tao-of-tmux Documentation

发布 *v1.0.2*

Tony Narlock

2020 年 04 月 18 日

Contents

Tao of tmux 的作者是 Tony Narlock。

作者英文版地址 <https://leanpub.com/the-tao-of-tmux/read>

翻译说明：我的翻译模式为，散步翻译，即有空就翻译翻译，欢迎协助翻译，在 issue 上说一哈就行。欢迎指出翻译错误的地方 https://github.com/talengu/tao-of-tmux-Chinese/issues

talen move it to sphinx.

Contents:

CHAPTER 1

前言

我所有的朋友几乎都在使用 tmux。我记得我们晚上出去喝酒，我们三个找一个桌子，掏出手机。这还是手机有“QWERTY”键的时代。

尽管放在家里的电脑开启睡眠模式或者关掉了，但我们经常访问的 IRC 频道中的用户名仍然存在于聊天室列表中。我们的黑色 terminal 上五颜六色的东西。我们使用 ConnectBot 连接云服务器并通过运行 `screen(1)` 重新连接聊天窗口。在凌晨 2 点，我们的土耳其咖啡到了，`|away` 状态消息被替换掉，我们开始工作。

有趣的是，即使我们知道对方的真实名字，我们有时也会用昵称相互打招呼。人际关系从线上到了线下。

似乎它是精心策划的，但我们每个人都陷入了生活的同一个潮起潮落。没有人告诉我们这样做，但一点一点地，我们逐步优化我们的个人和专业生活方式，以达到看似奇怪的目的地。

就像生活中的很多事情，当我随心所欲按导航走的时候，其实最后到达一个地方，这种常常会有发生。所以，当我写一本电脑软件的教育手册的时候，希望不是强塞给你 tmux，喜欢或者不喜欢，但告诉你 tmux 是什么，一些常用的使用技巧。

1.1 关于本书

我通过完全免费的 The Tao of tmux 帮助过成百上千人学习 tmux，这份材料也属于 tmuxp session manager 的一部分文档。现在，我将它拓展了一下，完善图，例子等，使之成为了一本书。

你只是想使用 tmux 的话，有一份 man 技术手册 manpage for tmux。然而 Manpages 很少解释清楚抽象概念，他们更多的是一种参考手册。在这本书中我总结了个人多年使用和教人使用 tmux 的经验。在这本书里，我们将 tmux 按它的组成拆分，从 servers 到 panes。当然也会包括一些 terminal 的基础知识照顾到那些自学者。我参加过许多开源项目，设计了一些有效的 terminal 的工作流程（workflows）。

tmux 真的很有用。它成为了我日常的一部分，除了官方的文档资源，我也写了受欢迎的 tmux 的配置文件configuration，和插件 tmux session manager。

我此刻正用运行在 tmux pane 里面的 vim，写此文档。

> I am writing this from vim running in a tmux pane, inside a window, in a session running on a tmux server, through a client.

对初学者的一句忠告：不要觉得你需要在一次就能掌握命令行和终端多路复用的概念。您可以根据自己的需要或兴趣选择自己喜欢的 tmux 概念。如果您还没有安装 tmux，请查看本书的 Installation section部分。

本书 twitter @TheTaoOfTmux 查看更新或者分享此文 share on Twitter!

1.2 代码等风格说明

强调符号如 `like this` ，这代表为代码部分。

以 开头的为 terminal 的命令（command）。`$ echo 'like this'`。在终端中输入时不需要输入 dollar 符号。想知道更多关于 dollar 提示符的意义，可查看在 Super User 网站上的 *What is the origin of the UNIX $ (dollar) prompt?*。

在 tmux 中，快捷键（shortcuts）需要有个预按键 prefix key 和其他键来组合命令。如 `Prefix` + `d` 代表将从 session 中脱离（detach）。通过 tmux 默认 `<Ctrl-b>`，用户也可以更改。这些内容在 *the prefix key* 部分和 *configuration* 章节。

1.3 本书主要内容

安装 installation和技术细节部分在附录部分。许多书在前几章就介绍安装，但是背景写的比较少，可能会使刚开始学习的人迷惑。还有一些特殊的章节，如 tmux on Windows 10。我在文章加了一些截图，方便读者可视理解。

tmux 初识（Thinking in tmux） 概览一下 tmux 的功能和它如何配合 GUI 来操作。你会了解 tmux 大概东西，从而使得更加愉快的写代码。

Terminal Fundamentals shows the text-based environments you' ll be dealing with. It' s great for those new to tmux, but also presents technical background for developers, who learned the ropes through examples and osmosis. At the end of this section, you' ll be more confident and secure using the essential components underpinning a modern terminal environment.

Practical usage covers common bread-and-butter uses for you to use tmux immediately.

Server gives life to the unseen workhorse behind the scenes powering tmux. You' ll think of tmux differently and may be impressed a client-server architecture could be presented to end users so seamlessly.

Sessions are the containers holding windows. You' ll learn what sessions are and how they help organize your workspace in the terminal. You' ll learn how to manipulate and rename and traverse sessions.

Windows are what you see when tmux is open in front of you. You' ll learn how to rename and move windows.

Panes are a terminal in a terminal. This is where you get to work and do your magic! You' ll learn how to create, delete, move between, and resize panes.

Configuration discusses customization of tmux and sets the foundation for how to think about `.tmux.conf` so you can customize your own.

Status bar and styling is devoted to the customization of the status line and colors in tmux. As a bonus, you' ll even learn how to display system information like CPU and memory usage via the status line.

Scripting tmux goes into command *aliases* and the advanced and powerful *Targets* and *Formats* concepts.

Technical stuff is a glimpse at tmux source code and how it works under the hood. You may learn enough to impress colleagues who already use tmux. If you like programming on Unix-like systems, this one is for you.

Tips and tricks wraps up with a whirlwind of useful terminal tutorials you can use with tmux to improve day to day development and administration experience.

Cheatsheets are organized tables of commands, shortcuts, and formats grouped by section.

1.4 打赏

If you enjoy my learning material or my open source software projects, please consider donating. Donations go directly to me and my current and future open source projects and are not squandered. Visit http://www.git-pull.com/support.html for ways to contribute.

1.5 书籍形式（Formats）

书本在 Leanpub 和 Amazon Kindle有销售。

免费的英文版本 free on the web。中文版本 https://tao-of-tmux.readthedocs.io

1.6 勘误说明（Errata）{#errata}

This is my first book. I am human and make mistakes.

If you find errors in this book, please submit them to me at tao.of.tmux nospam git-pull.com.

You can also submit a pull request via https://github.com/git-pull/tao-of-tmux.

I will update digital versions of the book with the changes where applicable.

1.7 感谢

Thanks to the contributors for spotting errors in this book and submitting errata through GitHub. In addition, readers like Graziano Misuraca, who looked through the book closely, providing valuable feedback.

Some copy, particularly in *cheatsheets*, comes straight out of the manual of tmux, which is ISC-licensed.

1.8 本书跟新和 tmux 的变动

本书基于 tmux 2.3，发布于 2016 年 9 月。

2017 年 1 月，作者发布到 Leanpub，还有一个更新的版本在 Kindle。

tmux does intermittently receive updates. I’ve accommodated many over the past 5 years on my personal configurations and software libraries set with continuous integration tests against multiple tmux versions. Sometimes, publishers overplay version numbers to make it seem as if it’s worth striking a new edition of a book over it. It’s effective for them, but I’d rather be honest to my readership.

If you’re considering keeping up to date with new features and adjustments to tmux, the `CHANGES` file in the project source serves as a way to see what’s updated between official releases.

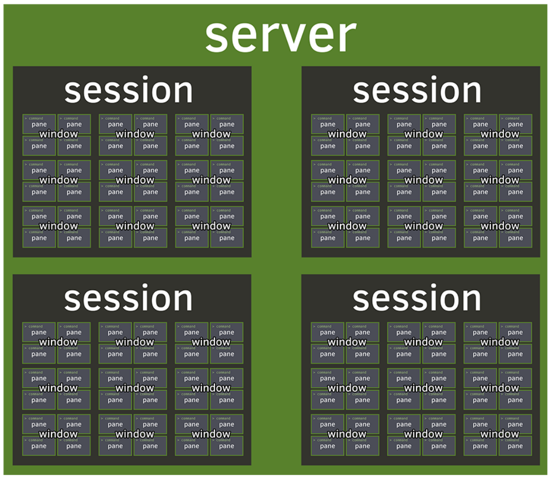
CHAPTER 2

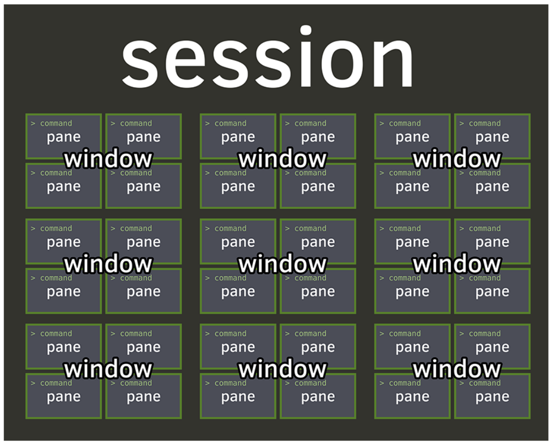
tmux 初识 {#thinking-tmux}

在计算机发展中，交互界面有两个王国：

1. 命令行界面交互
2. 图行界面交互

tmux，它存在在与 terminal 应用中，可以将 terminal 分割成多个块。



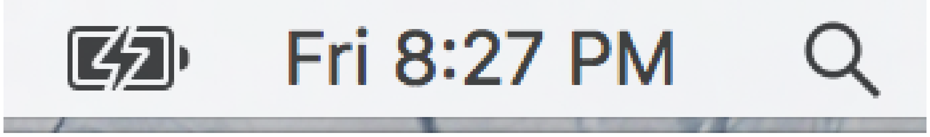


2.1 terminal 的窗口管理器

tmux 对于控制台来说，就像 windows 的 desktop 对于 GUI 应用。文字界面也有一个美丽的世界。使用 tmux，你可以：

- 多任务处理，运行多个程序
- 同一个窗口拥有多个指令输入的行（行文中叫 pane）
- 在一个工作会话 (session) 中可以有多个窗口 (window)
- 就像虚拟 windows 桌面一样，可以在 (session) 中相互切换

就像图形的桌面一样，也可以在 tmux 的状态栏放一个日期/时间。



2.2 多任务处理

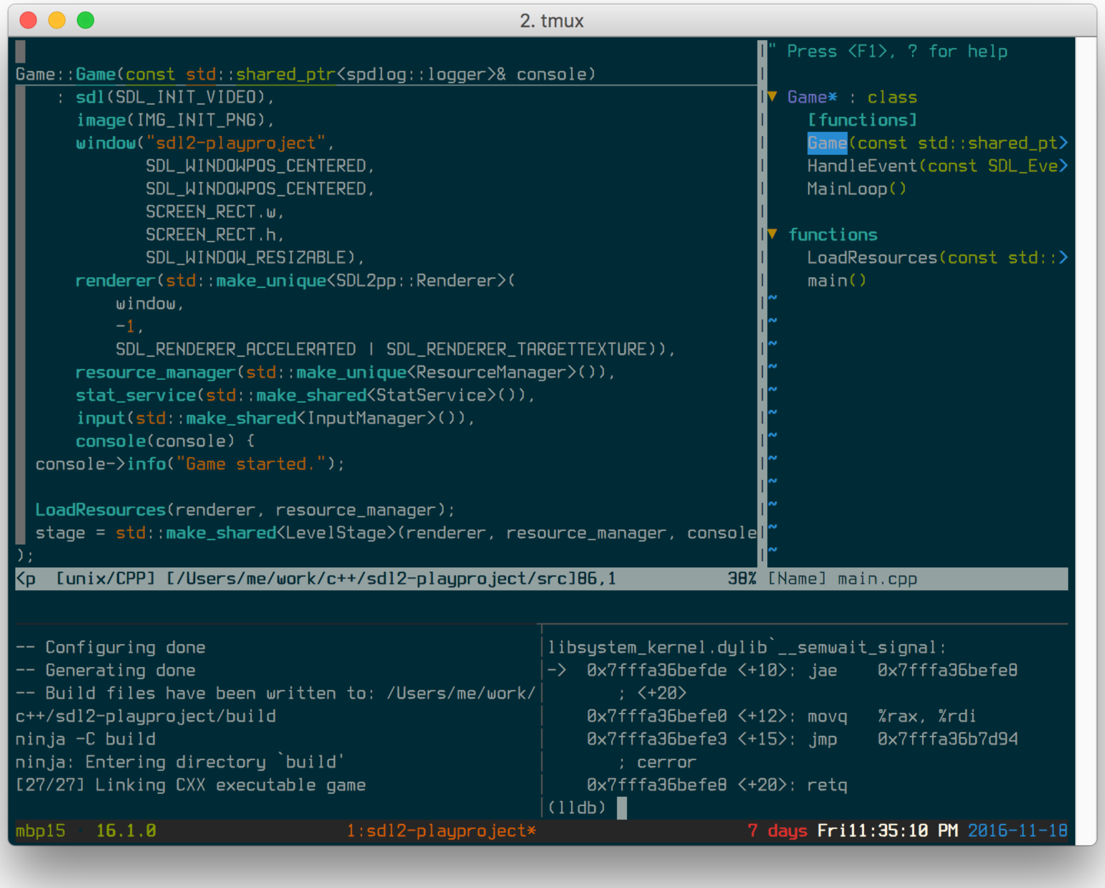
tmux 在一个窗口保持多个 termianl。“tmux” 缩写来自 **T**erminal **Mu**ltiple**x**er。

除了一个屏幕上的多个终端之外，tmux 还允许您创建和链接多个 “窗口” 附加在 tmux 会话上。

更好的是，你可以复制，粘贴和滚动。对图形也没有要求，所以你有完全的操控能力，即使你是通过 SSH 连接或在没有显示服务器的系统上工作，如 X。

下面有些常见的场景：

- 在 pane a 中运行 “$tail-F/var/log/apache2/error.log” 查阅正在改变的日志文件。
- 运行 file watcher，如 watchman，gulp-watch, grunt-watch, guard, 或者 entr。监测文件更改，可设定后续命令：
  - rebuild LESS or SASS files, minimize CSS and/or assets and static files
  - lint with linters, like cpplint, Cppcheck, rubocop, ESLint, or Flake8
  - rebuild with `make` or `ninja`
  - reload Express server
  - 运行一些自定义的命令
- 运行一个 text editor，如 vim，emacs，pico，nano，等等。运行一个主 pane，用其他两个，一个 CLI commands ，另一个用来使用 `make` 或者 `ninja` 命令进行编译。



2. tmux

```
Game::Game(const std::shared_ptr<spdlog::logger>& console)
    : sdl(SDL_INIT_VIDEO),
      image(IMG_INIT_PNG),
      window("sdl2-playproject",
             SDL_WINDOWPOS_CENTERED,
             SDL_WINDOWPOS_CENTERED,
             SCREEN_RECT.w,
             SCREEN_RECT.h,
             SDL_WINDOW_RESIZABLE),
      renderer(std::make_unique<SDL2pp::Renderer>(
          window,
          -1,
          SDL_RENDERER_ACCELERATED | SDL_RENDERER_TARGETTEXTURE)),
      resource_manager(std::make_unique<ResourceManager>()),
      stat_service(std::make_shared<StatService>()),
      input(std::make_shared<InputManager>()),
      console(console) {
  console->info("Game started.");

  LoadResources(renderer, resource_manager);
  stage = std::make_shared<LevelStage>(renderer, resource_manager, console
);
<p  [unix/CPP] [/Users/me/work/c++/sdl2-playproject/src]86,1          38%
```

```
" Press <F1>, ? for help

▼ Game* : class
    [functions]
    Game(const std::shared_pt>
    HandleEvent(const SDL_Eve>
    MainLoop()

▼ functions
    LoadResources(const std::>
    main()
[Name] main.cpp
```

```
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/me/work/
c++/sdl2-playproject/build
ninja -C build
ninja: Entering directory `build'
[27/27] Linking CXX executable game
```

```
libsystem_kernel.dylib`__semwait_signal:
->  0x7fffa36befde <+10>: jae    0x7fffa36befe8
      ; <+20>
    0x7fffa36befe0 <+12>: movq   %rax, %rdi
    0x7fffa36befe3 <+15>: jmp    0x7fffa36b7d94
      ; cerror
    0x7fffa36befe8 <+20>: retq
(lldb)
```

```
mbp15   16.1.0              1:sdl2-playproject*              7 days Fri11:35:10 PM 2016-11-18
```

vim + building a C++ project w/ CMake + Ninja using entr to rebuild on file changes, LLDB bottom right

使用 tmux，可以便利地做个 IDE 开发界面，快来试一试。

2.3 在后台运行程序

Sometimes, GUI applications will have an option to be sidelined to the system tray to run in the background. The application is out of sight, but events and notifications can still come in, and the app can be instantly brought to the foreground.

In tmux, a similar concept exists, where we can “detach” a tmux session.

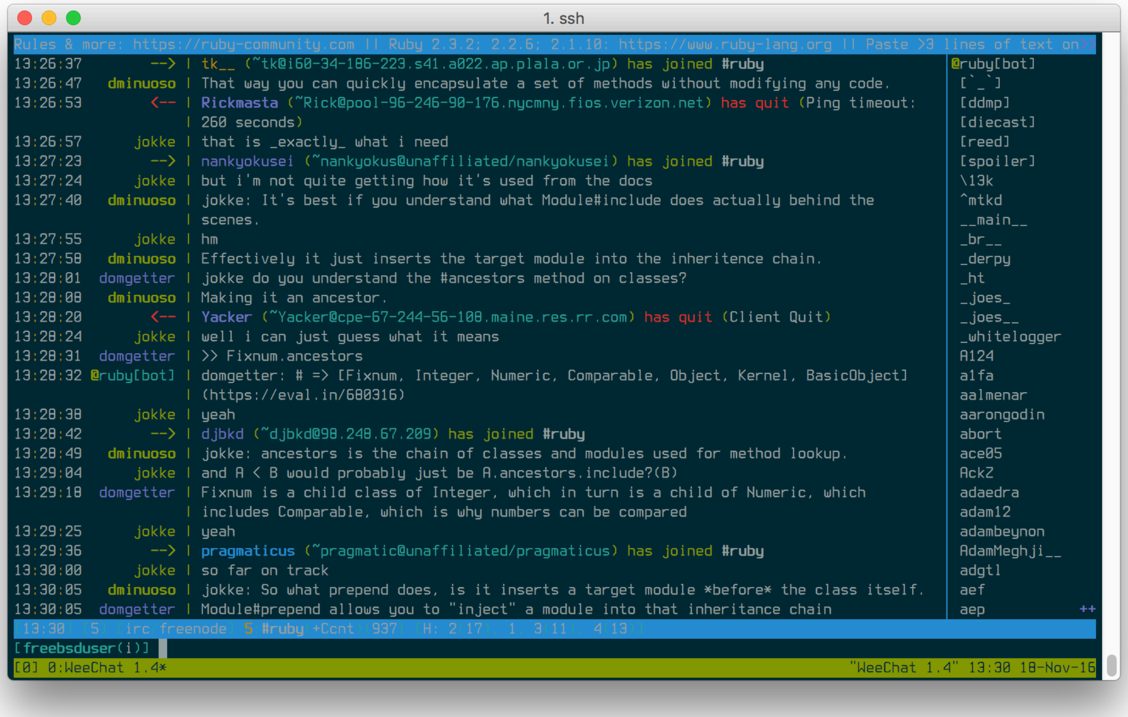
Detaching can be especially useful on:

- Local machines. You start all your normal terminal applications within a tmux session, you restart X. Instead of losing your processes as you normally would if you were using an X terminal, like xterm or konsole, you’d be able to `tmux attach` after and find all the processes inside that were alive and kicking all along.

- Remote SSH applications and workspaces you run in tmux. You can detach your tmux workspace at work before you clock out, then the next morning, reattach your session. Ahhh. Refreshing. :)
- Those servers you rarely log into. Perhaps, a cloud instance you log into 9 months later, and as a reflex, `tmux attach` to see if there is anything on there. And boom, you' re back in a session you' ve forgotten about, but still jogs your memory to what you were tweaking or fixing. It' s like a hack to restore your memory.

2.4 Powerful combos

Chatting on irssi or weechat, one of the "classic combos" , along with a bitlbee server to manage AIM, MSN, Google Talk, Jabber, ICQ, even Twitter. Then, you can detach your IRC and "idle" in your favorite channels, stay online on instant messengers, and get back to your messages when you return.



```
Rules & more: https://ruby-community.com || Ruby 2.3.2; 2.2.6; 2.1.10: https://www.ruby-lang.org || Paste >3 lines of text on>>
13:26:37         --> | tk__ (~tk@i60-34-186-223.s41.a022.ap.plala.or.jp) has joined #ruby          @ruby[bot]
13:26:47    dminuoso | That way you can quickly encapsulate a set of methods without modifying any code.   [`_`]
13:26:53         <-- | Rickmasta (~Rick@pool-96-246-90-176.nycmny.fios.verizon.net) has quit (Ping timeout:   [ddmp]
                     | 260 seconds)                                                                 [diecast]
13:26:57       jokke | that is _exactly_ what i need                                                 [reed]
13:27:23         --> | nankyokusei (~nankyokus@unaffiliated/nankyokusei) has joined #ruby           [spoiler]
13:27:24       jokke | but i'm not quite getting how it's used from the docs                        \13k
13:27:40    dminuoso | jokke: It's best if you understand what Module#include does actually behind the   ^mtkd
                     | scenes.                                                                      __main__
13:27:55       jokke | hm                                                                           _br__
13:27:58    dminuoso | Effectively it just inserts the target module into the inheritence chain.     _derpy
13:28:01   domgetter | jokke do you understand the #ancestors method on classes?                     _ht
13:28:08    dminuoso | Making it an ancestor.                                                       _joes_
13:28:20         <-- | Yacker (~Yacker@cpe-67-244-56-108.maine.res.rr.com) has quit (Client Quit)   _joes__
13:28:24       jokke | well i can just guess what it means                                           _whitelogger
13:28:31   domgetter | >> Fixnum.ancestors                                                          A124
13:28:32 @ruby[bot] | domgetter: # => [Fixnum, Integer, Numeric, Comparable, Object, Kernel, BasicObject]   a1fa
                     | (https://eval.in/680316)                                                     aalmenar
13:28:38       jokke | yeah                                                                         aarongodin
13:28:42         --> | djbkd (~djbkd@98.248.67.209) has joined #ruby                                abort
13:28:49    dminuoso | jokke: ancestors is the chain of classes and modules used for method lookup.   ace05
13:29:04       jokke | and A < B would probably just be A.ancestors.include?(B)                      AckZ
13:29:18   domgetter | Fixnum is a child class of Integer, which in turn is a child of Numeric, which   adaedra
                     | includes Comparable, which is why numbers can be compared                    adam12
13:29:25       jokke | yeah                                                                         adambeynon
13:29:36         --> | pragmaticus (~pragmatic@unaffiliated/pragmaticus) has joined #ruby           AdamMeghji__
13:30:00       jokke | so far on track                                                              adgtl
13:30:05    dminuoso | jokke: So what prepend does, is it inserts a target module *before* the class itself.   aef
13:30:05   domgetter | Module#prepend allows you to "inject" a module into that inheritance chain    aep          ++
[13:30] [5] [irc/freenode] 5:#ruby(+Ccnt){937} [H: 2(17), 1, 3(11), 4(13)]
[freebsduser(i)]
[0] 0:WeeChat 1.4*                                                             "WeeChat 1.4" 13:30 18-Nov-16
```

Chatting on weechat w/ tmux

Some keep development services running in a session. Hearty emphasis on *development*, you probably will want to daemonize and wrap your production web applications, using a tool like supervisor, with its own safe environmental settings.

让多个用户将其客户机附加到同一个会话，可以结对编程 (pair programming) 了。如果在打开同一个 session，其他人会看到相同的东西，相同的输入，相同的激活的 window 和 pane。

The above are just examples; any general workspace you' d normally use in a terminal could work, especially projects or repetitive efforts you multitask on. The tips and tricks section will dive into specific flows you can use today.

```
Q> ### 问题: tmux sessions 在系统重启后还会自动运行嘛?
Q>
Q> 不行。重启会关闭 tmux server 和 tmux 上正在运行的各种程序。
Q>
Q> Thankfully, the modern server can stay online for a long time. Even for
Q> consumer laptops and PC's with a day or two uptime, having tmux persist
Q> tasks for organizational purposes is satisfactory to run it.
Q>
Q> It comes as a disappointment, because some are interested in being able to
Q> persist a tree of processes after restart. It goes out of the scope of what
Q> tmux is meant to do.
Q>
Q> For tasks you repeat often, you can always use a tool, like
Q> [tmuxp](https://github.com/tony/tmuxp), [tmuxinator](https://github.com/tmuxinator/tmuxinator),
Q> or [teamocil](https://github.com/remiprev/teamocil), to resume common
Q> sessions.
Q>
Q> Besides session managers, [tmux-resurrect](https://github.com/tmux-plugins/tmux-resurrect)
Q> attempts to preserve running programs, working directories, and
Q> so on within tmux. The benefit with tmux-resurrect is there's no JSON/YAML
Q> config needed.
```

2.5 小节

tmux 是对终端工具带的一个丰富的补充。它有助于弥补多任务处理和工作空间组织之间的差距，因为没有图形用户界面会不太方便。此外，它还提供了一种将工作区放到后台，稍后可以重新连接（reattach）。

在下一小节，我们会接触一些 terminal 的基本操作，进一步深入 tmux。

CHAPTER 3

Terminal 基础知识（fundamentals）{#terminal-fundamentals}

在使用 tmux ，我们回顾一下命令行操作的基本知识。通常，我们使用命令行根据我们的使用经验和肌肉记忆，我们中很大一部人没有了解到工具之间的联系。

经验丰富的开发者对 Zsh, Bash, iTerm2, konsole, /dev/tty, shell scripting 等比较熟悉。如果使用 tmux，你将经常和这些工具打交道，不管你是使用 GUI 界面，还是使用 SSH 连接远程服务器。

如果你想如何在系统 kernel 层（数据结构等等）处理进程和 TTY 的细节可以参考 Marshall Kirk McKusick 的 *The Design and Implementation of the FreeBSD Operating System (2nd Edition)*, 尤其是 Chapter 4, *Process Management* and Section 8.6, *Terminal Handling*。和 Linus Åkesson 写的 *The TTY demystified* (在线) 深入 TTY，这些会帮助你理解。

从 Unix，4.2 BSD 等的历史中，我们可以找到更多相关知识，我们可以讨论一整天。我们可以从多个角度来看，比如 c 语言或者来自 Unix/BSD 谱系，或者是 Linux，GNU 等等。就像《权力的游戏》一样; 你可以从多个故事中找到线索。给一些好的 YouTube 视频资源 Marshall Kirk McKusick 讲的 *A Narrative History of BSD*、AT&T *The UNIX Operating System*、Stephen R. Bourne *Early days of Unix and design of sh*。

3.1 POSIX 标准

操作系统如 macOS () Operating systems like macOS (formerly OS X), Linux, and the BSDs, follow something similar to the POSIX specification in terms of how they square away various responsibilities and interfaces of the operating system. They’re categorized as “Mostly POSIX-compliant”.

In daily life, we often break compatibility with POSIX standards for reasons of sheer practicality. Operating systems, like macOS, will drop you right into Bash. `make(1)`, a POSIX standard, is GNU Make on macOS by default. Did you know, as of September 2016, POSIX Make has no conditionals?

I’ m not saying this to take a run at purists. As someone who tries to remain compatible in my scripting, it gets hard to do simple things after a while. On FreeBSD, the default Make (PMake) uses dots between conditionals:

```
.IF

.ENDIF
```

But on most Linux systems and macOS, GNU Make is the default, so they get to do:

```
IF

ENDIF
```

This is one of the many tiny inconsistencies that span operating systems, their userlands, their binary / library / include paths, and adherence / interpretation of the Filesystem Hierarchy Standard or whether they follow their own.

Find your path

Most operating systems inspired by Unix (BSD’ s, macOS, Linux) will allow you to get the info of your systems’ filesystem hierarchy via `hier(7)`.

```
$ man hier
```

These differences add up. A good deal of software infrastructure out there exists solely to abstract the differences across them. For example: CMake, Autotools, SFML, SDL2, interpreted programming languages, and their standard libraries are dedicated to normalizing the banal differences across BSD-derivatives and Linux distributions. Many, many `#ifdef` preprocessor directives in your C and C++ applications. You want open source, you get choice, but be aware; there’ s a lot of upkeep cost in keeping these upstream projects (and even your personal ones) compatible. But I digress, back to terminal stuff.

Why does it matter? Why bring it up? You’ ll see this stuff everywhere. So, let’ s separate the usual suspects into their respective categories.

3.2 Terminal interface

The terminal interface can be best introduced by citing official specification, laying out its technical properties, interfaces, and responsibilities. This can be viewed in its POSIX specification.

This includes TTYs, including text terminals and X sessions within them. On Linux / BSD systems, you can switch between sessions via `<ctrl-alt-F1>` through `<ctrl-alt-F12>`.

3.3 Terminal emulators

GUI Terminals: Terminal.app, iterm, iterm2, konsole, lxterm, xfce4-terminal, rxvt-unicode, xterm, roxterm, gnome terminal, cmd.exe + bash.exe

3.4 Shell languages {#shell-languages}

Shell languages are programming languages. You may not compile the code into binaries with `gcc` or `clang`, or have shiny npm package manager for them, but a language is a language.

Each shell interpreter has its own language features. Like with shells, many will resemble the POSIX shell language and strive to be compatible with it. Zsh and Bash should be able to understand POSIX shell scripts you write, but not the other way around (we will cover this in *shell interpreters*).

The first line of shell file is the shebang statement, which points to the interpreter to run the script in. They normally use the `.sh` extension, but they can also be `.zsh`, `.csh` and so on if they' re for a specific interpreter.

Zsh scripts are implemented by the Zsh shell interpreter, Bash scripts by Bash. But the languages are not as closely regulated and standardized as, say, C++' s standards committee workgroups or python' s PEPs. Bash and Zsh take features from Korn and C Shell' s languages, but without all the ceremony and bureaucracy other languages espouse.

3.5 Shell interpreters (Shells) {#shells}

Examples: POSIX sh, Bash, Zsh, csh, tcsh, ksh, fish

Shell interpreters *implement* the shell language. They are a layer on top of the kernel and are what allow you, interactively, to run commands and applications inside them.

As of October 2016, the latest POSIX specification covers in technical detail the responsibilities of the shell.

For shells and operating systems: each distro or group does their own darn thing. On most Linux distributions and macOS, you' ll typically be dropped into Bash.

On FreeBSD, you may default to a plain vanilla `sh` unless you specify otherwise during the installation process. In Ubuntu, `/bin/sh` used to be `bash` (Bourne Again Shell) but was replaced with `dash` (Debian Almquist Shell). So, here, you are thinking "hmm, `/bin/sh`, probably just a plain old POSIX shell"; however, system startup scripts on Ubuntu used to allow non-POSIX scripting via Bash. This is because specialty *shell languages*, such as Bash and Zsh, add helpful and practical features, but they' re not portable. For instance, you would need to install the Zsh interpreter across all your systems if you rely on Zsh-specialized scripting. If you conformed with POSIX shell scripting, your scripting would have the highest level of compatibility at the cost of being more verbose.

Recent versions of macOS include Zsh by default. Linux distributions typically require you to install it via package manager and install it to `/usr/bin/zsh`. BSD systems build it via the port system, `pkg(8)` on FreeBSD, or `pkg_add(1)` on OpenBSD, and it will install to `/usr/local/bin/zsh`.

It’s fun to experiment with different shells. On many systems, you can use `chsh -s` to update the default shell for a user.

The other thing to mention is, for `chsh -s` to work, you typically need to have it added to `/etc/shells`.

3.6 小节

To wrap it up, you will hear people talking about shells all the time. Context is key. It could be:

- A generic way to refer to any terminal you have open. “Type `$ top` into your shell and see what happens.” (Press q to quit.)
- A server they have to log into. Before the era of the cloud, it would be popular for small hosts to sell “C Shells” with root access.
- A shell within a tmux *pane*.
- If scripting is mentioned, it is likely either the script file, an issue related to the scripts’ behavior, or something about the shell language.

But overall, after this overview, go back to doing what you’re doing. If shell is what people say and they understand it, use it. The backing you have here should make you more confident in yourself. These days, it’s an ongoing battle catching our street smarts up with book smarts.

In the next chapter, we will touch some terminal basics before diving deeper into tmux.

CHAPTER 4

开始使用（Practical usage）{#practical-usage}

好的让我们开始吧！打开 terminal 输入 `tmux` 按回车键 enter。

```
$ tmux
```

欢迎进入 tmux 的世界。

4.1 前缀组合快捷键（prefix key ）{#prefix-key}

我们通过带前缀的组合快捷键向 tmux 输命令。我们可以拆分窗口（windows），移动窗口，切换窗口，切换会话（sessions），或者自定义的命令。

这是一个我们必须克服的困难。

这有点像打街头霸王（*Street Fighter*）

在游戏中，玩家输入一套组合健，控制角色飞转踢脚和射击火球。玩家的游戏学习，变得更加习惯连击，他们凭直觉重复动作，发展出肌肉记忆。

不知道使用组合键使用 tmux，你会死在水中的。

如果使用用 Vim，Emacs 或其他 (Terminal User Interface，TUI) 终端应用程序。如果尚未将组合命令的概念内在化，那就现在开始吧。在 TUI / GUI 应用程序中的组合命令的经验将有助于你理解。

记住一个组合键，就会减少一次手移开操作鼠标的操作。专注在短时记忆上，把工作一次性做完，减少错误的产生。

4.1.1 以前使用过 GNU Screen？

通过 tmux 配置文件设置 tmux 的 prefix key 。在文件 ~/.tmux.conf 中设置 prefix：

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

于是把 prefix 键设置成 screen(1) (另一个终端复用工具) 中的 prefix key 。

默认的 leader prefix 是 <Ctrl-b>。当按住 control 的时候，同时按 b。

4.1.2 发送 tmux 命令组合键

训练:

1. 按下 control 键保持
2. 再按下 b 键保持
3. 同时释放两个键

多尝试几次。多做几次你就会觉得自然而然记住快捷键。

接下来，训练:

<Ctrl-b> d 组合键

1. 按下 control 键保持
2. 按下 b 键保持
3. 同时释放两个键
4. 轻击打 d 键

你刚刚发出了使用 tmux 的第一次组合键命令，看你跳出了 tmux。

你已经断开和 tmux session 的连接。可通过 $ tmux attach 再次连接上。

4.1.3 Nested tmux sessions

You can also send the prefix key to *nested* tmux sessions. For instance, if you’re inside a tmux client on a *local* machine and you SSH into a *remote* machine in one of your panes, on the remote machine, you can attach the client via tmux attach as you normally would. To send the prefix key to the machine’s tmux client, not your local one, hit the prefix key again.

So, if your prefix key is the default, <Ctrl-b>, do <Ctrl-b> + b again, *then* hit the shortcut for what you want to do.

Example: If you wanted to create a window on the remote machine, which would normally be <Ctrl-b> + c locally, it’d be <Ctrl-b> + b + c.

Hereinafter, the book will refer to shortcuts by Prefix. Instead of <Ctrl-b> + d, you will see Prefix + d.

4.2 Session persistence and the server model

If you use Linux or a similar system, you' ve likely brushed through Job Control, such as `fg(1)`, `jobs(1)`. tmux behavior feels similar, like you ran `<Ctrl-z>` except, technically, you were in a "job" all along. You were just using a client to view it.

Another way of understanding it: `<Ctrl-b>` + `d` closed the client connection, therefore, 'detached' from the session.

Your tmux client disconnected from the server instance. The session, however, is still running in the background.

4.3 It' s all commands

Multiple roads can lead you to the same behavior. *Commands* are what tmux uses to define instructions for setting options, resizing, renaming, traversing, switching modes, copying and pasting, and so forth.

- *Configs* are the same as automatically running commands via `$ tmux command`.
- Internal tmux commands via `Prefix` + `:` prompt.
- Settings defined in your configuration can also set shortcuts, which can execute commands via key-bindings via `bind-key`.
- Commands called from CLI via `$ tmux cmd`
- To pull it all together, *source code* files are prefixed `cmd-`.

4.4 小节

We' ve established tmux automatically creates a server upon starting it. The server allows you to detach and later reattach your work. The keyboard sequences you send to tmux require understanding how to send the prefix key.

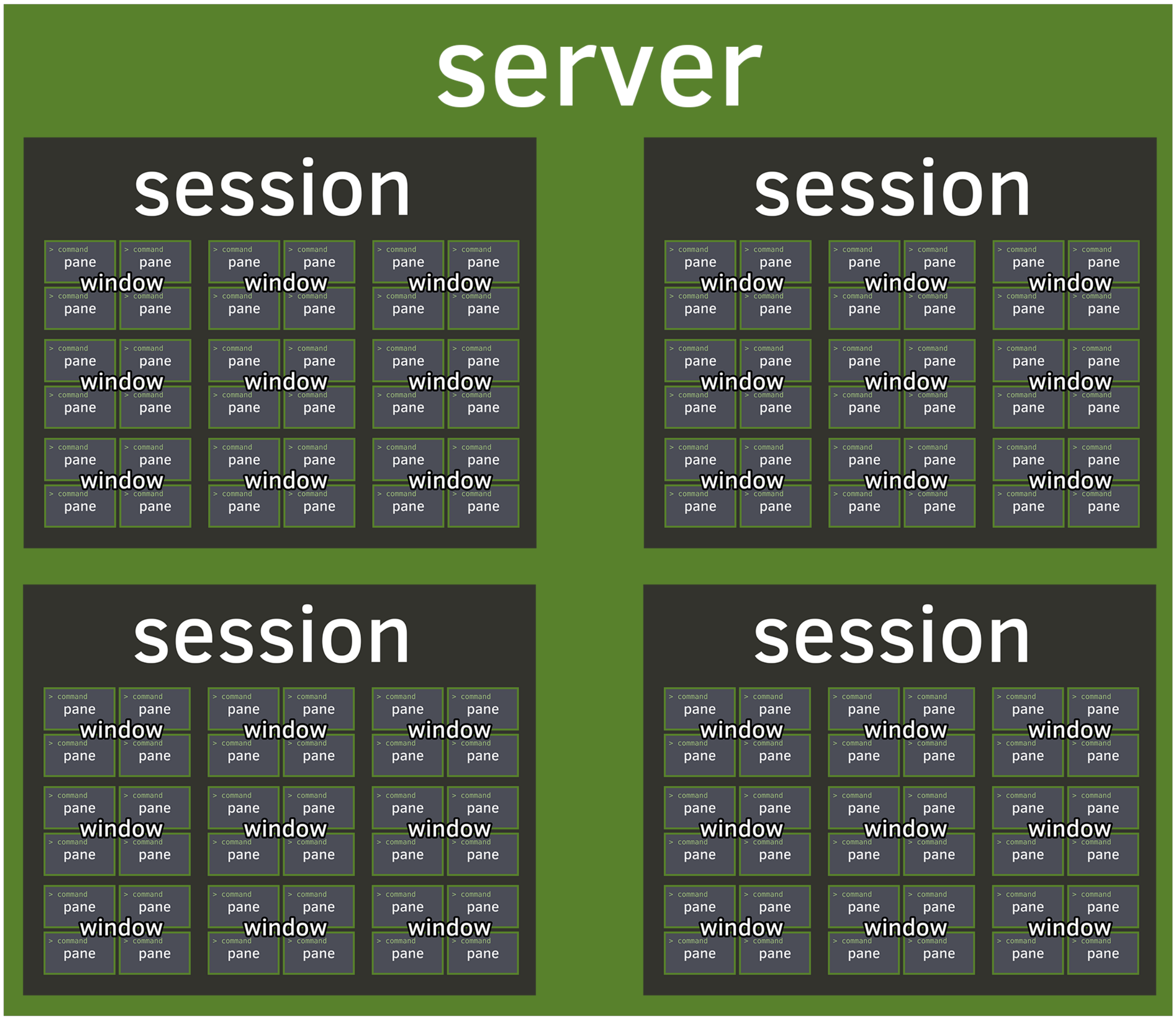
Keyboard sequences, configuration, and command line actions all boil down to the same core commands inside tmux. In our next chapter, we will cover the server.

CHAPTER 5

服务（Server ）{#server}

服务上面建立住 sessions 、windows 、panes 。

开启 tmux，通过 socket 机制连接一个 tmux 服务（server）。与你交互的其实是 tmux 的 client 客户端。本节我们将到启用这个 server 引擎，应用程序可以一次持续数月甚至数年。



5.1 What? tmux is a server?

Often, when “server” is mentioned, what comes to mind for many may be rackmounted hardware; to others, it may be software running daemonized on a server and managed through a utility, like upstart, supervisor, and so on.

Unlike web or database software, tmux doesn’t require specialized configuration settings or creating a service entry to start things.

tmux uses a client-server model, but the server is forked to the background for you.

5.2 Zero config needed

You don’t notice it, but when you use tmux normally, a server is launched and being connected via a client.

tmux is so streamlined, the book could continue to explain usage and not even mention servers. But, I’ d rather you have a true understanding of how it works on systems. The implementation feels like magic, while living up to the unix expectations of utilitarianism. One cannot deny it’ s exquisitely executed from a user experience standpoint.

How is it utilitarian? We’ ll go into it more in future chapters, where we dive into *Formats*, *Targets*, and tools, such as libtmux I made, which utilize these features.

It surprises some, because servers often beget a setup process. But servers being involved doesn’ t entail hours of configuration on each machine you run on. There’ s no setup.

When people think server, they think pain. It invokes an image of digging around `/etc/` for configuration files and flipping settings on and off just to get basic systems online. But not with tmux. It’ s a server, but in the good way.

5.3 Stayin’ alive

The server part of tmux is how your sessions can stay alive, even after your client is detached.

You can detach a tmux session from an SSH server and reconnect later. You can detach a tmux session, stop your X server in Linux/BSD, and reattach your tmux session in a TTY or new X server.

The tmux server won’ t go away until all sessions are closed.

5.4 Servers 包含 sessions

一个 server 包含一个或多个 sessions.

Starting tmux after a server already is running will create a new session inside the existing server.

```
W> ### Advanced: Multiple servers
W>
W> tmux is nimble. To use a separate server, pass in the `-L` flag to any
W> command.
W>
W> `tmux -L moo` - connect to server under socket name "moo" and attach
W> a new session. Create server if none already exists for socket.
W>
W> `tmux -L moo attach` will attempt to re-attach a session if one exists.
```

5.5 How servers are "named"

The default name for the server is `default`, which is stored as a socket in `/tmp`. The default directory for storing this can be overridden via setting the `TMUX_TMPDIR` environment variable.

So, something like:

```
$ export TMUX_TMPDIR=$HOME
$ tmux
```

Will give you a tmux directory created within your `$HOME` folder. On OS X, your home folder will probably be something like `/Users/yourusername`. On other systems, it may be `/home/yourusername`. If you want to find out, type `$ echo $HOME`.

5.6 Clients

Servers will have clients (you) connecting to them.

When you connect to a session and see windows and panes, it' s a client connection into tmux.

You can retrieve a list of active client connections via:

```
$ tmux list-clients
```

These commands and the other `list-` commands, in practice, are rare. But, they are part of tmux scriptability should you want to get creative. The *scripting tmux* chapter will cover this in greater detail.

5.7 Clipboard {#clipboard}

tmux clients 拥有一个共享 clipboard 的特性，在 essions, windows, 和 panes 都有用。

Much like vi, tmux handles copying as a mode in which a pane is temporarily placed. When inside this mode, text can be selected and copied to the *paste buffer*, tmux' s clipboard.

The default key to enter copy mode is `Prefix` + `[`.

1. From within, use `[space]` to enter copy mode.
2. Use the arrow keys to adjust the text to be selected.
3. Press `[enter]` to copy the selected text.

The default key to paste the text copied is `Prefix` + `]`.

> *Vi-like copy-paste*
>
> In your *config*, put this:

```
# Vi copypaste mode
set-window-option -g mode-keys vi
bind-key -t vi-copy 'v' begin-selection
bind-key -t vi-copy 'y' copy-selection
```

In addition to the “copy mode”, tmux has advanced functionality to programmatically copy and paste. Later in the book, the *Capturing pane content* section in the *Scripting tmux* chapter goes into `$ tmux capture-pane` and how you can use *targets* to copy pane content into your paste buffer or files with `$ tmux save-buffer`.

5.8 小节

The server is one of the fundamental underpinnings of tmux. Initialized automatically to the user, it persists by forking into the background. Running behind the scenes, it ensures sessions, windows, panes, and buffers are operating, even when the client is detached.

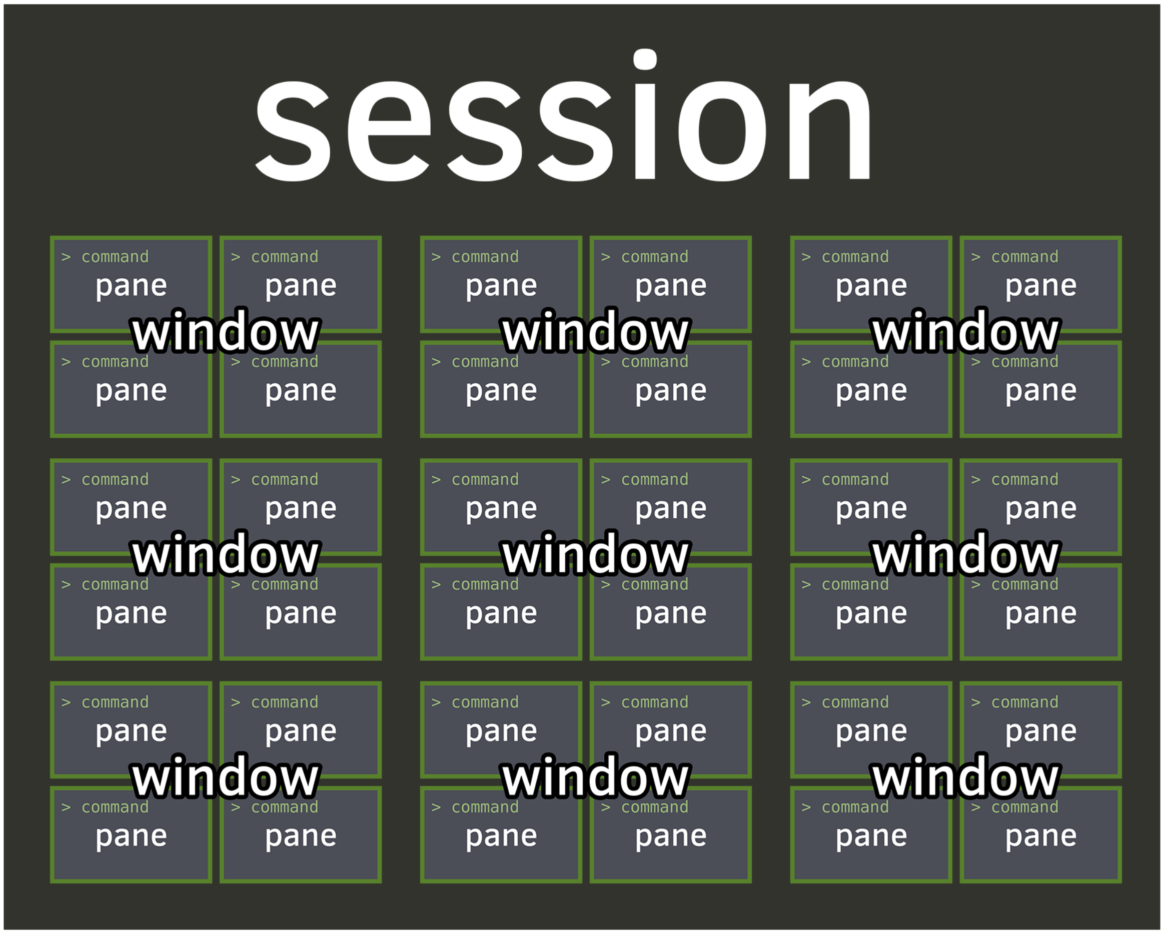
The server can hold one or more *sessions*. You can copy and paste between sessions via the clipboard. In the next chapter, we will go deeper into the role sessions play and how they help you organize and control your terminal workspace.

CHAPTER 6

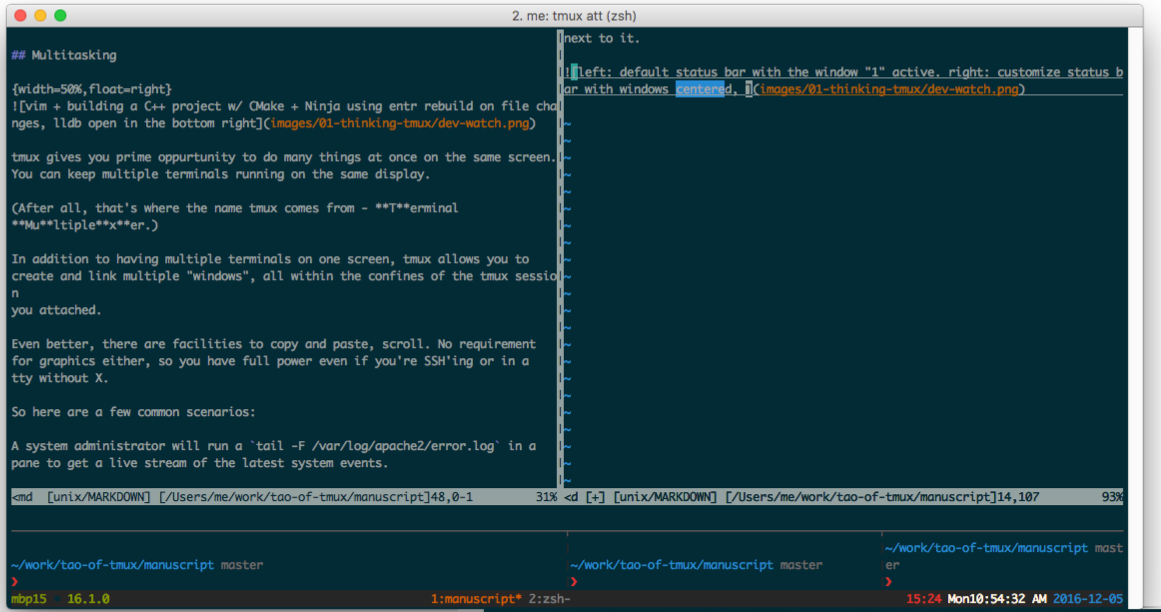
会话（Sessions）{#sessions}

Welcome to the session, the highest-level entity residing in the server instance. Server instances are forked to the background upon starting a fresh instance and reconnected to when reattaching sessions. Your interaction with tmux will have *at least* one session running.

一个会话包含多个窗口 windows。



激活的窗口有一个 * 在标签 tab 上。



2. me: tmux att (zsh)

```
## Multitasking

{width=50%,float=right}
![vim + building a C++ project w/ CMake + Ninja using entr rebuild on file cha
nges, lldb open in the bottom right](images/01-thinking-tmux/dev-watch.png)

tmux gives you prime oppurtunity to do many things at once on the same screen.
You can keep multiple terminals running on the same display.

(After all, that's where the name tmux comes from - **T**erminal
**Mu**ltiple**x**er.)

In addition to having multiple terminals on one screen, tmux allows you to
create and link multiple "windows", all within the confines of the tmux sessio
n
you attached.

Even better, there are facilities to copy and paste, scroll. No requirement
for graphics either, so you have full power even if you're SSH'ing or in a
tty without X.

So here are a few common scenarios:

A system administrator will run a `tail -F /var/log/apache2/error.log` in a
pane to get a live stream of the latest system events.
<md  [unix/MARKDOWN] [/Users/me/work/tao-of-tmux/manuscript]48,0-1       31%
```

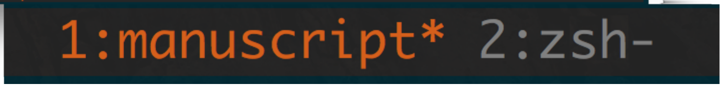
```
next to it.

![left: default status bar with the window "1" active. right: customize status b
ar with windows centered, ](images/01-thinking-tmux/dev-watch.png)
~
<d [+] [unix/MARKDOWN] [/Users/me/work/tao-of-tmux/manuscript]14,107      93%
```

```
~/work/tao-of-tmux/manuscript master
❯
```

```
~/work/tao-of-tmux/manuscript master
❯
```

```
~/work/tao-of-tmux/manuscript mast
er
❯
```

```
mbp15  16.1.0            1:manuscript* 2:zsh-           15:24 Mon10:54:32 AM 2016-12-05
```



```
1:manuscript* 2:zsh-
```

The first window, ID 1, titled ”manuscript” is active. The second window, ID 2, titled zsh.

6.1 创建会话 (sessions)

最简单的创建会话的方法是直接打 `tmux`:

```
$ tmux
```

`$ tmux` 不带参数等价于 `$ tmux new-session` 命令。

默认情况下，session 名字是一个数字，没有什么描述。下面的命令更好一些:

```
$ tmux new-session -s'my rails project'
```

6.2 tmux 切换会话 (sessions)

Some acquire the habit of detaching their tmux client and reattaching via `tmux att -t session_name`. Thankfully, you can switch sessions from within tmux!

`Prefix` + `s` will allow you to switch between sessions within the same tmux client.

This command name can be confusing. `switch-client` will allow you to traverse between sessions in the *server*.

Example usage:

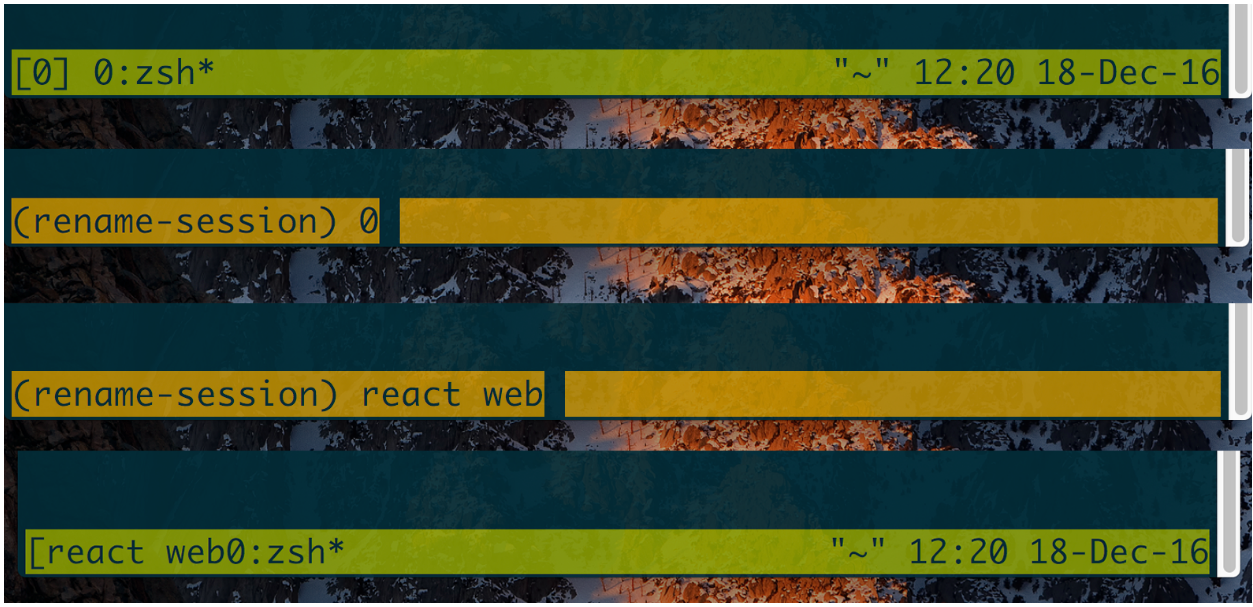
```
$ tmux switch-client -t dev
```

If already inside a client, this will switch to a session, named “dev” , if it exists.

6.3 重命名会话

Sometimes, the default session name given by tmux isn’ t descriptive enough. It only takes a few seconds to update it.

You can name it whatever you want. Typically, if I’ m working on multiple web projects in one session, I’ ll name it “web” . If I’ m assigning one software project to a single session, I’ ll name it after the software project. You’ ll likely develop your own naming conventions, but anything is more descriptive than the default.



```
[0] 0:zsh*                              "~" 12:20 18-Dec-16

(rename-session) 0

(rename-session) react web

[react web0:zsh*                        "~" 12:20 18-Dec-16
```

Renaming a session ’0’ to ’react web’

If you don’ t name your sessions, it’ ll be difficult to keep track of what the session contains. Sometimes, you may forget you have a project opened, especially if your machine has been running for a few days, weeks, or months. You can save time by reattaching your session and avoid creating a duplicate.

You can rename sessions from within tmux with `Prefix` + . The status bar will be temporarily altered into a text field to allow altering the session name.

Through command line, you can try:

```
$ tmux rename-session -t 1 "my session"
```

6.4 查找存在的会话

If you’re scripting tmux, you will want to see if a session exists. `has-session` will return a 0 exit code if the session exists, but will report a 1 exit code *and* print an error if a session does not exist.

```
$ tmux has-session -t 1
```

It assumes the session “1” exists; it’ll just return 0 with no output.

But if it doesn’t, you’ll get something like this in a response:

```
$ tmux has-session -t 1
> can't find session 1
```

To try it in a shell script:

```
if tmux has-session -t 0 ; then
    echo "has session 0"
fi
```

6.5 小节

In this chapter, you learned how to rename sessions for organizational purposes and how to switch between them quickly.

You’ll always be attached to a session when you’re using a client in tmux. When the last remaining session is closed, the server will close also.

Think of sessions as workspaces designed to help organize a set of windows, analogous to virtual desktop spaces in GUI computing.

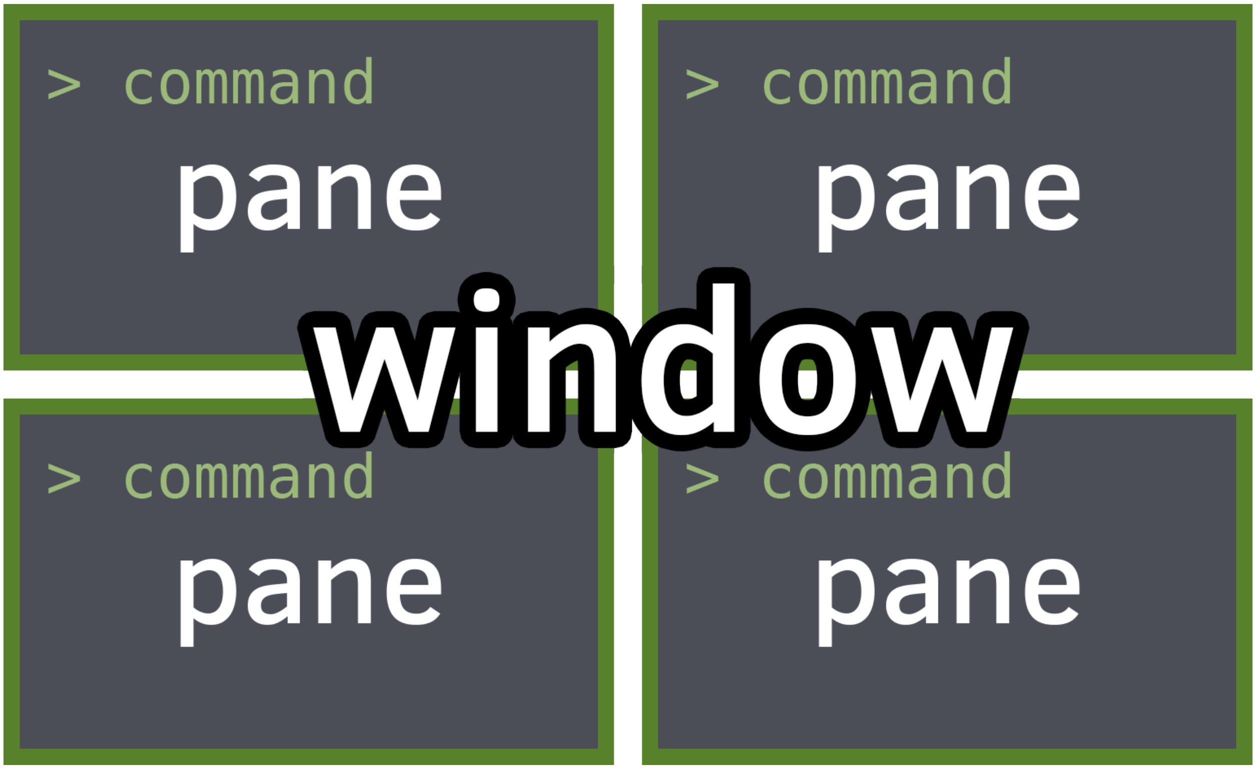
In the next chapter, we will go into *windows*, which, like sessions, are also nameable and let you switch between them.

CHAPTER 7

窗口（Windows）{#windows}

Windows 包含 *panes*. windows 又包含在 *session* 中。

They also have *layouts*, which can be one of many preset dimensions or a custom one done through *pane resizing*.



You can see the current windows through the *status bar* at the bottom of tmux.

7.1 Creating windows

All sessions start with at least one window open. From there, you can create and kill windows as you see fit.

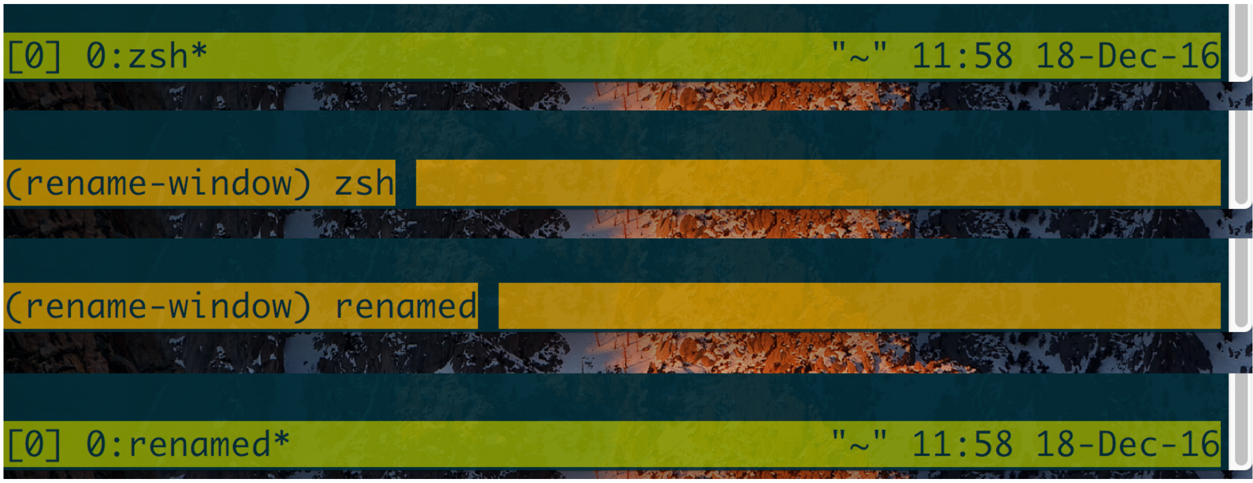
Window indexes are numbers tmux uses to determine ordering. The first window’ s index is 0, unless you set it via `base-index` in your *configuration*. I usually `set -g base-index 1` in my tmux configuration, since 0 is after 9 on the keyboard.

`Prefix` + `c` will create a new window at the first open index. So, if you’ re in the first window, and there is no second window created, it will create the second window. If the second window is already taken, and the third hasn’ t been created, it will create the third window.

If the `base_index` is 1 and there are 7 windows created, with the 5th window missing, creating a new window will fill the empty 5th index, since it’ s the next one in order and nothing is filling it. The next created window would be the eighth.

7.2 Naming windows

Just like with sessions, windows can have names. Labelling them helps keep track of what you’ re doing inside them.



```
[0] 0:zsh*                                  "~" 11:58 18-Dec-16

(rename-window) zsh

(rename-window) renamed

[0] 0:renamed*                              "~" 11:58 18-Dec-16
```

Renaming a window ’zsh’ to ’renamed’

When inside tmux, the shortcut `Prefix` + `,` is most commonly used. It opens a prompt in the tmux status line, where you can alter the name of the current window.

The default numbers given to windows also become muscle memory after a while. But naming helps you when you’ re in a new tmux flow and want to organize yourself. Also, if you’ re sharing tmux with another user, it’ s good practice to give a hint what’ s inside the windows.

7.3 Traversing windows

Moving around windows is done in two ways, first, by iterating through via Prefix + p and Prefix + n and via the window index, which takes you directly to a specific window.

Prefix + 1, Prefix + 2, and so on⋯allows quickly navigating to windows via their index. Unlike window names, which change, indexes are consistent and only require a quick key combo for you to invoke.

Prompt for a window index (useful for indexes greater than 9) with Prefix + '. If the window index is 10 or above, this will help you a lot.

7.3.1 Tip: Search + Traverse Windows for Text

You can forward to a window with a match of a text string by doing Prefix + f.

Bring up the last selected window with Prefix + l.

A list of current windows can be displayed with Prefix + w. This also gives some info on what’ s inside the window. Helpful when juggling a lot of things!

7.4 Moving windows

Windows can also be reordered one by one via move-window and its associated shortcut. This is helpful if a window is worth keeping open but not important or rarely looked at. After you move a window, you can continue to reorder them at any point in time after.

The command $ tmux move-window can be used to move windows.

The accepted arguments are -s (the window you are moving) and -t, where you are moving the window to.

You can also use $ tmux movew for short.

Example: move the current window to number 2:

```
$ tmux movew -t2
```

Example: move window 2 to window 1:

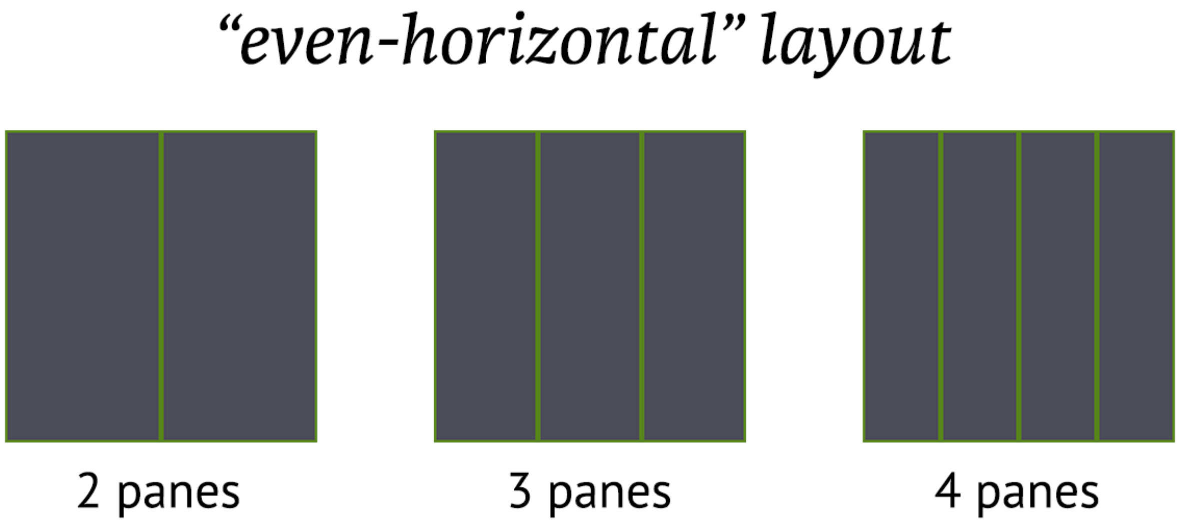
```
$ tmux movew -s2 -t1
```

The shortcut to prompt for an index to move the current window to is Prefix + ..

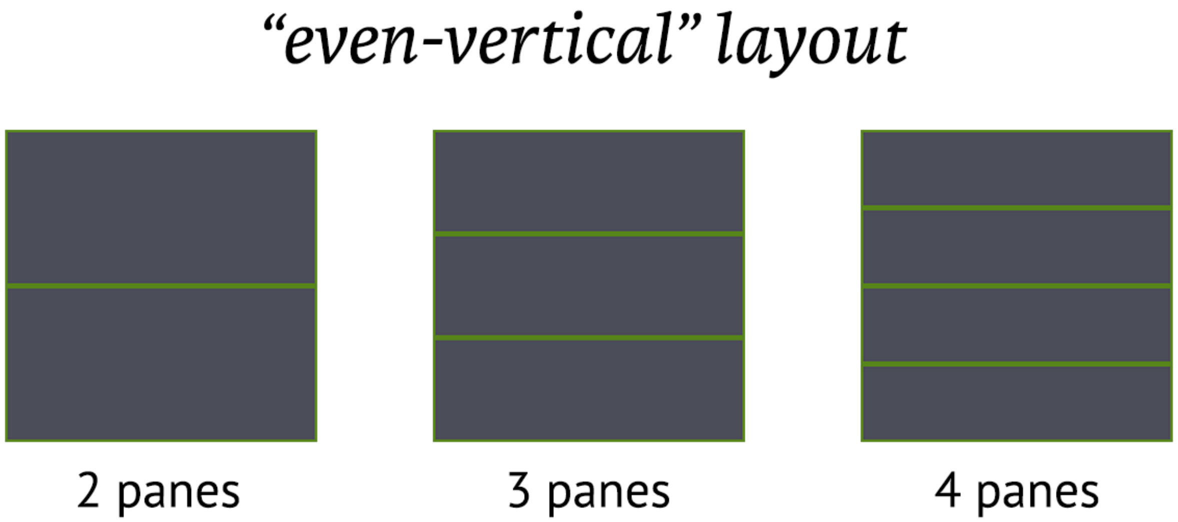
7.5 Layouts {#window-layouts}

`Prefix` + `space` switches window *layouts*. These are preset configurations automatically adjusting proportions of *panes*.

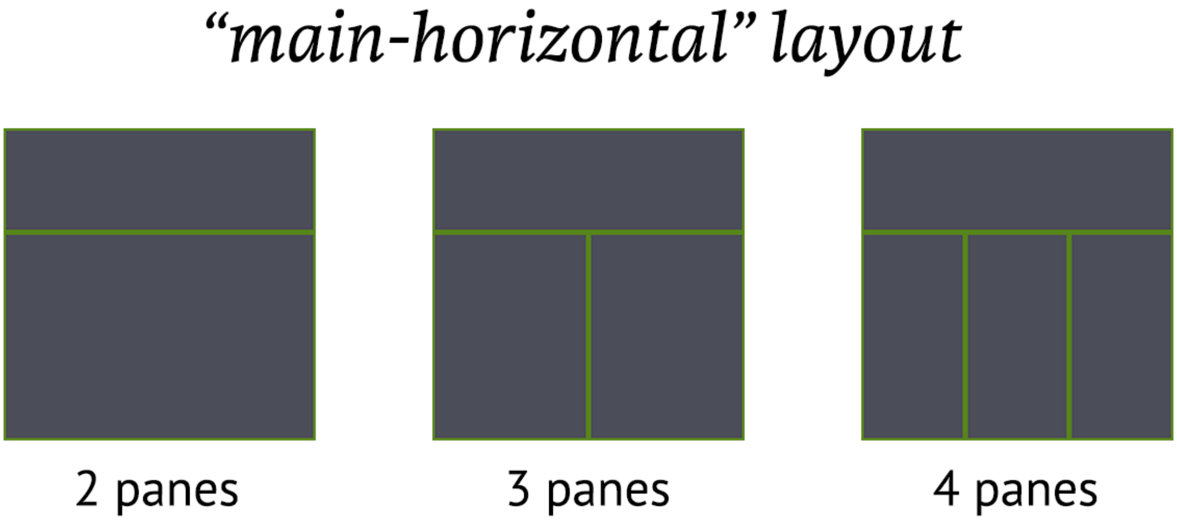
As of tmux 2.3, the supported layouts are:



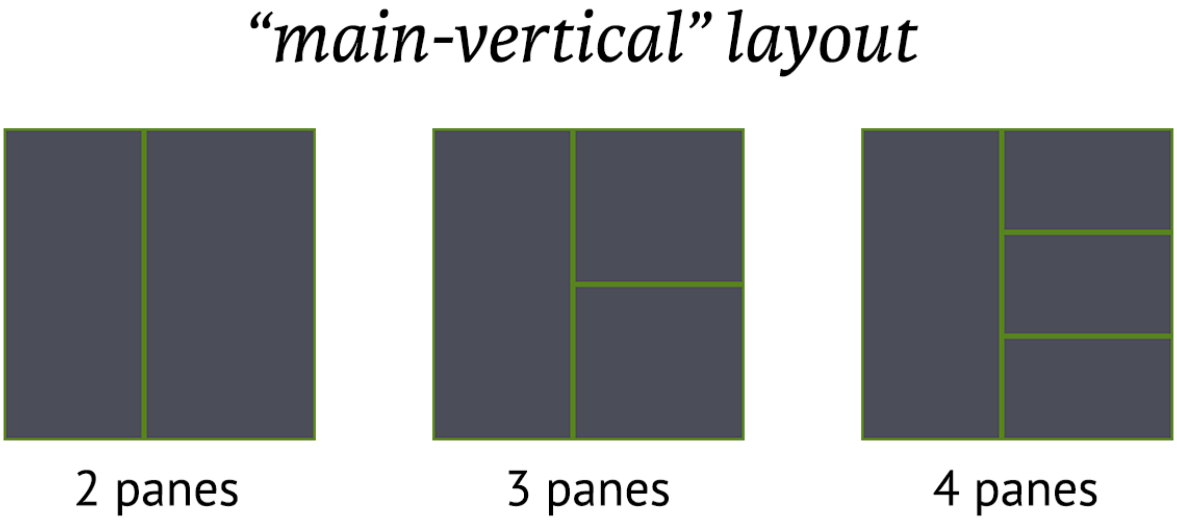
{width=75%}



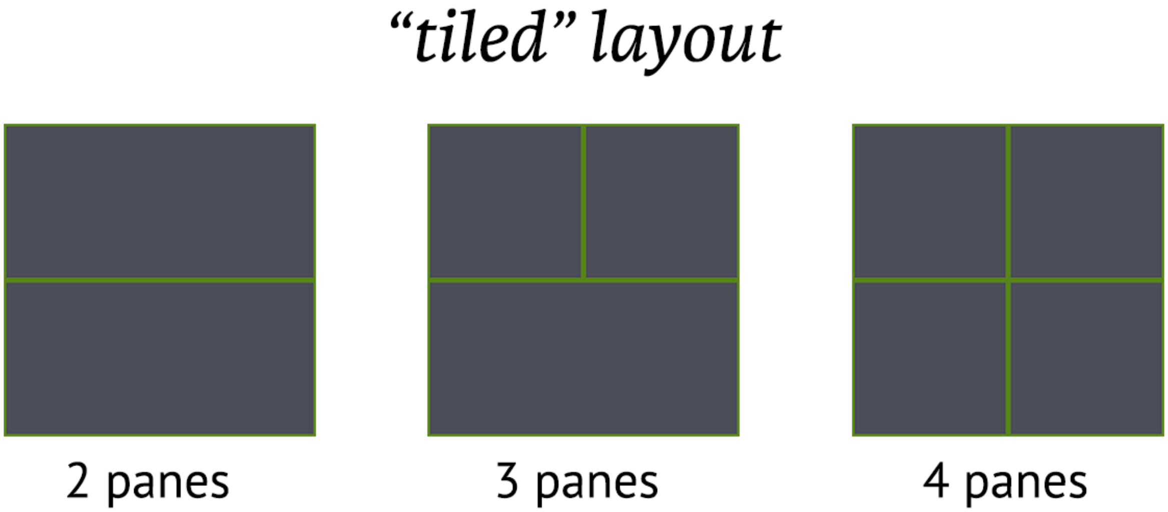
{width=75%}



{width=75%}



{width=75%}



{width=75%}

Specific touch-ups can be done via *resizing panes*.

To reset the proportions of the layout (such as after splitting or resizing panes), you have to run `$ tmux select-layout` again for the layout.

This is different behavior than some tiling window managers. *awesome* and *xmonad*, for instance, automatically handle proportions upon new items being added to their layouts.

To allow easy resetting to a sensible layout across machines and terminal dimensions, you can try this in your *config*:

```
bind m set-window-option main-pane-height 60\; select-layout main-horizontal
```

This allows you to set a `main-horizontal` layout and automatically set the bottom panes proportionally on the bottom every time you do `Prefix` + `m`.

Layouts can also be custom. To get the custom layout snippet for your current window, try this:

```
$ tmux lsw -F "#{window_active} #{window_layout}" | grep "^1" | cut -d " " -f2
```

To apply this layout:

```
$ tmux lsw -F "#{window_active} #{window_layout}" | grep "^1" | cut -d " " -f2
> 5aed,176x79,0,0[176x59,0,0,0,176x19,0,60{87x19,0,60,1,88x19,88,60,2}]

# resize your panes or try doing this in another window to see the outcome
$ tmux select-layout "5aed,176x79,0,0[176x59,0,0,0,176x19,0,60{87x19,0,60,1,88x19,88,60,2}]"
```

7.6 Closing windows

There are two ways to kill a window. First, exit or kill every pane in the window. Panes can be killed via `Prefix` + `x` or by `Ctrl` + `d` within the pane’s shell. The second way, `Prefix` + `&`, prompts if you really want to delete the window. Warning: this will destroy all the window’s panes, along with the processes within them.

From inside the current window, try this:

```
$ tmux kill-window
```

Another thing, when *scripting* or trying to kill the window from outside, use a *target* of the window index:

```
$ tmux kill-window -t2
```

If you’re trying to find the target of the window to kill, they reside in the number in the middle section of the *status line* and via `$ tmux choose-window`. You can hit “return” after you’re in choose-window to go back to where you were previously.

7.7 Summary

In this chapter, you learned how to manipulate windows via renaming and changing their layouts, a couple of ways to kill windows in a pinch or in when shell scripting tmux. In addition, this chapter demonstrated how to save any tmux layout by printing the `window_layout` template variable.

If you are in a tmux session, you' ll always have at least one window open, and you' ll be in it. And within the window will be "pane" ; a shell within a shell. When a window closes all of its panes, the window closes too. In the next chapter, we' ll go deeper into panes.

CHAPTER 8

面板（Panes）{#panes}

Panes are pseudoterminals encapsulating shells (e.g., Bash, Zsh). They reside within a window. A terminal within a terminal, they can run shell commands, scripts, and programs, like vim, emacs, top, htop, irssi, weechat, and so on within them.

8.1 创建新面板

To create a new pane, you can `split-window` from within the current *window* and pane you are in.

You can continue to create panes until you’ ve reached the limit of what the terminal can fit. This depends on the dimensions of your terminal. A normal window will usually have 1 to 5 panes open.

Example usage:

```
# Create pane horizontally, $HOME directory, 50% width of current pane
$ tmux split-window -h -c $HOME -p 50 vim
```

![](images/07-pane/splitw/-h -c $HOME -p 50 vim - 2 panes.png)

```
# create new pane, split vertically with 75% height
tmux split-window -p 75
```

![](images/07-pane/splitw/-p 75.png)

8.2 面板间切换（Traversing Panes）{#pane-traversal}

> *Moving around vimtuitively*
>
> If you like vim (hjkl) keybindings, add these to your *config*:
>
> ```
>     # hjkl pane traversal
>     bind h select-pane -L
>     bind j select-pane -D
>     bind k select-pane -U
>     bind l select-pane -R
> ```

8.3 面板最小化（Zoom in ）{#zoom-pane}

To zoom in on a pane, navigate to it and do `Prefix` + `z`.

You can unzoom by pressing `Prefix` + `z` again.

In addition, you can unzoom and move to an adjacent pane at the same time using a *pane traversal* key.

Behind the scenes, the keybinding is a shortcut for `$ tmux resize-pane -Z`. So, if you ever wanted to script tmux to zoom/unzoom a pane or apply this functionality to a custom key binding, you can do that too, for instance:

```
    bind-key -T prefix y resize-pane -Z
```

This would have `Prefix` + `y` zoom and unzoom panes.

8.4 面板大小（Resizing panes）{#resizing-panes}

Pane size can be adjusted within *windows* via *window layouts* and `resize-pane`. Adjusting window layout switches the proportions and order of the panes. Resizing the panes targets a specific pane inside the window containing it, also shrinking or growing the size of the other columns or rows. It’ s like adjusting your car seat or reclining on a flight; if you take up more space, something else will have less space.

8.5 Outputting pane to a file

You can output the display of a pane to a file.

```
$ tmux pipe-pane -o 'cat >>~/output.#I-#P'
```

The `#I` and `#P` are *formats* for window index and pane index, so the file created is unique. Clever!

8.6 小节

Panes are shells within a shell. You can keep adding panes to a tmux window until you run out of room on your screen. Within your shell, you can `tail -F` log files, write and run scripts, and run curses-powered applications, like vim, top, htop, ncmpcpp, irssi, weechat, mutt, and so on.

You will always have at least one pane open. Once you kill the last pane in the window, the window will close. Panes are also resizable; you can resize panes by targeting them specifically and changing the window layout.

In the next chapter, we will go into the ways you can customize your tmux shortcuts, status line, and behavior.

CHAPTER 9

配置（Configuration ）{#config}

Most tmux users break away from the defaults by creating their own customized configurations. These configurations vary from the trivial, such as adding keybindings, and adjusting the prefix key, to complex things, such as decking out the *status bar* with system stats and fancy glyphs via powerlines.

Configuration of tmux is managed through `.tmux.conf` in your `$HOME` directory. The paths `~/.tmux.conf` and `$HOME/.tmux.conf` should work on OS X, Linux, and BSD.

Configuration is applied upon initially starting tmux. The contents of the configuration are tmux commands. The file can be reloaded later via `source-file`, which is discussed in this chapter.

For a sample config, I maintain a pretty decked out one at https://github.com/tony/tmux-config. It’ s permissively licensed, and you’ re free to copy and paste from it as you wish.

Custom Configs

You can specify your config via the `-f` command. Like this:

```
    $ tmux -f path/to/config.conf
```

Note: If a tmux server is running in the background and you want to test a fresh config, you must either shut down the rest of the tmux sessions or use a different socket name. Like this:

```
$ tmux -f path/to/config.conf -Ltesting_tmux
```

And you can treat everything like normal; just keep passing `-Ltesting_tmux` (or whatever socket name you feel like testing configs with) for reuse.

```
$ tmux -Ltesting_tmux attach
```

9.1 重载配置文件 {#reload-config}

You can apply config files in live tmux sessions. Compare this to `source` or “dot” in the POSIX standard.

`Prefix` + `:` will open the tmux prompt, then type:

`:source /path/to/config.conf`

And hit return.

`$ tmux source-file /path/to/config.conf` can also achieve the same result via command line.

> 简便加载
>
> Even better, often, you will keep your default tmux config stored in `$HOME/.tmux.conf`. So, what can you do? You can `bind-key` to `source-file ~/.tmux.conf`:
>
> `bind r source ~/.tmux.conf`
>
> You can also have it give you a confirmation afterwards:
>
> `bind r source ~/.tmux.conf\; display "~/.tmux.conf sourced!"`
>
> Now, you can type `Prefix` + `r` to get the config to reload.

Note that reloading the configuration only *re-runs* the configuration file. It will not reset keybindings or styling you apply to tmux.

9.2 配置文件的工作原理

The tmux configuration is processed just like run commands in a `~/.zshrc` or `~/.bashrc` file. `bind r source ~/.tmux.conf` in the tmux configuration is the same as `$ tmux bind r source ~/.tmux.conf`.

You could always create a shell script prefixing `tmux` in front of commands and run it on fresh servers. The result is the same. Same goes if you manually type in `$ tmux set-option` and `$ tmux bind-key` commands into any terminal (in or outside tmux).

This in `.tmux.conf`:

```
bind-key a send-prefix
```

Is the same as having no `.tmux.conf` (or `$ tmux -f/dev/null`) and typing:

```
$ tmux bind-key a send-prefix
```

in a newly started tmux server.

The important thing to internalize is that a tmux configuration consists of setting server options (`set-option -s`), global session (`set-option -g`), and window options (`set-window-option -g`).

The rest of this chapter is going to proceed cookbook-style. You can pick out these tweaks and add them to your `.tmux.conf` and *reload*.

9.3 服务设置 (Server options)

Server options are set with `set-option -s option value`.

9.3.1 Tweak timing between key sequences

```
set -s escape-time 0
```

9.3.2 Terminal coloring

If you’ re having an issue with color detail in tmux, it may help to set `default-terminal` to `screen-256color`.

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

This sets the `TERM` variable in new panes.

9.4 会话设置 (Session options)

Aside from the *status bar*, covered in the next chapter, most user configuration will be custom keybindings. This section covers the few generic options, and the next section goes into snippets involving keybindings.

9.4.1 窗口计数 (Base index)

This was mentioned earlier in the book, but it’ s a favorite tweak of many tmux users, who find it more intuitive to start their window counting at *1*, rather than the default, *0*. To set the starting number (base index) for windows:

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

Setting `base-index` assures newly created windows start at 1 and count upwards.

9.5 窗口设置 (Window options)

Window options are set via `set-option -w` or `set-window-option`. They are the same thing.

9.5.1 Automatic window naming

Setting `automatic-rename` alters the name of the window based upon its active pane:

```
set-window-option -g automatic-rename
```

Automatic renaming will be disabled for the window if you rename it manually.

9.6 快捷键绑定 (Keybindings)

9.6.1 Prefix key

The default *prefix key* in tmux is `<Ctrl-b>`. You can customize it by setting a new prefix and unsetting the default. To set the prefix to `<Ctrl-a>`, like GNU Screen, try this:

```
set-option -g prefix C-a
unbind-key C-b
bind-key a send-prefix
```

9.6.2 New window with prompt

Prompt for window name upon creating a new window, `Prefix` + `C` (capital C):

```
bind-key C command-prompt -p "Name of new window: " "new-window -n '%%'"
```

9.6.3 Vi copy-paste keys

This is comprised of two-parts: Setting the `mode-keys` window option to vi and setting the `vi-copy` bindings to use `v` to begin selection and `y` to yank.

```
# Vi copypaste mode
set-window-option -g mode-keys vi
bind-key -t vi-copy 'v' begin-selection
bind-key -t vi-copy 'y' copy-selection
```

9.6.4 hjkl / vi-like pane traversal

Another one for vi fans, this keeps your right hand on the home row when moving directionally across panes in a window.

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

9.6.5 Further inspiration

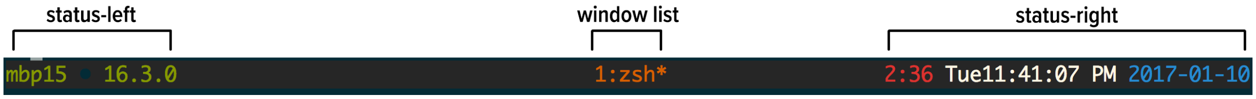
For more ideas, I have a `.tmux.conf` you can copy-paste from on the internet at https://github.com/tony/tmux-config/blob/master/.tmux.conf.

In the next chapter, we will go into configuring the *status line*.

CHAPTER 10

状态栏（Status bar）和个性化设置 {#status-bar}

在 tmux 界面的下部位置是状态栏，可以进行个性化。它由三个部分组成。左右他们都可以自定义。中间部分为一系列窗口的名字。如下图。



```
mbp15   16.3.0                         1:zsh*                    2:36 Tue11:41:07 PM 2017-01-10
```

左右的状态 `status-left` 和 `status-right` 可以参数设置，通过 configurable 文件 `.tmux.conf`，或者想直接命令行修改的方式 `$ tmux set-option`。

显示现在的状态栏设置

```
$ tmux show-options -g | grep status
```

10.1 窗口状态的标记

中间的窗口名字列表部分，在这些窗口名字上由一些标识代表不同的信息。整理成下面的表格：

小技巧：一个面板（pane）可以被最小化 pane zoomed 通过 `Prefix + z`，再按一次 `Prefix + z` 或者按面板的移动组合键，恢复。

10.2 日期和时间

`status-left` 和 `status-right` 部分可以设置日期。

This happens via piping the status templates through format_expand_time in format.c, which routes right into strftime(3) from time.h.

A full list of variables can be found in the documentation for strftime(3). This can be viewed through $ man strftime on Unix-like systems.

10.3 Shell command output

You can also call applications, such as tmux-mem-cpu-load, conky, and *powerline*.

For this example, we’ ll use tmux-mem-cpu-load. This works on Unix-like systems like FreeBSD, Linux distributions, and macOS.

To build from source, you must have CMake and git, which are available through your package manager. You must have a C++ compiler. On macOS, install Xcode CLI Utilities. You can do this by going to *Applications -> Utilities*, launching *Terminal.app* and typing $ xcode-select --install. macOS can use Homebrew to install the CMake and git package. Major Linux distributions package CMake, clang, and git.

Before this step, you can cd into any directory you’ re ok keeping code in.

```
$ git clone https://github.com/thewtex/tmux-mem-cpu-load.git
$ cd tmux-mem-cpu-load
$ mkdir ./build
$ cd ./build
$ cmake ..
$ make

# macOS, no sudo required
$ make install

# Linux, BSD will require sudo / root to install
$ sudo make install
```

If successful, you should see the output below:

```
[100%] Built target tmux-mem-cpu-load
Install the project...
-- Install configuration: "MinSizeRel"
-- Installing: /usr/local/bin/tmux-mem-cpu-load
```

You can remove the source code you cloned from the computer. The compiled application is installed.

You can now add #(tmux-mem-cpu-load) to your status-left or status-right option. In the “*Dressed up*” example below, I use status-left and also theme it to be green:

#[fg=green,bg=default,bright]#(tmux-mem-cpu-load)

So to apply it to your theme, you need to double check what you already have. You may have information on there you want to keep.

```
$ tmux show-option -g status-right
status-right " "#{=21:pane_title}" %H:%M %d-%b-%y"
```

Copy what you had in response (or change, rearrange as you see fit) then add the `#(tmux-mem-cpu-load)` to it. You can apply the new status line in your current tmux session via `$ tmux set-option -g status-right`:

```
$ tmux set-option -g status-right '"#{=21:pane_title}" #(tmux-mem-cpu-load) %H:%M %d-%b-%y'
```

Also, note how I switched out the double quotes on either side of the option with single quotes. This is required, since there are double quotes inside.

You can do this with anything, for instance, try adding `uptime`. This could be done by adding `#(uptime)` to your status line. Typically the output is pretty long, so trim it down by doing something like this:

`#(uptime | cut -f 4-5 -d " " | cut -f 1 -d ",")`

In the next section, we go into how you can style (color) tmux.

10.4 Styling

The *colors* available to tmux are:

- `black`, `red`, `green`, `yellow`, `blue`, `magenta`, `cyan`, `white`.
- bright colors, such as `brightred`, `brightgreen`, `brightyellow`, `brightblue`, `brightmagenta`, `brightcyan`.
- `colour0` through `colour255` from the 256-color set.
- `default`
- hexadecimal RGB code like `#000000`, `#FFFFFF`, similar to HTML colors.

10.4.1 Status line

You can use `[bg=color]` and `[fg=color]` to adjust the text color and background within for status line text. This works on `status-left` and `status-right`.

Let’ s say you want to style the background:

Command: `$ tmux set-option status-style fg=white,bg=black`

In config: `status-style fg=white,bg=black`

In the examples at the end of the chapter, you will see complete examples of how colors can be used.

10.4.2 Clock styling

You can style the color of the tmux clock via:

```
    set-option -g clock-mode-colour white
```

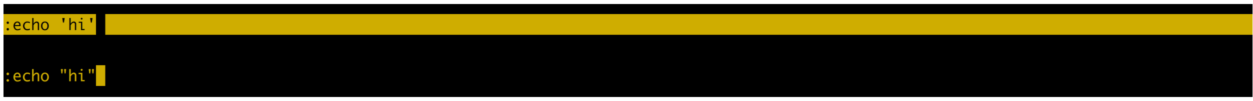
Reminder: Clock mode can be opened with `$ tmux clock-mode` or `Prefix` + `t`. Pressing any key will exit clock mode.

10.4.3 Prompt colors

The benefit of wrapping your brain around this styling is you will see it `message-command-style`, `message style` and so on.

Let' s try this:

```
$ tmux set-option -ag message-style fg=yellow,blink\; set-option -ag message-style bg=black
```



```
:echo 'hi'
:echo "hi"
```

Top: default scheme for prompt. Bottom: newly-styled.

10.5 Styling while using tmux

So, you want to customize your tmux status line before you write the changes to your *config* file.

Start by grabbing your current status line section you want to edit, for instance:

```
$ tmux show-options -g status-left
> status-left "[#S] "
$ tmux show-options -g status-right
> status-right " "#{=21:pane_title}" %H:%M %d-%b-%y"
```

Also, you can try to snip off the variable with `| cut -d' ' -f2-`:

```
$ tmux show-options -g status-left | cut -d' ' -f2-
> "[#S] "
$ tmux show-options -g status-right | cut -d' ' -f2-
> " "#{=21:pane_title}" %H:%M %d-%b-%y"
```

Then, add the options to your *configuration*.

To be sure your configuration fully works, you can start it in a different server via `tmux -Lrandom`, verify the settings, and close it. This is helpful to make sure your config file isn' t missing any styling info.

10.6 Toggling status line

The tmux status line can be hidden, as well. Turn it off:

```
$ tmux set-option status off
```

And, turn it on:

```
$ tmux set-option status on
```

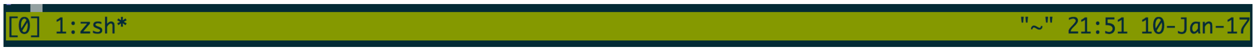
The above is best for scripting, but if you’ re binding it to a keyboard shortcut, *toggling*, or reversing the current option, it can be done via omitting the on/off value:

```
$ tmux set-option status
```

Bind toggling status line to `Prefix` + `q`:

```
$ tmux bind-key q set-option status
```

10.7 Example: Default config



```
[0] 1:zsh*                                                          "~" 21:51 10-Jan-17
```

This is an example of the default config you see if your tmux configuration has no status styling.

```
status on
status-interval 15
status-justify left
status-keys vi
status-left "[#S] "
status-left-length 10
status-left-style default
status-position bottom
status-right " "#{=21:pane_title}" %H:%M %d-%b-%y"
status-right-length 40
status-right-style default
status-style fg=black,bg=green
```

10.8 Example: Dressed up {#status-bar-example-dressed-up}

![](images/09-status-bar/dressed up.png)

```
    status on
    status-interval 1
    status-justify centre
    status-keys vi
    status-left "#[fg=green]#H #[fg=black]• #[fg=green,bright]#(uname -r | cut -c 1-6)#[default]"
    status-left-length 20
    status-left-style default
    status-position bottom
    status-right "#[fg=green,bg=default,bright]#(tmux-mem-cpu-load) #[fg=red,dim,bg=default]
↪#(uptime | cut -f 4-5 -d " " | cut -f 1 -d ",") #[fg=white,bg=default]%a%l:%M:%S %p#[default]
↪#[fg=blue]%Y-%m-%d"
    status-right-length 140
    status-right-style default
    status-style fg=colour136,bg=colour235

    # default window title colors
    set-window-option -g window-status-fg colour244  # base0
    set-window-option -g window-status-bg default

    # active window title colors
    set-window-option -g window-status-current-fg colour166  # orange
    set-window-option -g window-status-current-bg default
```

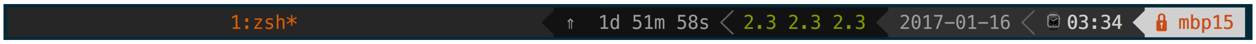
Configs can print the output of an application. In this example, tmux-mem-cpu-load is providing system statistics in the right-side section of the status line.

To build tmux-mem-cpu-load, you have to install CMake and have a C++ compiler, like clang or GCC.

On Ubuntu, Debian, and Mint machines, you can do this via `$ sudo apt-get install cmake build-essential`. On macOS w/ brew via `$ brew install cmake`.

Source: https://github.com/tony/tmux-config

10.9 Example: Powerline



```
1:zsh*                ⇑ 1d 51m 58s < 2.3 2.3 2.3  2017-01-16 < 03:34  mbp15
```

The most full-featured solution available for tmux status lines is powerline, which heavily utilizes the shell command outputs, not only to give direct system statistics, but also to generate graphical-like styling.

To get the styling to work correctly, special fonts must be installed. The easiest way to use this is to install powerline fonts, a collection of fixed width coder fonts patched to support Wingdings-like symbols.

Installation instructions are on Read the Docs. For a better idea:

```
$ pip install --user powerline-status psutil
```

psutil, a required dependency of powerline, is a cross-platform tool to gather system information.

Assure you *properly configured python with your PATHs*, and try this:

```
set -g status-interval 2
set -g status-right '#(powerline tmux right)'
```

10.10 Summary

Configuring the status line is optional. It can use the output of programs installed on your system to give you specialized information, such as CPU, ram, and I/O usage. By default, you' ll at least have a window list and a clock.

In addition, you can customize the colors of the status line, clock, and prompt. By default, it' s only a green bar with dark text, so take some time to customize yours, if you want, and save it to your *configuration*.

In the next chapter, we will go into the command line and scripting features of tmux.

CHAPTER 11

tmux 的脚本化（Scripting）{#scripting-tmux}

tmux 中的命令行快捷方式和选项比较多。

我整理以一些表格，每个人使用 tmux 都有点不同，你也可以整理一个你自己使用的表格。这里我尽量包含大家经常使用的，更多的清单可以看 附录清单（cheatsheets）

11.1 缩写（Aliases）{#aliases}

tmux 命令有很多缩写（alias）。使用缩写，使用 `$ tmux attach` 命令即可以执行 `$ tmux attach-session` 命令。缩写主要是方便记忆，不用将整个命令打出来。

If you know the full name of the command, if you were to chop the hyphen (-) from the command and add the first letter of the last word, you’ d get the shortcut, e.g., **swap-w**indow is swapw, **split-w**indow is splitw.

11.2 Pattern matching {#fnmatch}

In addition to aliases, tmux commands and arguments may all be accessed via `fnmatch(3)` patterns.

For instance, you need not type `$ tmux attach-session` every time. First, there’ s the *alias* of `$ tmux attach`, but additionally, more concise commands can be used if they partially match the name of the command or the target. tmux’ s pattern matching allows `$ tmux attac`, `$ tmux att`, `$ tmux at` and `$ tmux a` to reach `$ tmux attach`.

Every tmux command has shorthands; let’ s try this for `$ tmux new-session`:

```
$ tmux new-session

# ...

$ tmux new-s
```

and so on, until:

```
$ tmux new-
ambiguous command: new-, could be: new-session, new-window
```

The limitation, as seen above, is command matches can collide. Multiple commands begin with `new-`. So, if you wanted to use matches, `$ tmux new-s` for a new session or `$ tmux new-w` for a new window would be the most efficient way. But, the alias of `$ tmux new` for new session and `$ tmux neww` for new windows is even more concise than matching, since the special alias exists.

Patterns can also match *targets* with window and session names. For instance, a session named `mysession` can be matched via `mys`:

```
$ tmux attach -t mys
```

Matching targets will fail if a pattern matches more than one item. If 2 sessions exist, named `mysession` and `mysession2`, the above command would fail. To target either session, the complete target name must be specified.

11.3 Targets {#targets}

If a command allows target specification, it’ s usually done through `-t`.

Think of targets as tmux’ s way of specifying a unique key in a relational database.

What I use to help me remember:

So, sessions are represented by dollar signs ($) because they hold your projects (*ostensibly* where you make money or help someone else do it).

Windows are represented by the at sign (@). So, windows are like referencing / messaging a user on a social networking website.

Panes are the fun one, represented by the percent sign (%), like the default prompt for csh and tcsh. Hey, makes sense, since panes are pseudoterminals!

When scripting tmux, the symbols help denote the type of object, but also serve as a way to target something deeply, such as the pane, *directly*, without needing to know or specify its window or session.

Here are some examples of targets, assuming one session named `mysession` and a client at `/dev/ttys004`:

11.3.1 attach-session [-t target-session]

```
$ tmux attach-session -t mysession
```

11.3.2 detach-client [-s target-session] [-t target-client]

```
$ tmux detach-client -s mysession -t /dev/ttys004

# If within client, -t is assumed to be current client
$ tmux detach-client -s mysession
```

11.3.3 has-session [-t target-session]

```
$ tmux has-session -t mysession

# Pattern matching session name
$ tmux has-session -t mys
```

11.3.4 $ tmux kill-session [-t target-session]

```
$ tmux kill-session -t mysession
```

11.3.5 $ tmux list-clients [-t target-session]

```
$ tmux list-clients -t mysession
```

11.3.6 $ tmux lock-client [-t target-client]

```
$ tmux lock-clients -t /dev/ttys004
```

11.3.7 $ tmux lock-session [-t target-session]

```
$ tmux lock-session -t mysession
```

11.3.8 $ tmux new-session [-t target-session]

```
$ tmux new-session -t newsession

# Create new-session in the background
$ tmux new-session -t newsession -d
```

11.3.9 $ tmux refresh-client [-t target-client]

```
$ tmux refresh-client -t /dev/ttys004
```

11.3.10 $ tmux rename-session [-t target-session] session-name

```
$ tmux rename-session -t mysession renamedsession

# If within attached session, -t is assumed
$ tmux rename-session renamedsession
```

11.3.11 $ tmux show-messages [-t target-client]

```
$ tmux show-messages -t /dev/ttys004
```

11.3.12 $ tmux suspend-client [-t target-client]

```
$ tmux suspend-client -t /dev/ttys004

# If already in client
$ tmux suspend-client

# Bring client back to the foreground
$ fg
```

11.3.13 $ tmux switch-client [-c target-client] [-t target-session]

```
$ tmux suspend-client -c /dev/ttys004 -t othersession
```

(continues on next page)

(续上页)

```
    # Within current client, -c is assumed
    $ tmux suspend-client -t othersession
```

11.4 Formats {#formats}

tmux provides a minimal template language and set of variables to access information about your tmux environment.

Formats are specified via the `-F` flag.

You know how template engines, such as mustache, handlebars ERB in ruby, jinja2 in python, twig in PHP, and JSP in Java, allow template variables? Formats are a similar concept.

The `FORMATS` (variables) provided by tmux have expanded greatly since version 1.8. Some of the most commonly used formats as of tmux 2.3 are listed below. See the *appendix section on formats* for a complete list.

Let’s try to output it:

```
$ tmux list-windows -F "#{window_id} #{window_name}"
> @0 zsh
```

Here’s a cool trick to list all panes with the x and y coordinates of the cursor position:

```
    $ tmux list-panes -F "#{pane_id} #{pane_current_command} \
      #{pane_current_path} #{cursor_x},#{cursor_y}"
    > %0 vim /Users/me/work/tao-of-tmux/manuscript 0,34
      %1 tmux /Users/me/work/tao-of-tmux/manuscript 0,17
      %2 man /Users/me/work/tao-of-tmux/manuscript 0,0
```

Variables are specific to the objects being listed. For instance:

Server-wide variables: `host`, `host_short` (no domain name), `socket_path`, `start_time` and `pid`.

Session-wide variables: `session_attached`, `session_activity`, `session_created`, `session_height`, `session_id`, `session_name`, `session_width`, `session_windows` and all server-wide variables.

Window variables: `window_activity`, `window_active`, `window_height`, `window_id`, `window_index`, `window_layout`, `window_name`, `window_panes`, `window_width` and all session and server variables.

Pane variables: `cursor_x`, `cursor_y`, `pane_active`, `pane_current_command`, `pane_current_path`, `pane_height`, `pane_id`, `pane_index`, `pane_width`, `pane_pid` and all window, session and server variables.

This book focuses on separating the concept of server, sessions, windows, and panes. With the knowledge of targets and formats, this separation takes shape in tmux’s internal attributes. If you `list-panes` all variables up the ladder, including window, session and server variables are available for the panes being listed. Try this:

```
$ tmux list-panes -F "pane: #{pane_id}, window: #{window_id}, \
  session: #{session_id}, server: #{socket_path}"
> pane: %35, window: @13, session: $6, server: /private/tmp/tmux-501/default
  pane: %38, window: @13, session: $6, server: /private/tmp/tmux-501/default
  pane: %36, window: @13, session: $6, server: /private/tmp/tmux-501/default
```

Listing windows isn’t designed to display variables for pane-specific properties. Since a window is a collection of panes, it can have 1 or more panes open at any time.

```
$ tmux list-windows -F "window: #{window_id}, panes: #{window_panes} \
  pane_id: #{pane_id}"
> window: @15, panes: 1 pane_id: %40
  window: @13, panes: 3 pane_id: %36
  window: @25, panes: 1 pane_id: %50
```

This will show the window ID, prefixed by an `@` symbol, and the number of panes inside the window.

Surprisingly, `pane_id` shows up via `list-windows`, as of tmux 2.3. While this output occurs in this version of tmux, it’ s undefined behavior. It’ s advised to keep use of `-F` scoped to the objects being listing when scripting to avoid breakage. For instance, if you want the active pane, use `#{pane_active}` via `$ tmux list-panes -F "#{pane_active}"`.

By default, `list-panes` will only show panes in a window, unless you specify `-a` to output all on a server or `-s [-t session-name]` for all panes in a session:

```
$ tmux list-panes -s -t mysession
> 1.0: [176x29] [history 87/2000, 21033 bytes] %0
  1.1: [87x6] [history 1814/2000, 408479 bytes] %1 (active)
  1.2: [88x6] [history 1916/2000, 464932 bytes] %2
  2.0: [176x24] [history 9/2000, 2262 bytes] %13
  2.1: [55x11] [history 55/2000, 7395 bytes] %14
```

And the `-t` flag lists all panes in a window:

```
$ tmux list-panes -t @0
> 0: [176x29] [history 87/2000, 21033 bytes] %0
  1: [176x36] [history 1790/2000, 407807 bytes] %1 (active)
  2: [88x6] [history 1916/2000, 464932 bytes] %2
```

The same concept applies to `list-windows`. By default, The `-a` flag will list all windows on a server, `-t` lists windows within a session, and omitting `-t` will only list windows within the current session inside tmux.

```
    $ tmux list-windows
    > 1: zsh* (3 panes) [176x36] [layout f9a4,176x36,0,0[176x29,0,0,0,176x6,0,30{87x6,0,30,1,88x6,
↪88,30,2}]] @0 (active)
      2: zsh- (5 panes) [176x36] [layout 55ef,176x36,0,0[176x24,0,0,13,176x11,0,25{55x11,0,25,14,
↪58x11,56,25[58x7,56,25,16,58x3,56,33,17],61x11,115,25,15}]] @6
```

11.5 Controlling tmux {#send-keys}

tmux allows sending keys, including Ctrl via `C-` or `^`, alt (Meta) via `M-`, and special key names. Here’s a list of special keys straight from the manual:

`Up`, `Down`, `Left`, `Right`, `BSpace`, `BTab`, `DC` (Delete), `End`, `Enter`, `Escape`, `F1` to `F12`, `Home`, `IC` (Insert), `NPage/PageDown/PgDn`, `PPage/PageUp/PgUp`, `Space`, and `Tab`.

If special keys are not matched, the defined behavior is to send it as a string to the pane, character by character.

For this example, we will use `send-keys` through tmux prompt, because omitting target (`-t`) will direct the command to the current pane, but the keys sent will sometimes print before the prompt.

Open tmux command prompt via `Prefix` + `:` and type this after the ::

`send-keys echo 'hi'`

Hit enter. This inserted *hi* into the current active pane. You can also use targets to specify which pane to send it to.

Let’s now try to send keys to another pane in our current window. Create a second pane via splitting the window if one doesn’t exist. You can also do this exercise outside of tmux or inside a scripting file and running it.

Grab a pane ID from the output of `list-panes`:

```
    $ tmux list-panes
    > 0: [180x57] [history 87/2000, 21033 bytes] %0
      1: [89x14] [history 1884/2000, 509864 bytes] %1 (active)
      2: [90x14] [history 1853/2000, 465297 bytes] %2
```

%2 looks good. Replace %2 with the pane you want to target. This sends `cal` to the input:

```
    $ tmux send-keys -t %2 'cal'
```

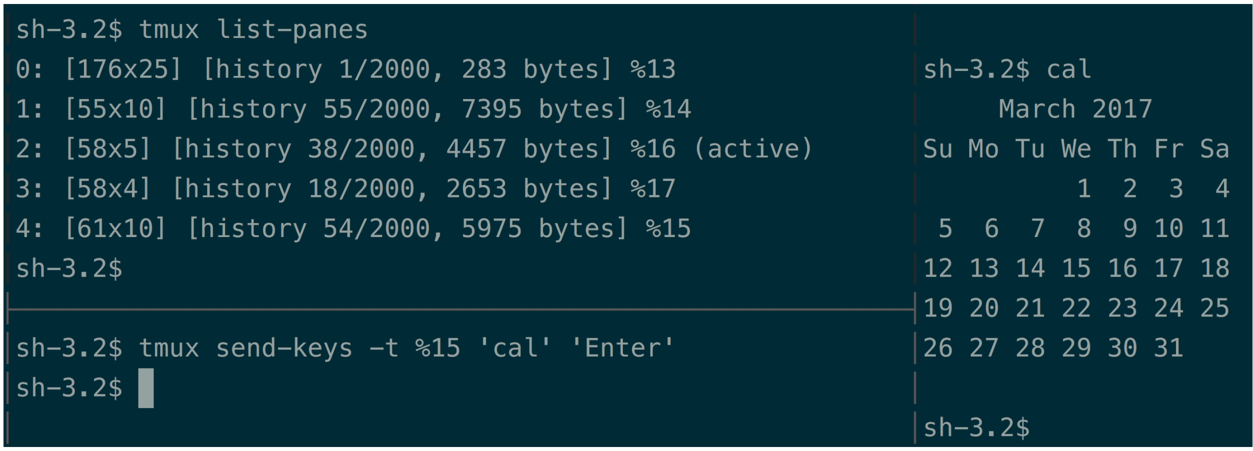
Nice, let’s cancel that out by sending a `SIGINT`:

```
    $ tmux send-keys -t %2 'C-c'
```

This cancelled the command and brought up a fresh input. This time, let’ s send an Enter keypress to run `cal(1)`.

```
$ tmux send-keys -t %2 'cal' 'Enter'
```

This outputs in the adjacent pane.



```
sh-3.2$ tmux list-panes                                     │
0: [176x25] [history 1/2000, 283 bytes] %13                 │sh-3.2$ cal
1: [55x10] [history 55/2000, 7395 bytes] %14                │     March 2017
2: [58x5] [history 38/2000, 4457 bytes] %16 (active)        │Su Mo Tu We Th Fr Sa
3: [58x4] [history 18/2000, 2653 bytes] %17                 │          1  2  3  4
4: [61x10] [history 54/2000, 5975 bytes] %15                │ 5  6  7  8  9 10 11
sh-3.2$                                                     │12 13 14 15 16 17 18
────────────────────────────────────────────────────────────┤19 20 21 22 23 24 25
sh-3.2$ tmux send-keys -t %15 'cal' 'Enter'                 │26 27 28 29 30 31
sh-3.2$ █                                                   │
                                                            │sh-3.2$
```

Top-left: Listing panes, Bottom-left: Sending keys to right pane, Right: Output of cal(1).

11.6 Capturing pane content {#capture-pane}

`$ tmux capture-pane` will copy a panes’ contents.

By default, the contents will be saved to tmux’ s internal clipboard, the *paste buffer*. You can run `capture-pane` within any pane, then navigate to an editor, paste the contents (don’ t forget to `:set paste` and go into insert mode with `i` in vim), and save it to a file. To *paste*, use `Prefix` + `]` inside the pane you’ re pasting into.

You can also add the `-p` flag to print it to stdout. From there, you could use redirection to place the output into a file. Let’ s do `>>` so we don’ t accidentally truncate a file:

```
$ tmux capture-pane -p >> ./test
```

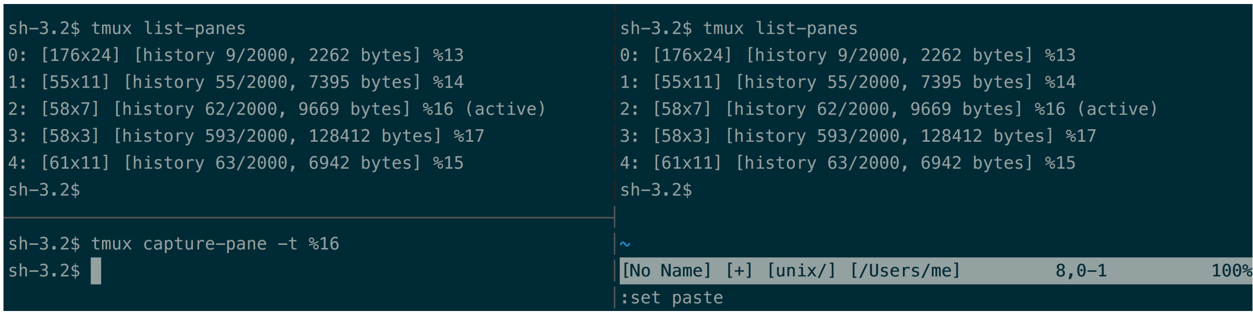
As an alternative to redirection, you can also use `save-buffer`. The `-a` flag will get you the same behavior as appended output direction.

```
$ tmux save-buffer -a ./test
```

To check what’ s inside:

```
$ cat ./test
```

Like with `send-keys`, *targets* can be specified with `-t`. Let’ s copy a pane into tmux’ s clipboard (“paste buffer”) and paste it into a text editor in a third pane:



```
sh-3.2$ tmux list-panes
0: [176x24] [history 9/2000, 2262 bytes] %13
1: [55x11] [history 55/2000, 7395 bytes] %14
2: [58x7] [history 62/2000, 9669 bytes] %16 (active)
3: [58x3] [history 593/2000, 128412 bytes] %17
4: [61x11] [history 63/2000, 6942 bytes] %15
sh-3.2$
```

```
sh-3.2$ tmux capture-pane -t %16
sh-3.2$ 
```

```
sh-3.2$ tmux list-panes
0: [176x24] [history 9/2000, 2262 bytes] %13
1: [55x11] [history 55/2000, 7395 bytes] %14
2: [58x7] [history 62/2000, 9669 bytes] %16 (active)
3: [58x3] [history 593/2000, 128412 bytes] %17
4: [61x11] [history 63/2000, 6942 bytes] %15
sh-3.2$

~
[No Name] [+] [unix/] [/Users/me]         8,0-1          100%
:set paste
```

Top-left: Listing panes, Bottom-left: Capturing pane output of top-left pane, Right: Pasting buffer into vim.

Remember, you can also copy, paste, and send-keys to other windows and sessions also. Targets are server-wide.

11.7 Summary

tmux has a well-devised and intuitive command system, enabling the user to access bread and butter functionality quickly. At the same time, tmux provides a powerful way of retrieving information on its objects between `list-panes`, `list-windows` and `list-sessions` and formats. This makes tmux not only accessible and configurable, but also scriptable.

The ability to retrieve explicitly and reliably, from a session down to a pane. All it takes is a pane’ s ID to capture its contents or even send it keys. Used by the skilled programmer, scripting tmux can facilitate orchestrating terminals in ways previously deemed unrealistic; anything from niche shell scripts to monitor and react to behavior on systems to high-level, intelligent and structured control via object oriented libraries, like libtmux.

In the next chapter, we delve into optimizations that showcase the latest generation of unix tools that build upon old, time-tested concepts, like man pages and piping, while maintaining portability across differences in platforms and graceful degradation to ensure development tooling works on machines missing optional tools. Also, the chapter will introduce *session managers*, a powerful, high-level tool leveraging tmux’ s scripting capabilities to consistently load workspace via a declarative configuration.

CHAPTER 12

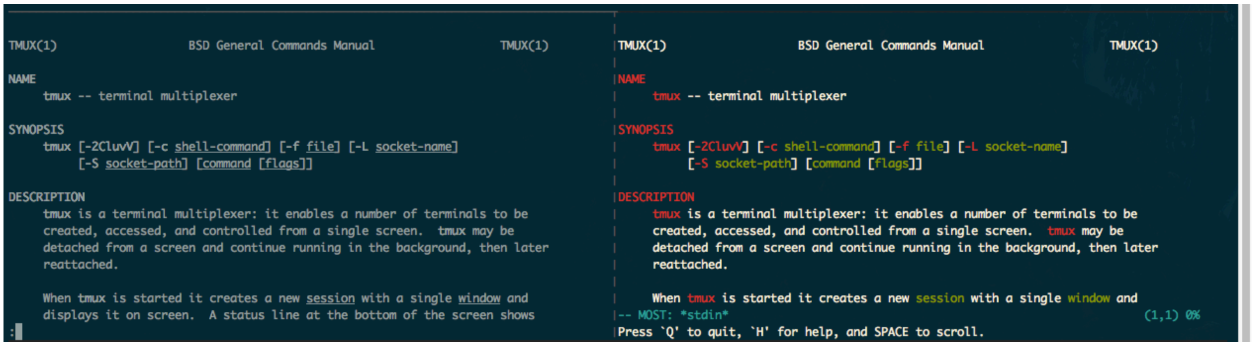
小技巧（tips and tricks）{#tips-and-tricks}

12.1 Read the tmux manual in style

`$ man tmux` is the command to load up the man page for tmux. You can do the same to find instructions for any command or entity with a manpage entry; here are some fun ones:

```
$ man less
$ man man
$ man strftime
```

most(1), a solid `PAGER`, drastically improves readability of manual pages by acting as a syntax highlighter.



```
TMUX(1)                 BSD General Commands Manual                TMUX(1)

NAME
     tmux -- terminal multiplexer

SYNOPSIS
     tmux [-2CluvV] [-c shell-command] [-f file] [-L socket-name]
          [-S socket-path] [command [flags]]

DESCRIPTION
     tmux is a terminal multiplexer: it enables a number of terminals to be
     created, accessed, and controlled from a single screen.  tmux may be
     detached from a screen and continue running in the background, then later
     reattached.

     When tmux is started it creates a new session with a single window and
     displays it on screen.  A status line at the bottom of the screen shows
:
```

```
TMUX(1)                 BSD General Commands Manual                TMUX(1)

NAME
     tmux -- terminal multiplexer

SYNOPSIS
     tmux [-2CluvV] [-c shell-command] [-f file] [-L socket-name]
          [-S socket-path] [command [flags]]

DESCRIPTION
     tmux is a terminal multiplexer: it enables a number of terminals to be
     created, accessed, and controlled from a single screen.  tmux may be
     detached from a screen and continue running in the background, then later
     reattached.

     When tmux is started it creates a new session with a single window and
-- MOST: *stdin*                                                  (1,1) 0%
Press `Q' to quit, `H' for help, and SPACE to scroll.
```

left: man, version 1.6c on macOS Sierra. right: MOST v5.0.0

To get this working, you need to set your `PAGER` environmental variable to point to the MOST binary. You can test it like this:

```
$ PAGER=most man ls
```

If you found you like `most`, you’ ll probably want to make it your default manpage reader. You can do this by setting an environmental variable in your “rc” (Run Commands) for your shell. The location of the file depends on your shell. You can use `$ echo $SHELL` to find it on most shells). In Bash and zsh, these are kept in `~/.bashrc` or `~/.zshrc`, respectively:

```
export PAGER="most"
```

I often reuse my configurations across machines, and some of them may not have `most` installed, so I will have my scripting only set `PAGER` if `most` is found:

```
#!/bin/sh

if command -v most > /dev/null 2>&1; then
    export PAGER="most"
fi
```

Save this in a file, for example, to `~/.dot-config/most.sh`.

Then you can `source` it in via your main rc file.

```
source $HOME/.dot-config/most.sh
```

Patterns like these help make your dot-configs portable, cross-platform, and modular. For inspiration, you can fork, copy, and paste from my permissively- licensed config at https://github.com/tony/.dot-config.

12.2 Log tailing

Not tmux specific, but powerful when used in tandem with it, you can run a follow (`-f`) using `tail(1)`. More modern versions of tail have the `-F` (capitalized), which checks for file renames and rotation.

On OS X, you can do:

```
$ tail -F /var/log/system.log
```

and keep it running in a pane while log messages come in. It’ s like Facebook newsfeed for your system, except for programmers and system administrators.

For monitoring logs, multitail provides a terminal-friendly solution. It’ d be an *Inception* moment, because you’ d be using a log multiplexer in a terminal multiplexer.

12.3 File watching {#file-watching}

In my never-ending conquest to get software projects working in symphony with code changes, I’ ve come to test many file watching applications and patterns. Pursuing the holy grail feedback loop upon file changes, I’ ve gradually become the internet’ s unofficial connoisseur on them.

File watcher applications wait for a file to be updated, then execute a custom command, such as restarting a server, rebuilding an application, running tests, linters, and so on. It gives you, as a developer, instant feedback in the terminal, empowering a tmux workspace to have IDE-like features, without the bloat, memory, and CPU fans roaring.

I eventually settled on `entr(1)`, which works superbly across Linux distros, BSDs and OS X / macOS.

The trick to make entr work is to pipe a list of files into it to watch.

Let’ s search for all `.go` files in a directory and run tests on file change:

```
$ ls -d *.go | entr -c go test ./...
```

Sometimes, we may want to watch files recursively, but we need it to run reliably across systems. We can’ t depend on `**` existing to grab files recursively, since it’ s not portable. Something more POSIX-friendly would be `find . -print | grep -i '.*[.]go'`:

```
$ find . -print | grep -i '.*[.]go' | entr -c go test ./...
```

To only run file watcher if entr is installed, let’ s wrap in a conditional `command -v` test:

```
$ if command -v entr > /dev/null; then find . -print | grep -i '.*[.]go' | \
  entr -c go test ./...; fi
```

And have it fallback to `go test` in the event `entr` isn’ t installed. This allows your command to degrade gracefully. You’ ll thank me when you use this snippet in conjunction with a *session manager*:

```
$ if command -v entr > /dev/null; then find . -print | grep -i '.*[.]go' | \
  entr -c go test ./...; else go test ./...; fi
```

If the project is a team or open source project, where a user never used the command before and could be missing a required software package, we can give a helpful message. This shows a notice to the user to install entr if not installed on the system:

```
$ if command -v entr > /dev/null; then find . -print | grep -i '.*[.]go' | \
  entr -c go test ./...; else go test ./...; echo "\nInstall entr(1) to \"
  echo "run tasks when files change. \nSee http://entrproject.org/"; fi
```

Here’ s why you want patterns like above: You can put it into a `Makefile` and commit it to your project’ s VCS, so you and other developers can have access to this reusable command across different UNIX-like systems, with and without certain programs installed.

Note: You may have to convert the indentation within the Makefiles from spaces to tabs.

Let' s see what a Makefile with this looks like:

```
    watch_test:
        if command -v entr > /dev/null; then find . -print | grep -i '.*[.]go' | entr -c go test ./
↪...; else go test ./...; echo "\nInstall entr(1) to run tasks when files change. \nSee http://
↪entrproject.org/"; fi
```

To run this, do `$ make watch_test` in the same directory as the Makefile.

But it' s still a tad bloated and hard to read. We have a couple tricks at our disposal. One would be to add continuation to the next line with a trailing backslash (\):

```
    watch_test:
        if command -v entr > /dev/null; then find . -print | \
        grep -i '.*[.]go' | entr -c go test ./...; \
        else go test ./...; \
        echo "\nInstall entr(1) to run tasks on file change. \n"; \
        echo "See http://entrproject.org/"; fi
```

Another would be to break the command into variables and make subcommands. So:

```
    WATCH_FILES= find . -type f -not -path '*/\.*' | \
    grep -i '.*[.]go$$' 2> /dev/null

    test:
            go test $(test) ./...

    entr_warn:
            @echo "-------------------------------------------------"
            @echo " ! File watching functionality non-operational ! "
            @echo "                                                 "
            @echo " Install entr(1) to run tasks on file change.    "
            @echo " See http://entrproject.org/                     "
            @echo "-------------------------------------------------"

    watch_test:
            if command -v entr > /dev/null; then ${WATCH_FILES} | \
            entr -c $(MAKE) test; else $(MAKE) test entr_warn; fi
```

`$(MAKE)` is used for portability. One reason is recursive calls, such as here. On BSD systems, you may try invoking `make` via `gmake` (to call GNU Make specifically). This happened to me, while building PDFs for the book AlgoXY. I had to write a patch to make it properly use `$(MAKE)` for recursive calls.

The `$(test)` after `go test` allows passing a shell variable with arguments in it. So, you could do `make watch_test test='-i'`. For examples of a similar Makefile in action, see the one in my tmuxp project.

The project is licensed BSD (permissive), so you can grab code and use it in compliance with the LICENSE.

One more thing, let’ s say you’ re running a server, like Gin, Iris, or Echo. `entr -c` likely won’ t be restarting the server for you. Try entering the `-r` flag to send a `SIGTERM` to the process before restarting it. Combining the current `-c` flag with the new `-r` will give you `entr -rc`:

```
run:
        go run main.go

watch_run:
        if command -v entr > /dev/null; then ${WATCH_FILES} | \
        entr -c $(MAKE) run; else $(MAKE) run entr_warn; fi
```

12.4 Session Managers {#session-manager}

For those who use tmux regularly to perform repetitive tasks, such as opening the same software project, viewing the same logs, etc., frequent tasks will often lead to the creation of tmux scripts.

A user can use plain shell scripting to build their tmux sessions. However, scripting is error prone, hard to debug, and requires tmux to split windows into panes in a certain order. In addition, there’ s the burden of assuring the shell scripts are portable.

A declarative configuration in YAML or JSON configuration abstracts out the commands, layout, and options of tmux. It prevents the mistakes and repetition scripting entails. These applications are called tmux *session managers*, and in different ways, they programmatically create tmux workspaces by running a series of commands based on a config.

Teamocil and Tmuxinator are the first ones I tried. By far, the most popular one is tmuxinator. They are both programmed in Ruby. There’ s also tmuxomatic, where you can “draw” your tmux sessions in text and have tmuxomatic build the layout.

I sort of have a home team advantage here, as I’ m author of tmuxp. Already having used teamocil and tmuxinator, I wrote my own in python instead of ruby, with many more features. For one, it builds on top of libtmux, a library which abstracts tmux *server*, *sessions*, *windows* and *panes* to build the state of tmux sessions. In addition, it has a naive form of session freezing, support for JSON, more flexible configuration options, and it will even offer to attach exiting sessions, instead of redundantly running script commands against the session if it’ s already running.

So, in tmuxp, we’ ll hollow out a tmuxp config directory with `$ mkdir ~/.tmuxp` then create a YAML file at `~/.tmuxp/test.yaml`:

```
session_name: 4-pane-split
windows:
- window_name: dev window
  layout: tiled
```

(continues on next page)

(续上页)

```
    shell_command_before:
      - cd ~/                    # run as a first command in all panes
    panes:
      - shell_command:           # pane no. 1
          - cd /var/log          # run multiple commands in this pane
          - ls -al | grep \.log
      - echo second pane         # pane no. 2
      - echo third pane          # pane no. 3
      - echo forth pane          # pane no. 4
```

gives a session titled *4-pane-split*, with one window titled *dev window* with 4 panes in it. 3 in the home directory; the other is in `/var/log` and is printing a list of all files ending with `.log`.

To launch it, install tmuxp and load the configuration:

```
$ pip install --user tmuxp
$ tmuxp -V   # verify tmuxp is installed, if not you need to fix your `PATH`
             # to point to your python bin folder. More help below.
$ tmuxp load ~/.tmuxp/test.yaml
```

If tmuxp isn' t found, there is a *troubleshooting entry on fixing your paths* in the appendix.

12.5 More code and examples {#example-projects}

I' ve dusted off a C++ space shooter and a new go webapp I' ve been playing with. They' re licensed under MIT so, you can use them, copy and paste from them, etc:

- C++14 space shooter minigame - side scrolling shmup demo (sdl2, cmake, json resource manifests, Linux/BSD/OS X compatible)
- Go tmux web frontend - display current tmux session and window information via browser (gin, bower)

Both support `tmuxp load .` within the project directory to load up the project.

Make sure to install `entr(1)` beforehand!

12.6 tmux-plugins and tpm

tmux-plugins and tmux package manager are a suite of tools dedicated to enhancing the experience of tmux users.

- tmux-resurrect: Persists tmux environment across system restarts.
- tmux-continuum: Continuous saving of tmux environment. Automatic restore when tmux is started. Automatic tmux start when computer is turned on.

- tmux-yank: Tmux plugin for copying to system clipboard. Works on OSX, Linux and Cygwin.
- tmux-battery: Plug and play battery percentage and icon indicator for Tmux.

CHAPTER 13

总结或最后（Takeaway）{#takeaway}

在这本书中，我们采用了一种有组织的方法来理解 tmux。随着越来越多地使用 tmux，请继续返回并使用此资源帮助来记忆。不必一次就理解 tmux 的复杂性，更不用说终端了。适应是随着时间的推移会发生的。

tmux’ s userbase varies in skill level. Some readers of this book may have just learned how to use the `Prefix` key yesterday. Others are looking to tweak their configurations and host it in their “dot files” on github. There also exists a very clever hacker who utilizes the advanced scripting capabilities tmux offers to pilot the terminal in ways previously thought impossible.

We’ ve covered the *server*, *session*, *window*, and *pane* concepts. Panes are shells, AKA pseudoterminals or PTYs. The command system. That *configuration* is basically a file filled with commands. An overview of the *target* system lets you specify objects to interact with tmux commands. A breeze through *formats*, a template system with variables to retrieve information on tmux’ s current state. How to *send keystrokes* and *copy from tmux panes* programmatically. A lot of *terminal tricks* that work across platforms and well with tmux, including a *file watching workflow* to run linting, testing, and build commands on file changes. *Two permissively licensed open source projects* for demonstration. A tmux configuration you can copy and paste from. An object oriented tmux API wrapper and a tmux session manager.

If you liked this book, please leave a review on Amazon and Goodreads. I would also appreciate you leaving something in my tip jar. I am an independent software developer and could use all the help I can get.

If you found an error or have a suggestion, please contact me at `tao.of.tmux@git-pull.com`. I want this book to be the best it can be. If you are having technical difficulties with Kindle, please send me your receipt and I will comp you a leanpub coupon.

{backmatter}

CHAPTER 14

附录：清单（cheatsheets）{#appendix-cheatsheets}

下面直接从 tmux 的使用参考（manual pages）拷贝过来，整理后做成了表格。

14.1 命令（Commands）

14.1.1 会话（Session）

14.1.2 窗口（Window）

14.1.3 面板（Pane）

14.2 （快捷键）Keybindings

14.2.1 混杂快捷键（Miscellaneous）

14.2.2 复制/粘贴（Copy/Paste）相关

14.2.3 会话（Session）相关

会话遍历（Session Traversal）

14.2.4 窗口（Window）相关

窗口遍历（Window Traversal）

窗口移动（Window Moving）

14.2.5 面板（Pane）相关

面板遍历（Pane Traversal）

面板移动（Pane Moving）

面板大小调整（Pane Resizing）

14.3 Formats {#appendix-formats}

14.3.1 Copy / paste

14.3.2 Clients

14.3.3 Panes

14.3.4 Sessions

14.3.5 Windows

14.3.6 Servers

14.3.7 Commands

For `$ tmux list-commands`.

CHAPTER 15

附录：安装 tmux （installation）{#appendix-installation}

15.1 macOS / OS X

15.1.1 brew

```
$ brew install tmux
```

15.1.2 macports

```
$ sudo port install tmux
```

15.1.3 fink

```
$ fink install tmux
```

15.2 Linux

15.2.1 Ubuntu / Mint / Debian, etc.

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

15.2.2 CentOS / Fedora / Redhat, etc.

```
$ sudo yum install tmux
```

15.2.3 Arch Linux (pacman)

```
$ sudo pacman -S tmux
```

15.2.4 Gentoo (portage)

```
$ sudo emerge --ask app-misc/tmux
```

15.3 BSD

15.3.1 FreeBSD

pkg(1)

```
# pkg install tmux
```

pkg_add(1)

```
# pkg_add -r tmux
```

15.3.2 OpenBSD

在 OpenBSD 4.6, 包含了 tmux .

使用以前的系统版本的话:

```
# pkg_add tmux
```

15.3.3 NetBSD

```
$ make -C /usr/pkgsrc/misc/tmux install
```

15.4 Windows 10

查看 附录中 tmux 在 Windows 10 使用小节.

CHAPTER 16

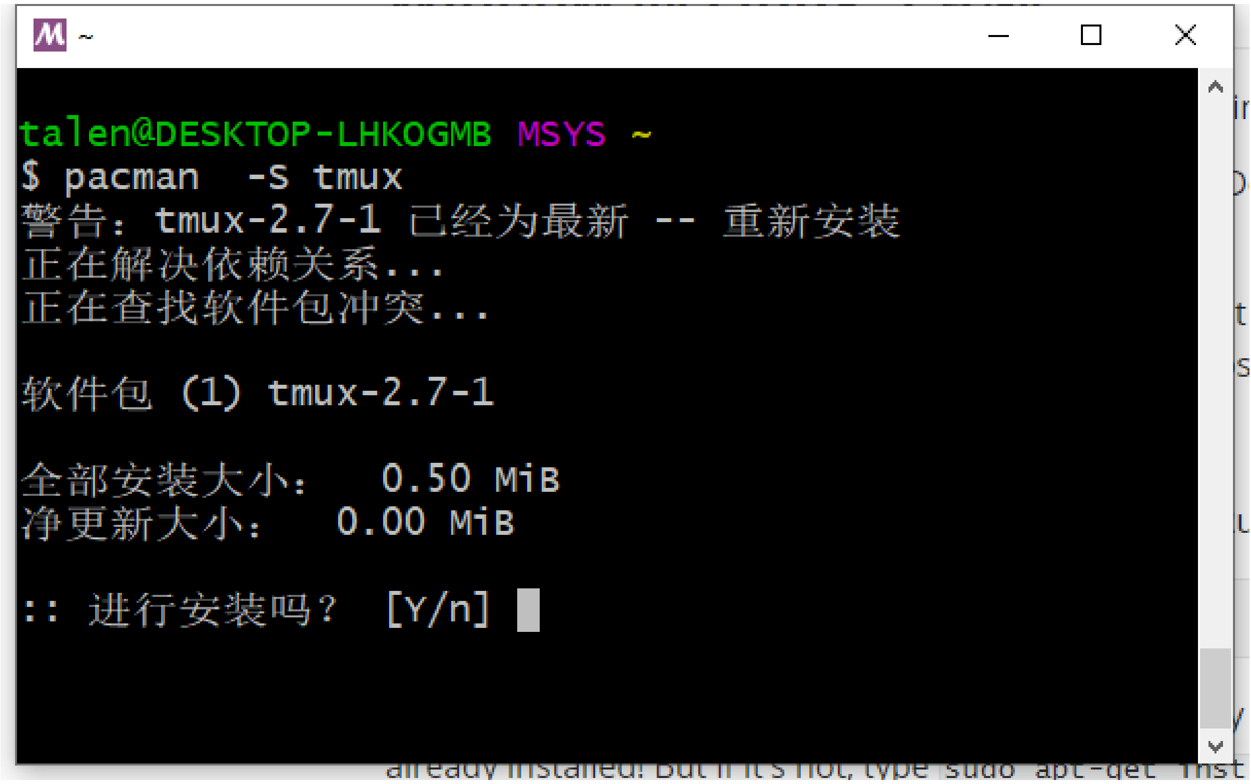
附录：tmux 在 Windows 10 使用 {#appendix-windows-bash}

要在 windows 系统上安装 tmux，可以使用 MSYS2，或者安装 windows 的子系统 Linux。

16.1 通过 MSYS2

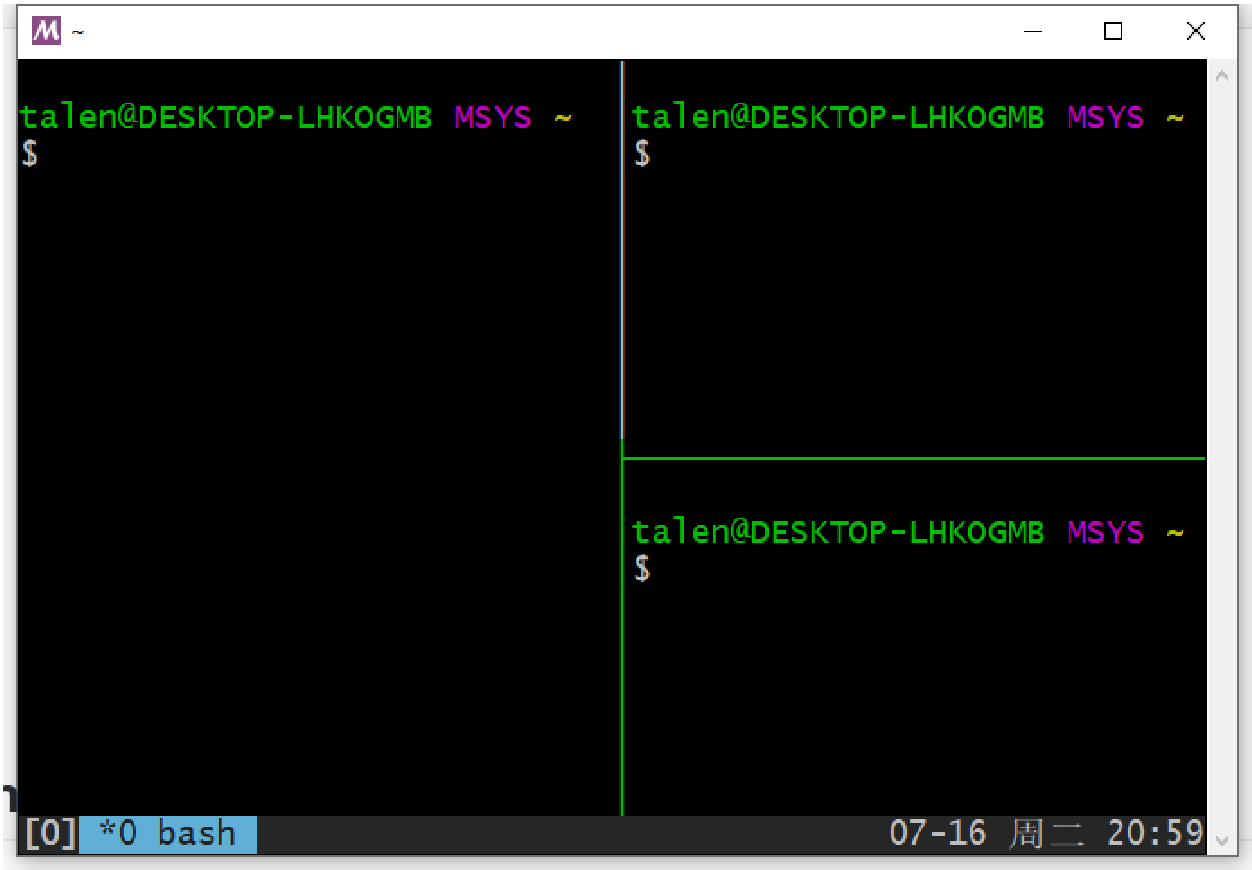
下载安装 MSYS2

```
pacman  -S tmux
```



```
talen@DESKTOP-LHKOGMB MSYS ~
$ pacman  -S tmux
警告：tmux-2.7-1 已经为最新 -- 重新安装
正在解决依赖关系...
正在查找软件包冲突...

软件包 (1) tmux-2.7-1

全部安装大小：   0.50 MiB
净更新大小：   0.00 MiB

:: 进行安装吗？ [Y/n]
```

1563281005928



```
talen@DESKTOP-LHKOGMB MSYS ~        talen@DESKTOP-LHKOGMB MSYS ~
$                                   $

                                    talen@DESKTOP-LHKOGMB MSYS ~
                                    $

[0] *0 bash                                      07-16 周二 20:59
```

1563281982394

16.2 Window 的 Linux 子系统

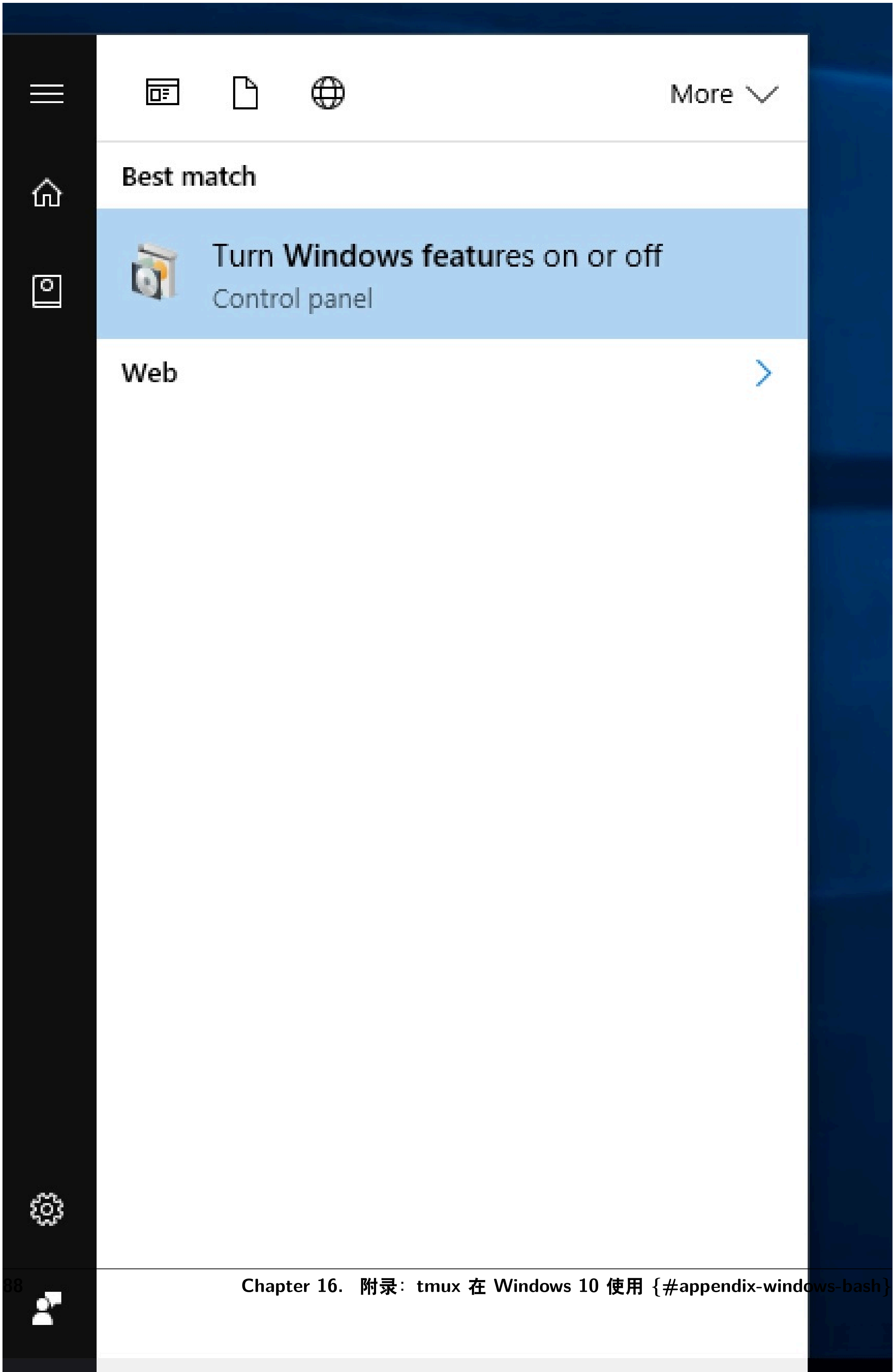
从 Windows 10 build 14361 开始，可以通过 Window 的 Linux 子系统 运行 tmux 。

在设置的 “Update & security” 下的 “For Developers”，使能 **Developer mode** 选项。之后，打开 “Windows Features”。你可以在开始搜索 “Turn Windows features on or off”，然后打开 “Windows Subsystem for Linux (Beta)” 功能，需要重启电脑。

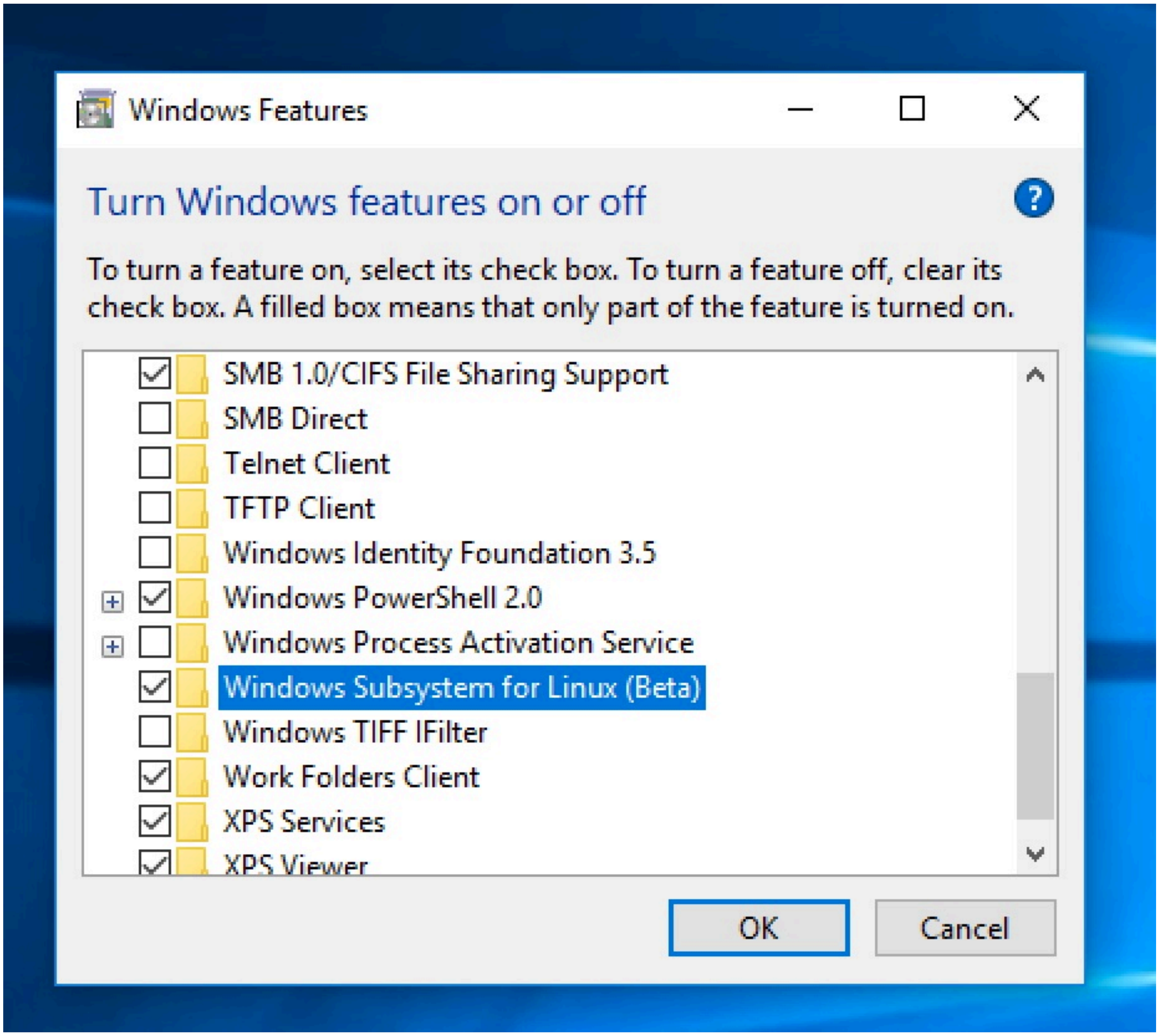
接下来打开 cmd 窗口 (Run cli.exe)，运行：

```
C:\Users\tony> bash.exe
```

可能需要同意一些条款，创建一个用户。在作者的电脑上，tmux 已经安装好了。如果没有安装可以使用 `sudo apt-get install tmux` 安装。


More
Best match
Turn Windows features on or off
Control panel
Web

Turn Windows Features on or off



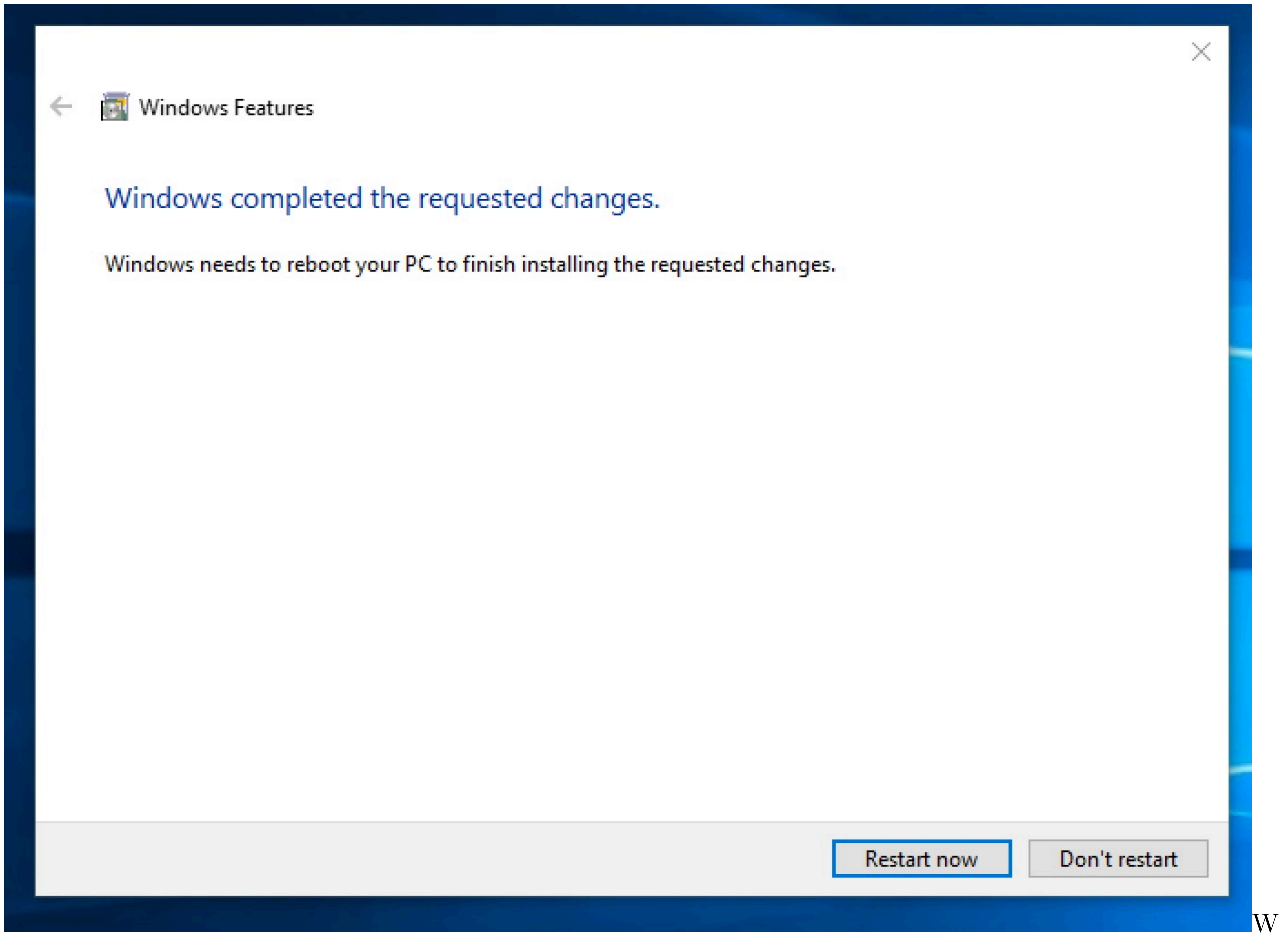
Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☑ SMB 1.0/CIFS File Sharing Support
- □ SMB Direct
- □ Telnet Client
- □ TFTP Client
- □ Windows Identity Foundation 3.5
- ⊞ ☑ Windows PowerShell 2.0
- ⊞ □ Windows Process Activation Service
- ☑ Windows Subsystem for Linux (Beta)
- □ Windows TIFF IFilter
- ☑ Work Folders Client
- ☑ XPS Services
- ☑ XPS Viewer

Check Windows Subsystem for Linux (Beta)

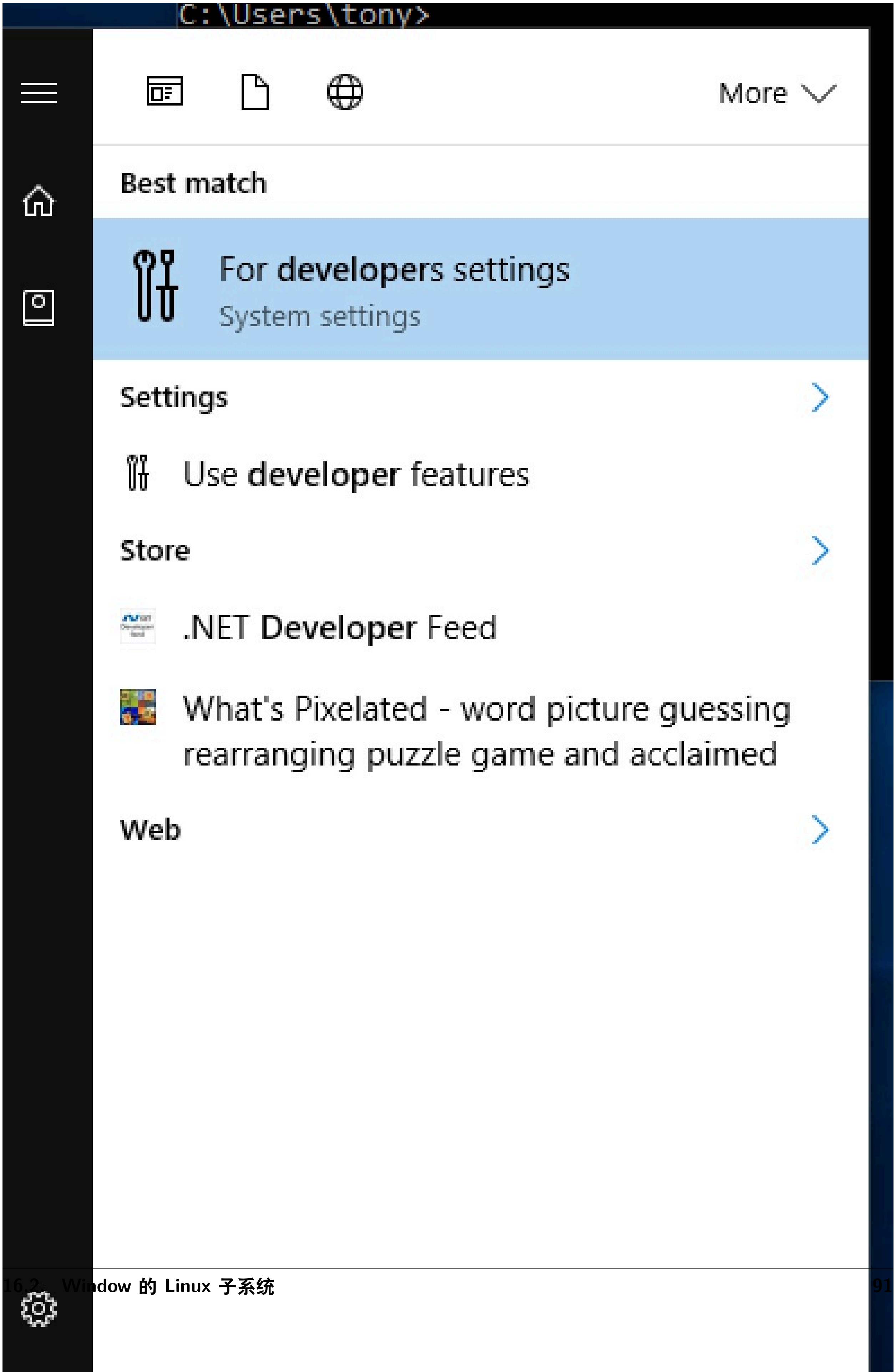


Windows Features

Windows completed the requested changes.

Windows needs to reboot your PC to finish installing the requested changes.

Windows completed the requested changes. Restart



```
C:\Users\tony>
```

More

Best match

For developers settings
System settings

Settings

Use developer features

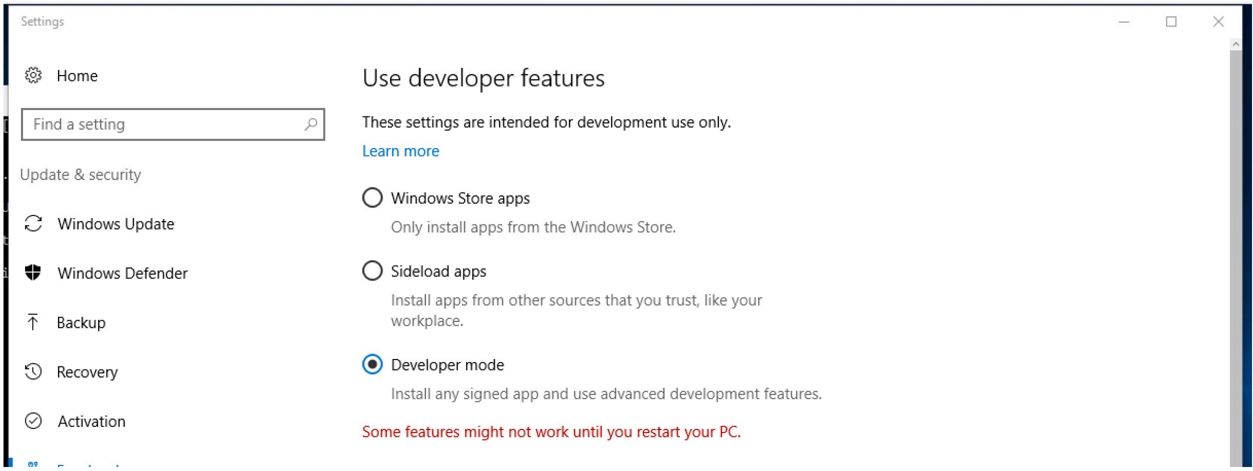
Store

.NET Developer Feed

What's Pixelated - word picture guessing rearranging puzzle game and acclaimed

Web

Developer features



Use developer features

These settings are intended for development use only.
Learn more

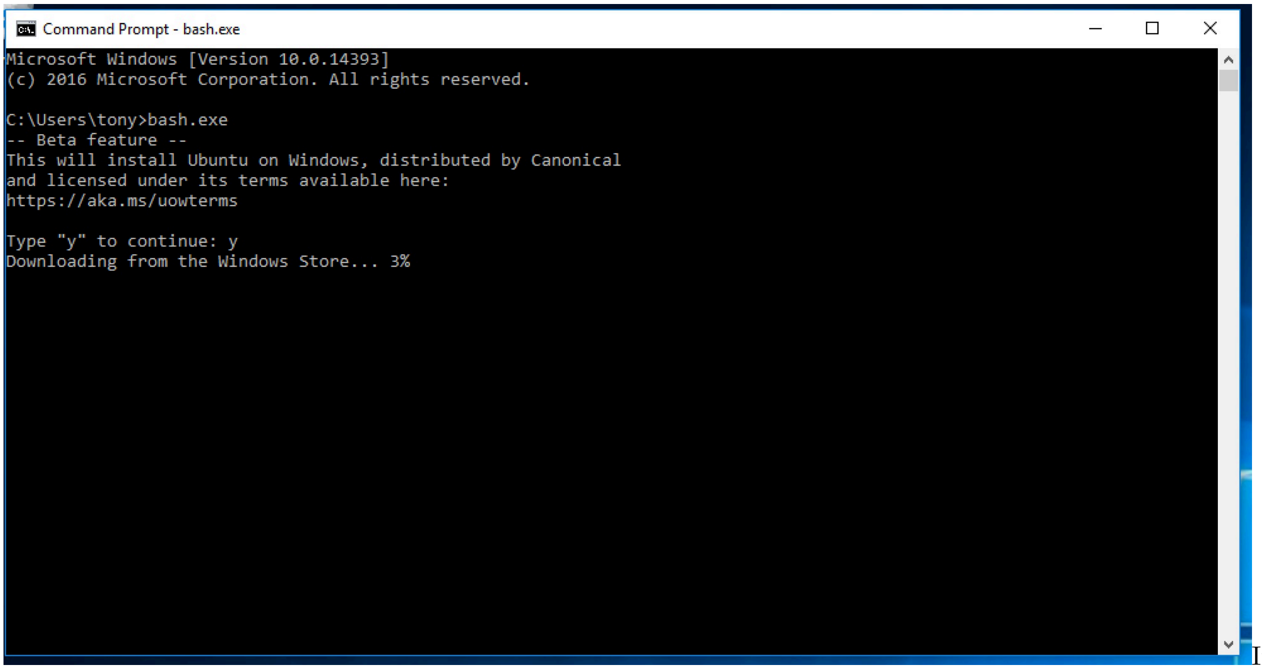
○ Windows Store apps
Only install apps from the Windows Store.

○ Sideload apps
Install apps from other sources that you trust, like your workplace.

● Developer mode
Install any signed app and use advanced development features.

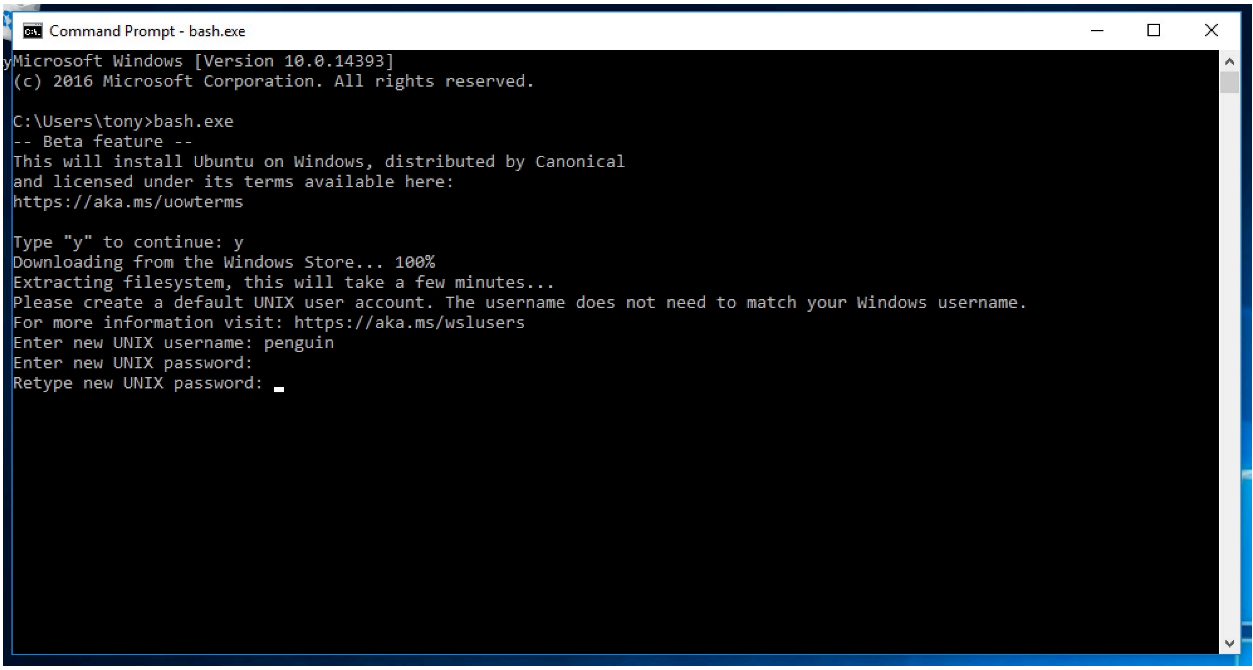
Some features might not work until you restart your PC.

Select Developer mode in Update & Security



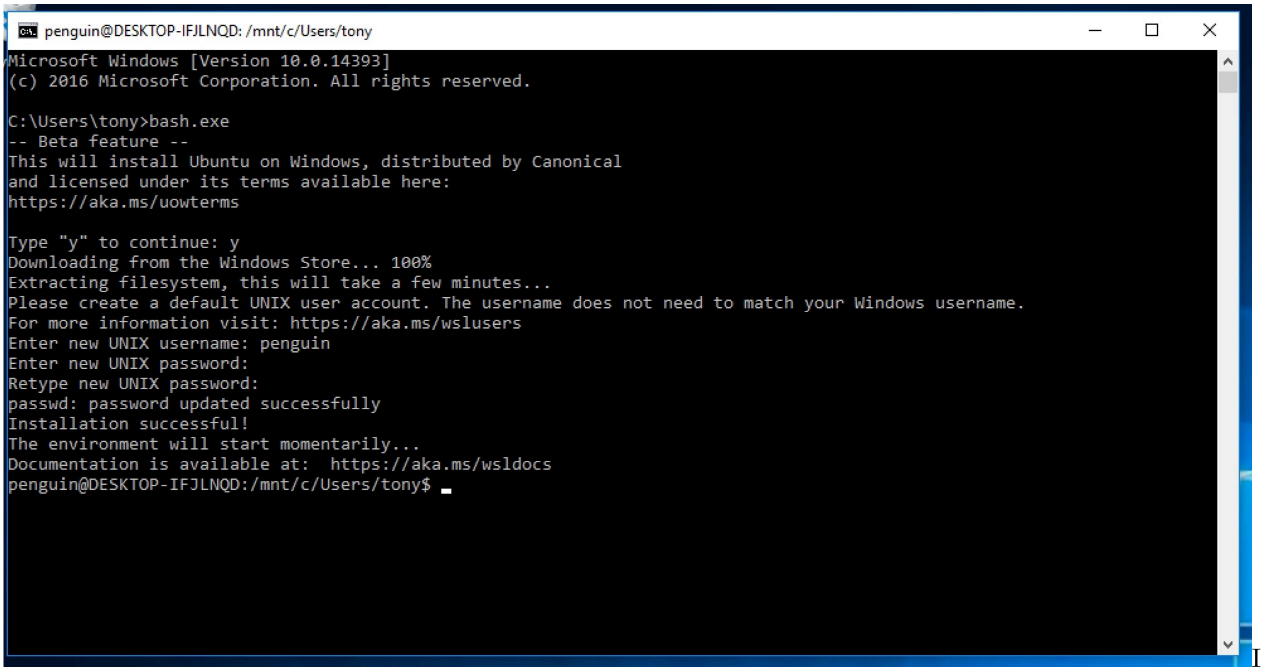
```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\tony>bash.exe
-- Beta feature --
This will install Ubuntu on Windows, distributed by Canonical
and licensed under its terms available here:
https://aka.ms/uowterms

Type "y" to continue: y
Downloading from the Windows Store... 3%
```

Installing Ubuntu from Windows Store



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\tony>bash.exe
-- Beta feature --
This will install Ubuntu on Windows, distributed by Canonical
and licensed under its terms available here:
https://aka.ms/uowterms

Type "y" to continue: y
Downloading from the Windows Store... 100%
Extracting filesystem, this will take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: penguin
Enter new UNIX password:
Retype new UNIX password:
```

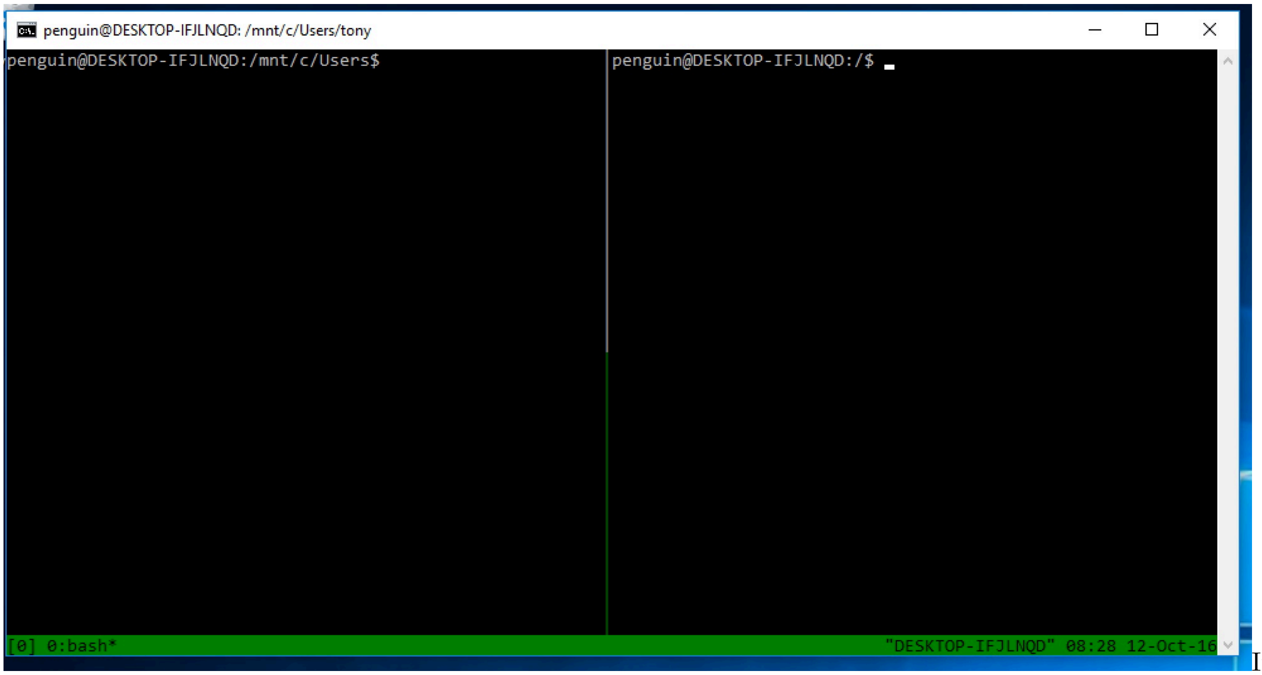
Create Linux user



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\tony>bash.exe
-- Beta feature --
This will install Ubuntu on Windows, distributed by Canonical
and licensed under its terms available here:
https://aka.ms/uowterms

Type "y" to continue: y
Downloading from the Windows Store... 100%
Extracting filesystem, this will take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: penguin
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Installation successful!
The environment will start momentarily...
Documentation is available at:  https://aka.ms/wsldocs
penguin@DESKTOP-IFJLNQD:/mnt/c/Users/tony$
```

In bash!

```
yourusername@COMPUTERNAME-ID321FJ:/mnt/c/Users/username$ tmux
```



penguin@DESKTOP-IFJLNQD: /mnt/c/Users/tony

```
penguin@DESKTOP-IFJLNQD:/mnt/c/Users$          penguin@DESKTOP-IFJLNQD:/$
[0] 0:bash*                                    "DESKTOP-IFJLNQD" 08:28 12-Oct-16
```

In tmux!

这下你可以在 bash.exe 里面运行 tmux 了。

在这个 ubuntu 系统里面，你可以通过 sudo apt-get install **packagename** 继续安装其他软件和 sudo apt-get update && sudo apt-get upgrade 更新软件。

CHAPTER 17

附录：常见问题（troubleshooting）{#appendix-troubleshooting}

17.1 问题：在 vim 中粘贴时出现 E353: Nothing in register * 错误

macOS / OS X ，在 tmux 中使用 vim，尝试粘贴时，出现 bug，可以尝试通过 brew 安装 reattach-to-user-namespace 解决问题。

```
$ brew install reattach-to-user-namespace
```

17.2 问题：tmuxp: command not found 和 powerline: command not found {#troubleshoot-site-paths}

这个问题时你安装的 python 包并没有在你的 python 环境的 site package 里面。首先，找到你的 user site packages base directory：

```
$ python -m site --user-base
```

在 macOS 系统上，这会返回 /Users/me/Library/Python/2.7 ，或者在 Linux/BSD 上会返回/home/me/.local 。

这些应用的包在 bin/，把他们加到你的 PATH里面。一般在 ~/.bashrc 和 ~/.zshrc 文件上配置。

```
export PATH=/Users/me/Library/Python/2.7/bin:$PATH     # macOS w/ python 2.7
export PATH=$HOME/.local/bin:$PATH                     # Linux/BSD
export PATH="`python -m site --user-base`/bin":$PATH   # May work all-around
```

接着开一个新的 terminal，或者使能 `. ~/.zshrc` / `. ~/.bashrc` 在当前的 terminal。现在你可以运行 `$ tmuxp -V`, `$ tmuxp load` 和 `$ powerline tmux right` 等命令了。

CHAPTER 18

Indices and tables

- search