话中而无需共享你的账户信息。

这两种方法都有一种固有的安全缺陷：它们能让他人看到你的屏幕信息和你的账户。你可能在放任某人任意查看你的文件。要解决这个问题，最好使用一个中间服务器来进行结对编程。使用一个便宜的 VPS 或是一个用 VirtualBox ([链接](#)) 或 Vagrant ([链接](#)) 创建的虚拟机，你可以快速地创建一个用于结对编程的开发环境。在本章，我们会学习在远程服务器上分别使用这两种进行结对编程。

5.1 使用共享账户结对编程

使用共享账户是和他人共同协作的最简便的方式。在一个 nutshell 里，你可以开启机器上的 SSH 访问权限，将它作为主机，然后在机器上安装并配置 tmux，之后再创建一个 tmux 会话。第二个用户可以使用相同的账户登录到这台机器里然后连接到刚才创建的 tmux 会话中。通过使用 SSH 公开密钥，可以让登录过程变得透明化。下面我们来讨论这个过程。假设我们在使用一台名为 puzzles 的服务器，它使用 Ubuntu 操作系统并且已经安装了 SSH 后台程序。

首先，在服务器上创建一个名为 tmux 的用户账户。这个账户专门用于结对操作。

```
tmux@puzzles$ adduser tmux
```

使用新创建的账户前，需要配置这个账户让它添加其他开发人员的 SSH 密钥，这样他们就可以通过这些密钥来登录这个账户。可以在 tmux 账户下创建 `~/.ssh/authorized_keys` 文件来完成这个操作。首先使用 `su` 命令切换到 tmux 账户：

```
tmux@puzzles$ su tmux
```

然后要创建 `.ssh` 目录和 `.ssh/authorized_keys` 文件，并为它们配置相应的文件权限。确保只有 `tmux` 这个用户可以读、写或执行这些文件。命令如下：

```
tmux@puzzles$ mkdir ~/.ssh
tmux@puzzles$ touch ~/.ssh/authorized_keys
tmux@puzzles$ chmod 700 ~/.ssh
tmux@puzzles$ chmod 600 ~/.ssh/authorized_keys
```

接下来，就可以把公钥传输到这台服务器上了。在桌面工作机上，生成本机的公钥然后上传到服务器：

```
$ scp -p id_rsa.pub tmux@puzzles.local
```

然后在服务器上，把公钥添加到 `authorized_keys` 里。

```
tmux@puzzles$ cat id_rsa.pub >> ~/.ssh/authorized_keys
```

其他需要这个共享账户登录到服务器上的所有人都要重复上述过程。

从这里开始，我们来配置 `tmux`，文本编辑器，编译器，编程需要和版本控制系统，就像在其他开发环境要做的那样。然后在服务器上创建一个新的 `tmux` 会话。

```
tmux@puzzles$ tmux new-session -s Pairing
```

团队中的其他成员可以登录到服务器上然后通过以下命令连接到这个会话中：

```
tmux@puzzles$ tmux attach -t Pairing
```

然后就可以在同一个项目里共同协作了。当然，我们可以从会话中分离出来在后来再重新连接到这个会话，也就是说可以让开发环境一次就运行数天甚至数周。这样只要有一个支持 SSH 的终端就可以在任何时间、任何地点都能连接到这个开发环境了。

5.2 使用共享账号和组会话

当两个人连接到同一个 `tmux` 会话时，他们通常会看到相同的内容并在同一个窗口里交互。但是很多时候人们希望能够在独立的、不同的窗口工作而不用被完全控制。

使用“组会话（grouped session）”可以实现这个功能。下面我们来演示一下，首先先在远程服务器上创建一个新的会话叫做 `groupedsession`。

```
tmux@puzzles$ tmux new-session -s groupedsession
```

然后，另一个用户不去连接到这个会话，而是通过“创建一个新的会话”来加入到这个会话里，他需要指定原始会话 `groupedsession`，然后再指定一个他或她自己的会话名，就像这样：

```
tmux@puzzles$ tmux new-session -t groupedsession -s mysession
```

当第二个会话启动时，两个用户都可以同时在这个会话里进行交互，就像是第二个用户已经连接到这个会话一样。但是，每个用户可以创建相互独立的窗口。因此，如果新用户创建了一个窗口，都会在状态栏里看到新的窗口出现了，但是用户仍然会处于当前工作的窗口！这对那些“嘿，我有个想法我先去试试”的时刻特别适合，或者有时候有人想用 Emacs 而有人想用 Vim 的时候也很合适，就像图 13（两个用户共享同一个会话）所示的那样。

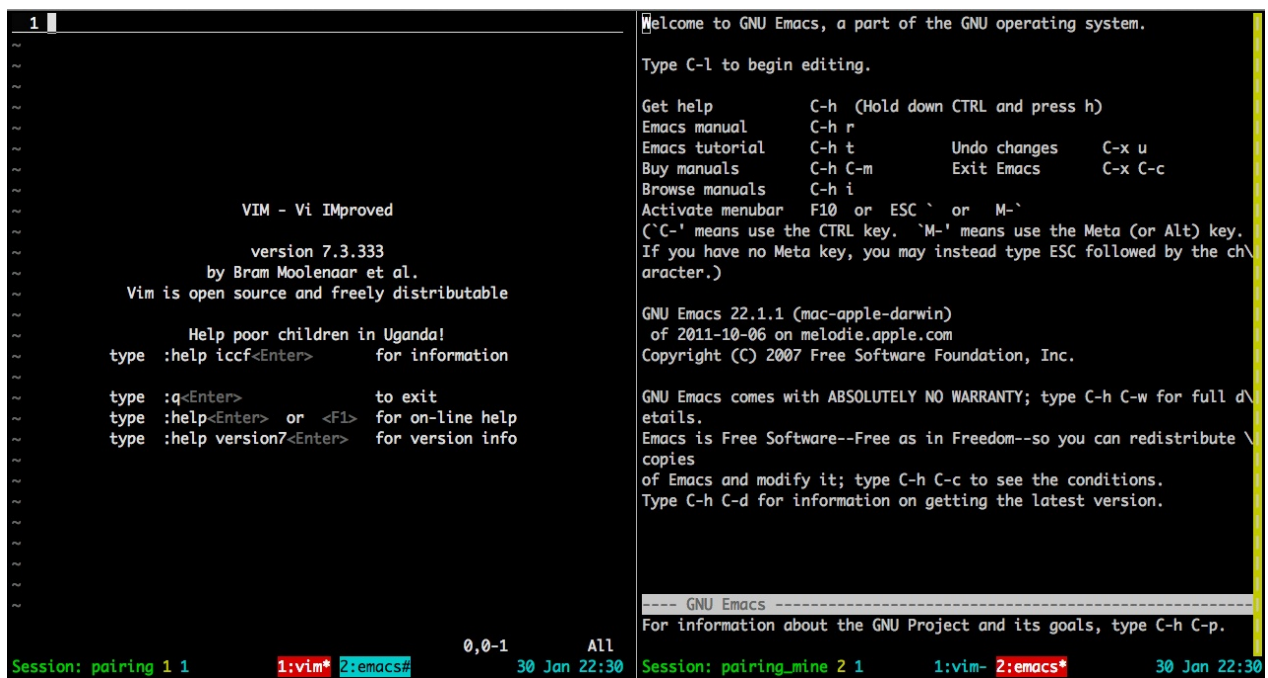


图 13 - 两个用户共享同一个会话

第二个用户可以使用 `kill-session` 命令杀死他或她的会话，而原始会话依然会存在。但是，如果所有的窗口都关闭的话那么这两个会话都会被杀死。

使用共享账户和 `tmux` 是结对编程最简单的方式，但是有时候我们并不是总是愿意和团队成员共享账号。下面我们来看看另一种方式。

5.3 使用独立账户和 Socket 结对编程

使用 `tmux` 提供的 `socket` 支持，可以创建让多个用户非常容易连接的 `tmux` 会话。

首先，我们创建两个新的用户账号：一个名为 `ted`，另一个名为 `barney`。

```
tmux@puzzles$ sudo adduser ted
tmux@puzzles$ sudo adduser barney
```

接下来，创建“tmux”分组以及 `/var/tmux` 文件夹用来保存共享会话。

```
tmux@puzzles$ sudo groupadd tmux
tmux@puzzles$ sudo mkdir /var/tmux
```

要更改 `/var/tmux` 目录的组权限，这样 `tmux` 组就能访问它了：

```
tmux@puzzles$ sudo chgrp tmux /var/tmux
```

然后修改文件夹的权限让新的文件都能被所有的 `tmux` 组内用户访问：

```
tmux@puzzles$ sudo chmod g+ws /var/tmux
```

最后，添加 `Ted` 和 `Barney` 用户到 `tmux` 组内。

```
tmux@puzzles$ sudo usermod -aG tmux ted
tmux@puzzles$ sudo usermod -aG tmux barney
```

创建并共享会话

之前，我们已经使用过 `new-session` 命令来创建会话，但是那会使用默认的 `socket` 位置，我们是无法对它进行操作的。因此我们在这里不会创建命名会话，而是通过 `-s` 参数来创建我们自己的会话。

我们先登录 `ted` 账户然后使用 `socket` 创建一个新的 `tmux` 会话：

```
ted@puzzles$ tmux -S /var/tmux/pairing
```

在另一个终端窗口里，可以登录 `barney` 然后连接到这个会话。连接时不必指定会话名，我们需要做的就是指定 `socket` 连接和 `socket` 文件的位置，就像这样：

```
barney@puzzles$ tmux -S /var/tmux/pairing attach
```

`barney` 用户现在已经连接到 `tmux` 会话而且他看到的就是 `ted` 用户所看到的内容。

需要注意的一点是，使用这种方法来使用 tmux 时，`.tmux.conf` 文件使用的配置是第一个启动这个会话的用户的配置。使用两个独立的账户并不意味着每个账户都可以在 tmux 会话里使用各自的配置文件，而是说他们可以根据其他的目的来定制他们的账户，然后在需要时都可以初始化他们所独有的 tmux 会话。

5.4 接下来做什么？

现在你已经学会了如何通过 tmux 与他人分享屏幕，你可以使用它来进行远程训练、在开源项目上进行即兴合作，或者是远程演示。

另外，还可以使用这个技术在你的生产服务器上开启 tmux 会话，加载监测工具或者控制台，然后从会话分离出来，让这些工具一直在后台运行。然后可以在你的机器上连接到这个会话，所有的一切都如你断开连接时那样。我曾经在我的开发环境上就做过类似的事情。我在一个 VPS 上配置了 tmux，然后在家里使用一台 iPad，一个 SSH 客户端以及一个蓝牙键盘就可以写代码了。即便是在 3G 网络下这样做依然很流畅。

结对编程和远程协作只是 tmux 应用在工作时提高工作效率的两个例子。在下一个章节，我们会学习在使用 tmux 窗口、面板和系统时其他更加高级的功能。

以备参考

命令	描述
<code>tmux -S [socket]</code>	使用 socket 创建一个新的会话
<code>tmux -S [socket] attach</code>	连接到一个已有的 socket 会话
<code>tmux new-session -t [existing session] -s [new session]</code>	创建一个到组会话的连接

第 6 章 工作流

就 tmux 自身来说，它只是一个添加了一些附加功能的另一个终端而已，只是显示了更多的终端会话。但是 tmux 让我们在这些会话里运行程序时更加方便，所以本章将会探讨一些常见、不常见的配置和命令，它们可能对你的日常工作有很大的帮助。你会学到管理面板和会话的高级方式，如何让 tmux 和 shell 一起工作，如何使用外部脚本扩展 tmux 命令，如何创建能执行数条命令的快捷键。我们先从管理窗口和面板开始。

6.1 高效使用面板和窗口工作

在本书中，你已经见到数种方式把 tmux 会话分割为多个面板和窗口。在本节，我们会学习使用面板和窗口工作的几种高级方式。

把面板变为窗口

面板很适合用来划分工作空间，但是有时我们需要把一个面板“弹出（pop out）”变为一个独立的窗口，这样看这部分内容就会更方便。tmux 有这样一个命令来实现这个功能。在任意面板内，按下 `PREFIX !` 键，tmux 就会依据当前面板创建一个新的窗口。

把窗口变为面板

有时候，我们需要合并一个工作空间，我们可以很简便地把窗口变为一个面板。为此，我们要提一下 `join-pane` 命令。

在“合并（join）”一个面板时，我们有可能把面板从一个会话移动到另一个会话里。此时需要指定源窗口和面板，后面跟随目标窗口和面板。如果不指定目标窗口，那么当前焦点窗口就会作为目标窗口。

下面我们通过创建一个带有两个窗口的 tmux 会话来演示：

```
$ tmux new-session -s panes -n first -d
$ tmux new-window -t panes -n second
$ tmux attach -t panes
```

现在，要把第一个窗口移动，作为第二个窗口的一个面板，按下 `PREFIX :` 键进入命令模式，然后输入这些：

```
join-pane -s 1
```

这句话的意思是“取出窗口（当窗口中有多个面板时则指取出面板，译者注）1 并把它添加到当前窗口”，因为我们没有指定一个目标窗口。

也可以使用这种方法来移动面板。如果第一个窗口有两个面板，可以像下面这样指定源面板，注意我们设置窗口从 1 开始编号，而面板从 0 开始编号：

```
join-pane -s 1.0
```

在这里，我们取出了第一个窗口的第一个面板然后把它添加到当前窗口。

更进一步地，甚至可以指定一个源会话，使用格式 `-t [session_name]:[window].[pane]` 指定一个目标窗口。

最大化和恢复面板

有时我们只是想让一个面板最大化显示一会，这样就可以细看它的内容，这时可以使用 `break-pane` 命令，然后再使用 `join-pane` 命令把它放回原处。这个操作做起来有些繁琐，因此我们编写一个脚本来实现这个功能。这是[链接](#)。

首先，我们释放 `UP` 箭头键，把它设置为最大化命令。然后，创建一个新的快捷键 `PREFIX UP` 来触发这个 `tmux` 命令串，配置如下：

```
unbind Up
bind Up new-window -d -n tmp \; swap-pane -s tmp.1 \; select-window -t tmp
```

在配置里，我们创建了一个名为 `tmp` 的新窗口。通过给它命名，可以在子序列命令里调用它。当使用 `-d` 参数创建窗口时，`tmux` 会在后台创建这个窗口而不是把焦点转到这个窗口上。然后使用 `swap-pane` 命令选取已经选择的面板和临时窗口的已有面板进行交换。

要恢复窗口，只需要使用 `swap-pane` 命令把面板从临时窗口交换到原来的窗口里，选择源面板，然后杀掉临时窗口。我们把这个命令序列绑定到 `PREFIX DOWN` 键，就像这样：

```
unbind Down
bind Down last-window \; swap-pane -s tmp.1 \; kill-window -t tmp
```

由于它使用了 `last-window` 命令来返回源窗口，因此这个过程看起来就像是把一个面板啪的一下最大化了，然后又啪的一下把它恢复原位置，这个简单的例子说明了 `tmux` 高度灵活性。我们可以通过一个简单的快捷键来自动化实现一系列命令。

在面板里执行命令

在第 3 章，我们已经学习了如何使用 shell 命令和 `send-keys` 在面板里启动程序，我们还可以让 tmux 在新建一个窗口或面板时自动执行命令。

假设有两台服务器，bums 和 smithers，分别是 web 服务器和数据库服务器。当启动 tmux 时我们想让 tmux 使用一个窗口的两个面板分别连接到这两台服务器上。

下面我们来创建一个新的名为 `servers.sh` 的脚本然后创建一个会话连接到两台服务器：

```
$ tmux new-session -s servers -d "ssh deploy@bums"
$ tmux split-window -v "ssh dba@smithers"
$ tmux attach -t servers
```

新建一个会话时，可以把要执行的命令作为最后一个参数传入到 tmux 中。在这里我们先是新建了一个会话然后在第一个窗口连接到 bums 服务器，然后从会话中分离出来。之后我们使用垂直分割切分窗口然后连接到 smithers 服务器。

这个配置有个副作用：从远程服务器上退出登录时，面板或窗口会关闭。

在 OS X 系统的同一目录下打开新面板

在 Linux 系统上创建 tmux 新的面板时，新面板使用的是当前面板的路径。但是在 OS X 系统上，新的面板会位于启动 tmux 会话时的那个目录。只需要做一点小小的工作，就可以在打开一个面板时捕捉它的工作路径然后自动地切换路径，就像 Linux 做的那样。

为此，我们使用 `send-keys` 命令来调用一个脚本把当前路径保存到环境变量中，然后这个脚本回调 `send-keys` 向拆分的窗口中发送命令，再把路径更改为环境变量中保存的那个路径。

首先，在主目录下创建一个名为 `~/tmux-panes` 的新文件，写入以下内容：

```
TMUX_CURRENT_DIR=`pwd`
tmux split-window $1
tmux send-keys "cd $TMUX_CURRENT_DIR;clear" C-m
```

然后编辑 `.tmux.conf` 文件来调用这个文件做垂直和水平分割。这里使用 `PREFIXI v` 键和 `PREFIXI n` 键，以防覆盖了当前已有的分割快捷键。代码如下：

```
unbind v
unbind n
bind v send-keys " ~/tmux-panes -h" C-m
bind n send-keys " ~/tmux-panes -v" C-m
```

就像在之前讨论过的，我们需要使用 `-v` 参数来水平分割窗口，使用 `-h` 参数来垂直分割窗口。

最后，为 `tmux-panes` 脚本添加执行权限：

```
$ chmod +x ~/tmux-panes
```

在重新加载 `.tmux.conf` 配置文件后就可以分割面板了。

这种方法的弊端是它看起来有点 hack。它把命令输入到已有的 `tmux` 窗口并执行脚本。也就是说它只能在一个有命令提示符的窗口里才能被触发。所以，如果你的主窗口在运行 `Vim`，这个命令是不会有有效的。即便是把 `send-keys` 命令换成 `run-shell` 命令也不会有效，因为新产生的 `shell` 也没有访问环境变的权限，因此它也就无法处理这个脚本了。但是这个脚本依然是一个比较方便的小技巧，通过自定义键盘快捷键，依然还保留了原始的命令。

6.2 管理会话

随着使用 `tmux` 越来越顺手，你会发现你会同时使用多个 `tmux` 会话。例如，你可能会为每个程序都开启一个 `tmux` 会话，这样就可以保持开发环境的相对独立。`tmux` 提供了多种特性能让你在管理这些会话时不会感到痛苦。

在会话间移动

单机上的所有 `tmux` 会话都通过一个服务器进行管理。也就是说我们能够在同一台机器上就可以实现在会话之间来回移动。

下面来演示这个过程，我们会启动两个分离的 `tmux` 会话，一个名为 `editor`，它打开了 `Vim`，一个名为 `processes` 的会话则在运行 `top` 命令，命令如下所示：

```
$ tmux new -s editor -d vim
$ tmux new -s processes -d top
```

可以这样连接到 `editor` 会话：

```
$ tmux attach -t editor
```

然后按下 `PREFIX` (键进入前一个会话，按下 `PREFIX`) 则可以跳转到下一个会话。

还可以使用 `PREFIX s` 键显示一个会话列表，这样就可以快速地从一個会话跳转到另一个会话。

你可以添加自定义的快捷键到 `.tmux.conf` 文件里来绑定 `switch-client` 命令。默认的配置应该是像这样：

```
bind -r ( switch-client -p
bind -r ) switch-client -n
```

如果你已经配置了多个工作空间，这样操作会极大地提高你的效率，而且它不需要分离会话再重新连接。

创建或连接到已有会话

到目前为止，我们学会了多种办法在任意时刻创建新的 tmux 会话。然而，事实上还可以判断一个 tmux 会话是否存在，如果存在的话就连接到它。

`has-session` 命令返回一个可以用在 shell 脚本里的布尔值。可以用它在 bash 脚本做一些类似这样的事情：

```
if ! tmux has-session -t remote; then
    exec tmux new-session -s development -d
    # other setup commands before attaching....
fi
exec tmux attach -t development
```

如果修改这个脚本让它通过参数读取会话名称，你就可以用它来连接或创建任意 tmux 会话。

在会话之间移动窗口

可以把一个会话的窗口移动到另一个会话里。如果已经在一个开发环境里打开了一个进程，现在想把它移动到另一个环境中，或者想合并工作空间时会非常有用。

`move-window` 命令被映射到快捷键 `PREFIX .` 键（英文句号键，译者注），这样可以很方便地把要移动的窗口作为当前焦点窗口，按下快捷键，然后输入目标会话即可。

下面演示一下这个过程，创建一个会话，一个名为 `editor`，一个名为 `processes` 分别运行了 `vim` 和 `top` 命令：

```
$ tmux new -s editor -d vim
$ tmux new -s processes -d top
```

我们会把 `processes` 会话的窗口移动到 `editor` 会话里。

首先，连接到 `processes` 会话，就像这样：

```
$ tmux attach -t processes
```

然后，按下 `PREFIX .` 键，然后在显示的命令行里输入 `editor`。

这会把 `processes` 会话里的唯一窗口移动出来，因此 `processes` 会话会自动关闭。如果连接到 `editor` 会话，就可以看到这两个窗口。

也可以使用 `shell` 命令来完成这个功能，因此不必在合并窗口时要连接会话。可以这样使用 `move-window` 命令：

```
$ tmux move-window -s processes:1 -t editor
```

这个命令的意思就是，把 `processes` 会话的第 1 个窗口移动到 `editor` 会话中。

6.3 tmux 和你的操作系统

既然 `tmux` 已经变成了你工作流的一部分，那么你肯定想让它和操作系统集成地越紧密越好。在本机，我们会向你展示多种方式，让你的 `tmux` 和操作系统一起工作。

使用一个不同的 Shell

在本书中，我们使用的 `shell` 环境都是 `bash`，但是如果你更喜欢 `zsh`，你依然可以使用所有的 `tmux` 优良特性。

可以在 `.tmux.conf` 文件里明确地设置默认的 `shell` 环境，就像这样：

```
set -g default-command /bin/zsh
set -g default-shell /bin/zsh
```

由于 `tmux` 只是一个终端复用器而并没有拥有独立的 `shell`，因此可以精确地指定使用哪个 `shell`。

默认启动 tmux

可以配置操作系统让它在打开一个终端时自动启动 `tmux`。通过使用会话名可以创建一个不存在的会话。

当 `tmux` 在运行时，它会把 `TERM` 变量设置为 `screen` 或者是 `default-terminal` 配置文件里的配置。可以在 `.bashrc` 文件（OS X 系统是 `.bash_profile` 文件）里使用这个变量来确定当前是否处于一个 `tmux` 会话中。我们在第 2 章配置了 `tmux` 终端为 `screen-256color`，因此可以使用这样一个脚本：

```
if [[ "$TERM" != "screen-256color" ]]
then
    tmux attach-session -t "$USER" || tmux new-session -s "$USER"
    exit
fi
```

如果没有在 tmux 会话里，我们会尝试连接到一个名为 \$USER 的会话里，也就是当前用户名。可以把这个值替换为任意你想要的值，在这里使用用户名能帮助我们避免冲突。

如果会话不存在，tmux 会抛出一个错误，shell 脚本会把这个错误解释为 false 值。然后脚本会继续执行右侧的命令，也就是创建一个以用户名作为名称的会话。然后退出脚本。

当 tmux 会话启动时，它会再次运行配置文件，但是这次它会看到我们处于一个 tmux 会话中，因此就会略过这部分的后续代码，然后继续执行配置文件的其他配置，确保所有环境变量都已被配置。

现在，创建一个新的终端会话时，我们就自动地连接到一个 tmux 会话中。但是请小心，因为你每次打开新的终端窗口时都会连接到相同的会话，在任意终端窗口里输入 exit 命令都会关闭所有连接到会话的终端窗口！

把程序输出记录到日志里

有时需要把一个终端会话的输出记录到日志文件里。我们在之前已经讨论过如何使用 capture-pane 和 save-buffer 命令来完成这些操作，但是实际上 tmux 可以通过 pipe-pane 命令把一个面板里的活动都记录到一个文本文件里。这很像许多 shell 里的 script 命令，使用 pipe-pane 命令可以选择打开或关闭这个功能，而且可以在一个程序已经运行之后再开始使用这个命令。

要激活这个功能，在命令模式输入命令 pipe-pane -o "mylog.txt" 。

-o 参数让我们打开了输出功能，也就是说如果再次发送相同的命令就可以把这个功能关掉。为了方便地执行这个命令，我们把它添加到配置文件里并绑定一个快捷键。像这样：

```
bind P pipe-pane -o "cat >>~/#W.log" \; display "Toggled logging to ~/#W.log"
```

现在可以按下 PREFIX P 命令来控制打开日志功能了。多亏了 display 命令（display-message 命令的简写），我们可以在状态栏看见日志文件名。display 命令和状态栏一样能访问相同的变量。

在状态栏添加电池电量显示

如果你在笔记本电脑上使用 tmux，你可能想在状态栏里显示电池剩余电量，尤其是终端在全屏状态下运行的时候。幸亏有 \$(shellcommand) 变量能够让这个事情变得相当容易。

现在我们把电池状态添加到配置文件里。我们抓取了一个简单的 shell 脚本，它能够获取剩余电池电量并把它写入主目录下名为 `battery` 的文件里。要让 tmux 能够使用这个脚本，要赋予它执行权限。在终端里运行这些命令：

```
$ wget --no-check-certificate \
https://raw.githubusercontent.com/richoH/dotfiles/master/bin/battery
$ chmod +x ~/battery
```

如果在终端里运行这个命令：

```
$ ~/battery Discharging
```

我们就会看到电池剩余电量的百分比。可以让 tmux 通过 `#{<command>}` 命令把它显示在状态栏里。所以，要在时钟之前显示电池电量，需要这样修改配置文件里 `status-right` 这一行：

```
set -g status-right "#{~/battery Discharging) | #[fg=cyan]%d %b %R"
```

重新加载 `.tmux.conf` 文件时，电池的剩余电量就会显示出来。

要想在电池充电时获取它的状态，需要执行这个命令：

```
$ ~/battery Charging
```

然后根据上面的方法，可以把这个命令添加到状态栏里。这部分的工作留给你来完成。

使用这种方法更深度地定制状态栏。只需要编写自己的脚本然后返回你想要显示的任意值，然后把它扔到状态栏里。

6.4 接下来做什么？

你已经学会了 tmux 的基础你就可以做很多事情，而且你现在已经对不同的配置有了一定的经验。tmux 的用户手册，可以在终端里获取，命令如下：

```
$ man tmux
```

这里面有完整的配置选项列表和所有 tmux 可用命令。

别忘了 tmux 本身也是在快速进化之中。下一个版本将会带来新的配置选项，会为你带来更高的灵活性。

现在你已经把 tmux 集成到你的工作流之中了，你可以尝试发掘一些其他的常用技术。例如，可以一起使用 tmux 和 Vim 来创建更高效的开发环境。你还可以在 tmux 会话里使用 irssi（一个终端界面的 IRC 客户端）和 Alpine（一个基于终端的邮件应用），每个程序占用你的一个面板，和你的文本编辑器并排排列，或是让它们运行在后台窗口里。然后你可以从会话中分离出来，过段时间再连接到 tmux 会话中，一如既往。

附录：本书使用的配置文件

```
# 本书使用的 .tmux.conf 文件

# 把前缀键从 C-b 更改为 C-a
set -g prefix C-a

# 释放之前的 Ctrl-b 前缀快捷键
unbind C-b

# 设定前缀键和命令键之间的延时
set -sg escape-time 1

# 确保可以向其它程序发送 Ctrl-A
bind C-a send-prefix

# 把窗口的初始索引值从 0 改为 1
set -g base-index 1

# 把面板的初始索引值从 0 改为 1
setw -g pane-base-index 1

# 使用 Prefix r 重新加载配置文件
bind r source-file ~/.tmux.conf \; display "Reloaded!"

# 分割面板
bind | split-window -h
bind - split-window -v

# 在面板之间移动
bind h select-pane -L
bind j select-pane -D
bind k select-pane -U
bind l select-pane -R

# 快速选择面板
bind -r C-h select-window -t :-
bind -r C-l select-window -t :+

# 调整面板大小
bind -r H resize-pane -L 5
bind -r J resize-pane -D 5
bind -r K resize-pane -U 5
bind -r L resize-pane -R 5

# 鼠标支持 - 如果你想使用的话把 off 改为 on
setw -g mode-mouse off
set -g mouse-select-pane off
set -g mouse-resize-pane off
```

```
set -g mouse-select-window off

# 设置默认的终端模式为 256 色模式
set -g default-terminal "screen-256color"

# 开启活动通知
setw -g monitor-activity on
set -g visual-activity on

# 设置状态栏的颜色
set -g status-fg white
set -g status-bg black

# 设置窗口列表的颜色
setw -g window-status-fg cyan
setw -g window-status-bg default
setw -g window-status-attr dim

# 设置活动窗口的颜色
setw -g window-status-current-fg white
setw -g window-status-current-bg red
setw -g window-status-current-attr bright

# 设置面板和活动面板的颜色
set -g pane-border-fg green
set -g pane-border-bg black
set -g pane-active-border-fg white
set -g pane-active-border-bg yellow

# 设置命令行或消息的颜色
set -g message-fg white
set -g message-bg black
set -g message-attr bright

# 设置状态栏左侧的内容和颜色
set -g status-left-length 40
set -g status-left "#[fg=green]Session: #S #[fg=yellow]#I #[fg=cyan]#P"
set -g status-utf8 on

# 设置状态栏右侧的内容和颜色
# 15% | 28 Nov 18:15
set -g status-right "#(~ / battery Discharging) | #[fg=cyan]%d %b %R"

# 每 60 秒更新一次状态栏
set -g status-interval 60

# 设置窗口列表居中显示
set -g status-justify centre

# 开启 vi 按键
setw -g mode-keys vi

# 在相同目录下使用 tmux-panes 脚本开启面板
```



```
unbind v
unbind n
bind v send-keys " ~/tmux-panes -h" C-m
bind n send-keys " ~/tmux-panes -v" C-m

# 临时最大化面板或恢复面板大小
unbind Up
bind Up new-window -d -n tmp \; swap-pane -s tmp.1 \; select-window -t tmp
unbind Down
bind Down last-window \; swap-pane -s tmp.1 \; kill-window -t tmp

# 把日志输出到指定文件
bind P pipe-pane -o "cat >>~/#W.log" \; display "Toggled logging to ~/#W.log"
```