The Tao of tmux

and Terminal Tricks

tony narlock

The Tao of tmux

and Terminal Tricks

Tony Narlock

This book is for sale at http://leanpub.com/the-tao-of-tmux

This version was published on 2017-01-23

This book is dedicated to those who use my software. There's no better feeling of satisfaction or meaning than someone else using what I created.

I'd also like to thank those in open source who sacrifice their time and often go unnoticed; To my international friends abroad who exchanged language and culture with me. To those who deal with naysayers, keep your head in the books and ABC (Always Be Coding), as Linus Torvalds' said, "Talk is Cheap. Show me the code."

To anyone else who has supported me and stuck with me along the way, you have my gratitude.

Table of Contents

Foreword
- About this book
- Styles
- How this book is structured
- Donations
- Available Formats
- Errata
- Thanks
- Book Updates and tmux changes

1. Thinking in tmux
- Window manager for the terminal
- Multitasking
- Keep your applications running in the background
- Powerful combos

2. Terminal fundamentals
- POSIX stuff
- Terminal interface
- Terminal emulators
- Shell languages
- Shell interpreters (Shells)
- It’s about context

3. Practical Usage
- The prefix key
- Session persistence and the server model
- It’s all commands

4. Server
- What? tmux is a server?
- Zero config needed
- Stayin’ alive
- Servers hold sessions
- How servers are “named”
- Clients
- Clipboard

5. Sessions
- Creating a session

Switching sessions within tmux
Naming sessions
Does my session exist?

6. Windows
Naming windows
Traversing Windows
Moving Windows
Layouts
Closing windows

7. Panes
Creating new panes
Traversing Panes
Zoom in
Resizing panes
Outputting pane to a file

8. Configuration
Updating configs in current sessions
How configs work
Common options

9. Status bar
The symbology behind windows
Date and time
Shell command output
Styling
Tweaking your status bar, live!
Turn your status line off
Example: default config
Example: Dressed up
Example: Powerline

10. Scripting tmux
Aliases
Pattern matching
Targets
Formats

11. Tips and Tricks
Read the tmux manual in style
Log tailing
File watching

Session Managers
More code and examples
tmux-plugins and tpm

12. Cheatsheets
Commands
Keybindings
Formats

Appendix: Installing tmux
macOS / OS X
Linux
BSD
Windows 10

Appendix: tmux on Windows 10

Troubleshooting
`E353: Nothing in register *` when pasting on vim
`tmuxp: command not found` and `powerline: command not found`

Foreword

Pretty much all my friends use tmux. I remember going out at night for drinks and the 3 of us would take a seat at a round table and take our smart phones out. This was back when phones still had physical "QWERTY" keyboards.

Despite our home computers being asleep or turned off, our usernames in the IRC channel we frequently visited persisted in the chatroom list. Our screens were lit by a kaleidoscope of colors on a black background. We ssh'd with ConnectBot into our cloud servers and reattached by running `screen(1)`. Just as it hit 2AM, our Turkish coffee arrived, the `|away` status indicator trailing our online nicknames disappeared.

It was funny that even though we knew each other by our real names, we sometimes opted to call each other by our nicks. It's something about how personal relationships, formed completely online, persist in real life.

It seemed as if it were orchestrated, but each of us fell into the same ebb and flow of living our lives. No one told us to do it, but bit by bit we incrementally optimized our lifestyles personally and professionally to arrive at destinations that felt eerily alike.

Like many things in life, when we act on autopilot, we sometimes arrive at similar destinations. This is often completely unplanned.

So when I go off and write an educational book about a computer application, I hope to write it for human beings. I'm not trying to sell you on tmux, not to make you like it, or hate it, but to tell you what it is, how some people use it. I'll leave the rest up to you.

About this book

I've helped thousands learn tmux through my free resource under the name The Tao of tmux, which I kept as part of the documentation for the tmuxp session manager. And now, it's been expanded into a full blown book with refined graphics, examples and much more.

You do not need a book to use or understand tmux. If you want a technical manual, look at the manpage for tmux. Manpages, however, are almost never sufficient to wrap your brain around abstract concepts, they're there for reference. This learning book is the culmination of years of explaining tmux to others online and in person.

In this book we will break down tmux by the way of its objects, from servers down to panes. It also includes a rehash of terminal facilities we use every day to keep us autodidacts up to speed with what is what. On top of that, I've included numerous examples of projects, permissively licensed source code and workflows designed for efficiency in the world of the terminal.

tmux is a tool I find incredibly useful. While I don't attach it to my personal identity, it's been part of my daily life for years. In addition to the original resource, I've written a popular tmux starter configuration, a pythonic tmux library, and a tmux session manager.

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

A word to absolute beginners: don't feel you need to grasp the concepts of the command line and terminal multiplexing in a single sitting. You have the choice of picking out concepts of tmux that you like according to your needs or interests. If you haven't installed tmux yet, please view the Installation section in the Appendix of the book.

Feel free to follow @TheTaoOfTmux for updates or share on Twitter!

Styles

Formatted text `like this` is source code.

Formatted text with a $ in front is a terminal command. `$ echo 'like this'`. This means you type that text right into the console, without the dollar character. For more information on the meaning of the "dollar prompt", check out *What is the origin of the UNIX $ (dollar) prompt?* on Super User.

Shortcuts require a prefix key to be sent before hand. Sections describing similar keyboard commands typically will appear in a table. For example:

Shortcut	**Action**
`Prefix` + `d`	Detach client from session.

How this book is structured

First, anything involving installation and hard technical details are in the Appendix. A lot of books tend to use installation instructions as filler in the early chapters. For me it's more of not wanting to confuse complete beginners.

For special circumstances like tmux on Windows 10, I decided that adding screenshots is best since many readers may be more comfortable with a visual approach.

Thinking Tmux goes over what tmux does and how it relates to the GUI desktops already on our computers. You'll understand the big picture of what tmux is and how it can make your life easier.

Terminal Fundamentals shows the text-based environments you'll be dealing with. It's great for those new to tmux, but also presents some 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 start using tmux immediately.

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

Sessions are the containers holding windows. You'll learn what sessions are and how they helps 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 set the foundation for how to think about `.tmux.conf` so you can customize your own.

Status bar is devoted singularly to the customization of the status line in tmux. You'll even learn how to show CPU and memory usage via the status line.

Scripting tmux goes into command shorthands, as well as 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 be able to impress friends who currently use tmux daily. 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 in conjunction with tmux to improve day to day development and administration experience.

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

Donations

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.

Available Formats

This book is available for sale on Leanpub and Amazon Kindle.

It's also available to read for free on the web.

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 <AT> 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.

Thanks

Thanks to the contributors that helped me spot errors in this book.

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

Book Updates and tmux changes

This book was written for tmux 2.3, released September 2016.

As of January 2017, it's pretty trivial for me to push out minor changes to this book on Leanpub and Amazon.

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

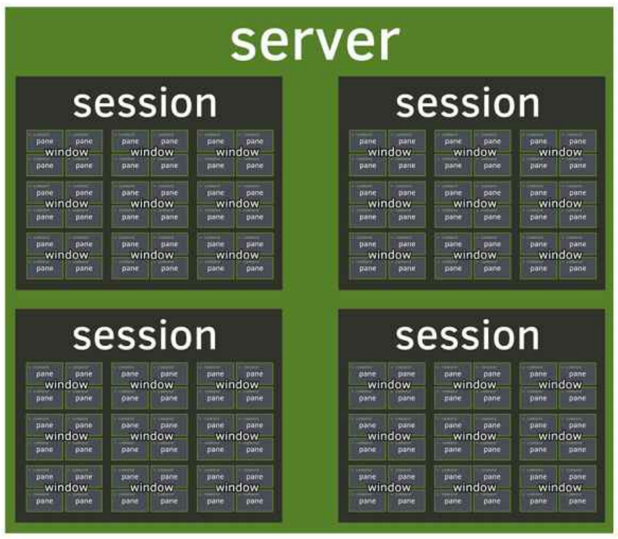
You can refer to tmux's `CHANGES` for a list of changes between versions.

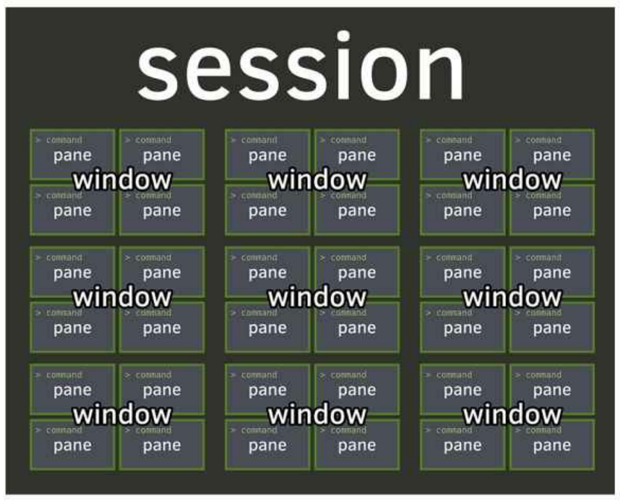
1. Thinking in tmux

In the world of modern computing, user interaction has 2 realms:

1. The text realm
2. The graphical realm

tmux lives in the graphical realm in which fixed-width fonts appear in a rectangular grid in a window, like in a terminal from the 1980s.





Window manager for the terminal

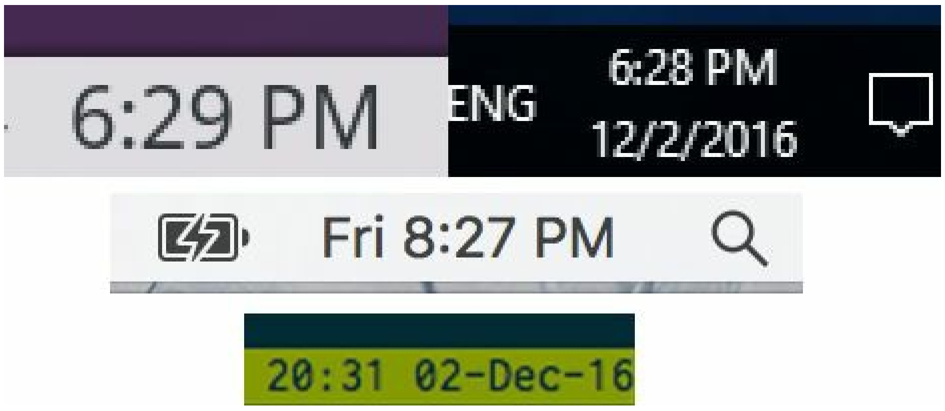
tmux is to the console what a desktop is to gui apps. It's a world inside the text dimension. Inside tmux you can:

- multitask inside the terminal, run multiple applications
- have multiple command lines (pane) in the same window
- have multiple windows (window) in the workspace (session)

- switch between multiple workspaces, like virtual desktops

tmux	"Desktop"-Speak	Plain English
Multiplexer	Multi-tasking	Multiple applications simultaneously.
Session	Desktop	Applications are visible here
Window	Virtual Desktop or applications	A desktop that stores it own screen screen
Pane	Application	Performs operations

Just like in a graphical desktop environment, they throw in a clock too.



top-left: KDE. top-right: Windows 10. center: macOS Sierra. bottom: tmux 2.3 default status bar.

Multitasking

tmux allows you to keep multiple terminals running on the same screen.

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

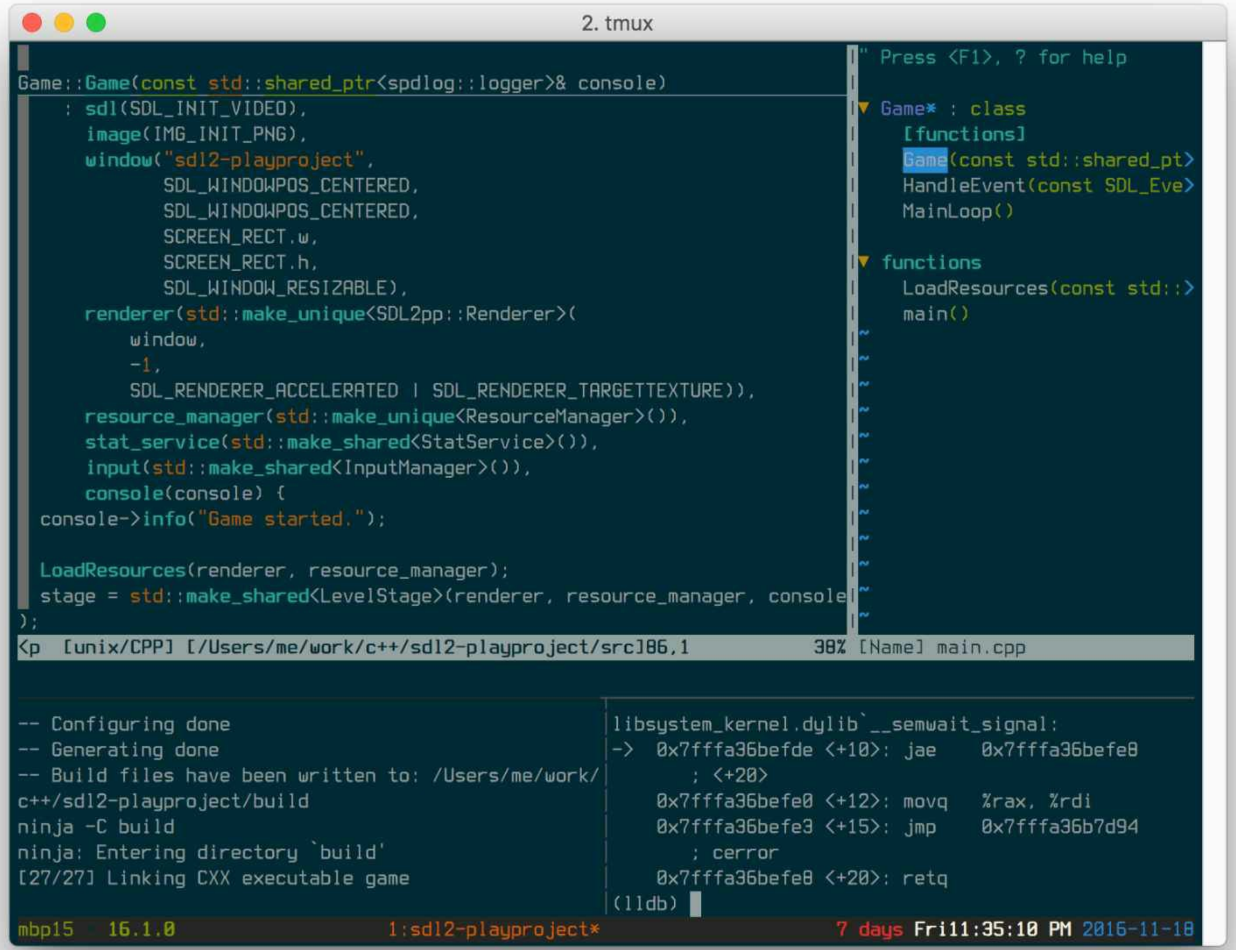
In addition multiple terminals on one screen, tmux allows you to create and link multiple "windows", all within the confines of the tmux session you attached.

Even better, you can copy and paste as well as scroll. No requirement for graphics either, so you have full power even if you're SSH'ing or on a system without X.

Here are a few common scenarios:

- Running `tail -F /var/log/apache2/error.log` in a pane to get a live stream of the latest system events.
- Running a file watcher like watchman, gulp-watch, grunt-watch, guard or entr. On file change, you could do stuff like:
  - 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 your Express server
    - run any other custom command of your liking
- Keeping a text editor like vim, emacs, pico, nano, etc. open in a main pane, while leaving two other open for CLI commands and building via `make` or `ninja`.


```
2. tmux

Game::Game(const std::shared_ptr<spdlog::logger>& console)        " Press <F1>, ? for help
    : sdl(SDL_INIT_VIDEO),
      image(IMG_INIT_PNG),                                        ▼ Game* : class
      window("sdl2-playproject",                                      [functions]
             SDL_WINDOWPOS_CENTERED,                                  Game(const std::shared_pt>
             SDL_WINDOWPOS_CENTERED,                                  HandleEvent(const SDL_Eve>
             SDL_WINDOWPOS_CENTERED,                                  MainLoop()
             SCREEN_RECT.w,
             SCREEN_RECT.h,                                       ▼ functions
             SDL_WINDOW_RESIZABLE),                                   LoadResources(const std::>
      renderer(std::make_unique<SDL2pp::Renderer>(                    main()
          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% [Name] main.cpp

-- Configuring done                                   |libsystem_kernel.dylib`__semwait_signal:
-- Generating done                                    |->  0x7fffa36befde <+10>: jae    0x7fffa36befe8
-- Build files have been written to: /Users/me/work/ |         ; <+20>
c++/sdl2-playproject/build                            |    0x7fffa36befe0 <+12>: movq   %rax, %rdi
ninja -C build                                        |    0x7fffa36befe3 <+15>: jmp    0x7fffa36b7d94
ninja: Entering directory `build'                     |         ; cerror
[27/27] Linking CXX executable game                   |    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

With tmux, you quickly have the makings of an IDE! And on your terms.

Keep your applications running in the background

Sometimes GUI applications will have an option to be sidelined to the system tray to run in the background. The application is out of the 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.

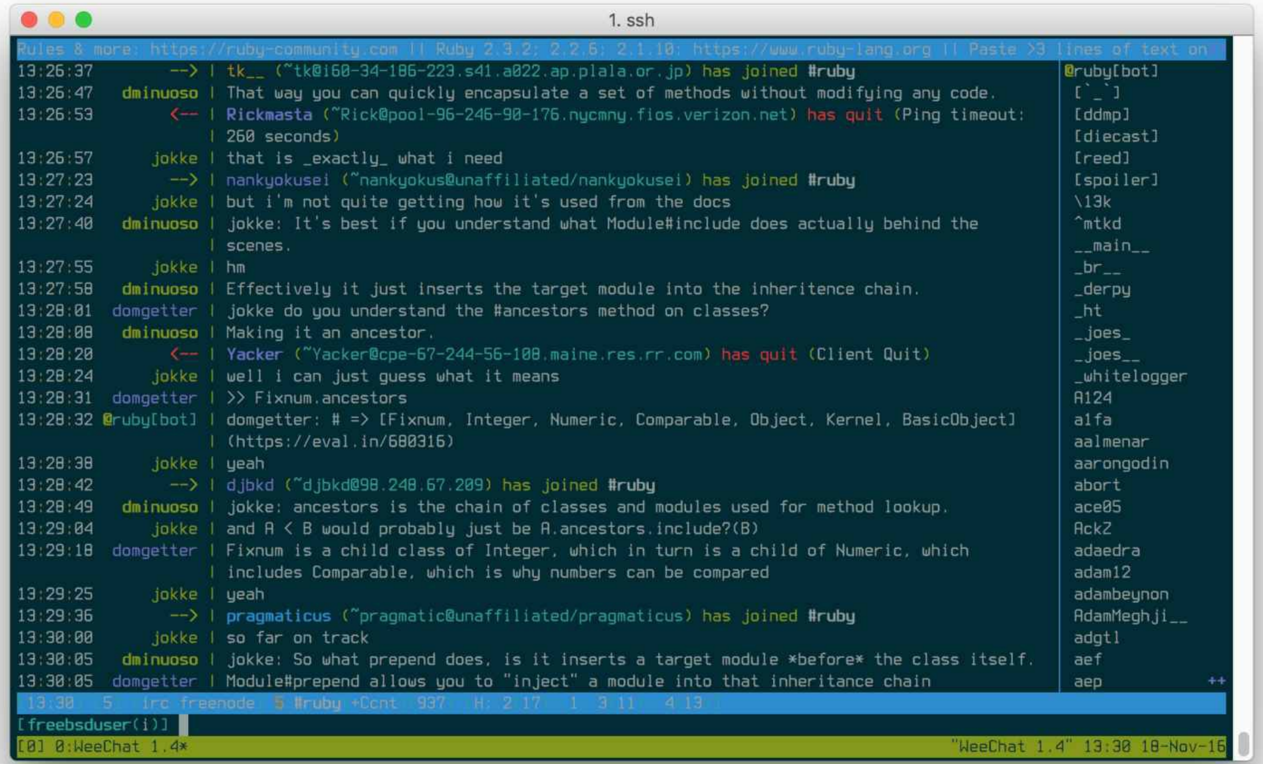
Detaching can be especially useful on:

- Local machines. You start all your normal terminal applications within a tmux session, you restart X. Instead of losing your processes as your normally would if you were using an X terminal like xterm or konsole, you'd be able to `tmux attach` after and find all processes 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 next morning reattach your session. Ahhh. Refreshing. :)

- Those servers you rarely log into. It could be that cloud instance you have that you log into 9 months later, and as a reflex, `tmux attach` to see if there we anything on there. And boom, you’re back in a session that’s 9 months old. It’s like a hack to restore your short term memory.

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.



```
1. ssh
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.

You can also have multiple users attach their clients to the same sessions, which is great for pair programming. If you were in the same session, you and the other person would see the same thing, share the same input, and the same active window and pane.

The above are just examples, but any general workspace you'd normally use in a terminal for any task can gain the benefit of you being able to persist it! That includes projects or repetitive efforts you'd multitask on. The *tips and tricks* section will dive into specific flows you can start using today.

Do tmux sessions persist after a system restart?

Unfortunately not. A restart will kill the tmux server and any processes running within it.

Thankfully, the modern server can stay online for a long time. Even for consumer laptops and PC's with a day or two uptime, having tmux persist tasks for organizational purposes is satisfactory to run it.

It comes as a disappointment because some are interested in the idea of being able to persist a tree of processes after restart. That goes out of scope of what tmux is meant to do.

For tasks you repeat often, you can always use a tool like tmuxp, tmuxinator or teamocil to resume common sessions.

In addition to session managers, tmux-resurrect is a tool that attempts to preserve running programs, working directories and so on within tmux. The benefit with tmux-resurrect is there's no JSON/YAML config needed.

2. Terminal fundamentals

Before getting into tmux, a few fundamentals of the command line should be reviewed. Often, we're so used to using these out of street smarts and muscle memory a great deal of us never see the relation of where these tools stand next to each other.

Seasoned developers are familiar with zsh, Bash, iTerm2, konsole, /dev/tty, shell scripting, and so on. If you use tmux, you'll be around these all the time, regardless of whether you're in a GUI on a local machine or SSH'ing into a remote server.

If you want to learn more about how processes and TTY's work at the kernel level (data structures and all) the book *The Design and Implementation of the FreeBSD Operating System (2nd Edition)* by Marshall Kirk McKusick is nice. In particular, Chapter 4, *Process Management* and Section 8.6, *Terminal Handling*. *The TTY demystified* by Linus Åkesson (available online) dives into the TTY and is a good read as well.

Much more exists to glean off the history of Unix, 4.2 BSD, etc. I probably could have a coffee / tea with you discussing it for hours. You could look at it from multiple perpsectives (The C Language, anything from the Unix/BSD lineage, etc.) and some clever fellow would likely chime in mentioning Linux, GNU and so on. It's like *Game of Thrones*, there's multiple story arcs you can follow, some of which intersect. A few good resources would be A Narrative History of BSD by Marshall Kirk McKusick (Video), The UNIX Operating System by AT&T (Video), Early days of Unix and design of sh (Video) by Stephen R. Bourne.

POSIX stuff

Operating systems like macOS (formerly OS X), Linux and the BSD's all 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)`, which is also a POSIX standard, is in actuality GNU Make on macOS by default. Did you know that 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 as compatible as possible 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 span across 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 so much 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 kind of stuff everywhere. So let's separate the common suspects into their respective categories.

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.

That's your TTY's, including text terminals and X sessions that live within them. On Linux / BSD systems, you can switch between sessions via `<ctrl-alt-F1>` through `<ctrl-alt-F12>`.

Terminal emulators

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

Shell languages

Shell languages *are* programming languages.

Sure you may not compile the code into binaries with `gcc` or `clang`. And there may not be a shiny npm for them. But a language is a language.

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

The top of `.sh` files shebang statement which can invoke shellscripts in different dialects.

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 that other languages entail.

Shell interpreters (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 to, interactively, run commands and applications inside them.

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

When it comes to 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. That's because it's what Apple decided to use as a *default shell* for users.

On BSD you may be placed into use 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 like Bash, Zsh and so on add a lot of helpful and practical features may work on one interpreter, but not another. For instance, you would need to install the 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 installs 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 that in order for `chsh -s` to work you typically need to have it added to `/etc/shells`.

It's about context

To wrap it up, you're going to 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 itself, the scripting issue at hand or something about the shell language.

But overall, once you get out of this chapter, go back to doing what you're doing, if shell is what people say and they understand it, use it. The backing you got here should make you more confident in yourself. It's an ongoing battle these days to keep street smarts we pick up with book smarts.

3. Practical Usage

This is the easiest part, open up your terminal and type `tmux`, hit enter.

```
$ tmux
```

You're in tmux.

The prefix key

The *prefix* is how we send commands into tmux. With this, we can split windows, move windows, switch windows, switch sessions, send in custom commands, you name it.

And it's a hump we have to get over.

It's kind of like *Street Fighter*. In this video game the player inputs a combination of buttons in sequence to perform flying spinning kicks and shoot fireballs; sweet. As the player grows more accustomed with the combos, they begin to repeat moves by intuition since they grow muscle memory.

Without understanding how to send *command sequences* to tmux via the prefix key, you'll pretty much be dead in the water.

Key sequences will come up later if you use Vim, Emacs or other TUI (Terminal User Interface) applications. If you haven't internalized the concept, let's do it now. If you already have done similar command sequences before in TUI/GUI applications, that'll come in handy.

When you memorize a key combo it's one less time you'll be moving your hand away from the keyboard to grab your mouse. You can focus your short term memory on getting stuff done, that means less mistakes.

Coming from `GNU Screen`?

Your tmux prefix key can be set via your tmux configuration later on! In your `~/.tmux.conf` file, set the `prefix` option:

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

This will set the prefix key to `screen(1)`'s (another terminal multiplexer's) prefix key.

The default leader prefix is `<Ctrl-b>`. That's holding down the `control` key, then `b`.

Sending tmux commands

Practice:

1. Press `control` key down, and *hold it.*
2. Press `b`, and hold it.
3. Release both keys at the same time.

That's all it takes. Now let's try something:

`<Ctrl-b> d`. So,

1. Press `control` key down, and *hold it.*
2. Press `b`, and hold it.
3. Release both keys at the same time.
4. Hit `d`!

You've sent tmux your first command, and you're now outside of tmux!

You've detached the tmux session you were in.

Session persistence and the server model

If you use Linux or similar system, you've likely brushed through Job Control, such as `fg(1)`, `jobs(1)` before. tmux behavior feels similar, it feels 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.

It’s all commands

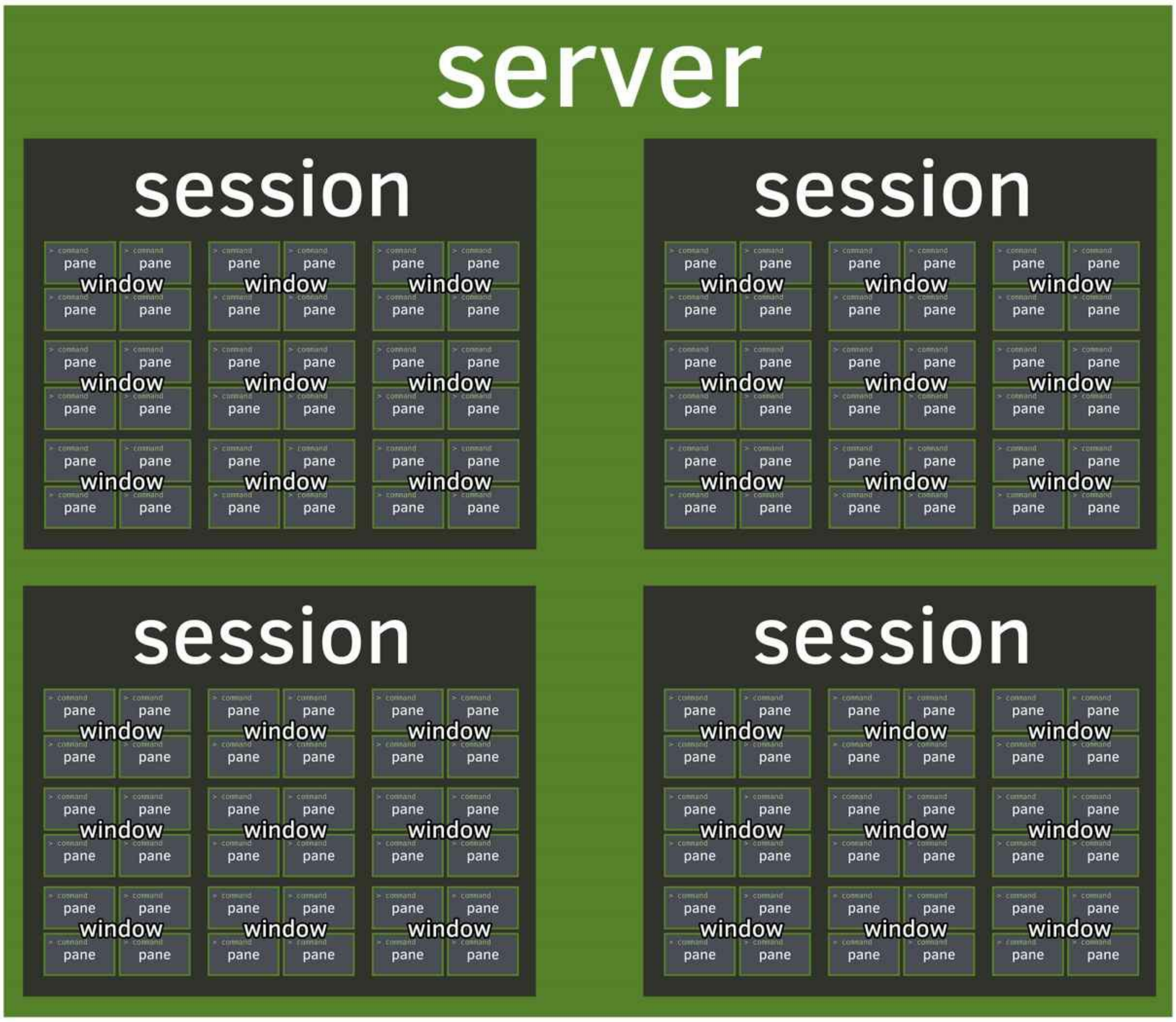
Multiple roads can lead you to the same behavior. *Commands* are what tmux uses to define instructions for setting options, resizing, renaming, traversing, switching modes, copying and pasting and so forth.

- 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 keybindings via `bind-key`.
- Commands called from CLI via `$ tmux cmd`
- To pull it all together, source code files are prefixed `cmd-`.

4. Server

Your server holds sessions and the windows and Panes within them.

What happens in tmux is you are connected via a socket connection to a server. What you see presented to you in your shell is merely a client connection. In this chapter, we go into the one of the secret sauces that allow your terminal applications to persist for months or even years if you want to.



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.

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 solid understanding that while tmux feels like magic, it's utilitarian first and foremost. 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 that I made which utilize these features.

It surprises some because servers often beget a setup process. But servers being involved doesn't necessarily 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.

Stayin' alive

The server part of tmux is how your sessions are able to stay alive even after your client is detached.

You can detach a tmux session from an SSH server and reconnect at a later time. 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 first.

Servers hold sessions

One server can contain one or multiple sessions.

Recurring usage of tmux after a server already exist will create a new session inside that server.

Advanced: Multiple servers

tmux is nimble. If you want to use a separate server, pass in the `-L` flag to any command.

`tmux -L moo` - start a new tmux server + session if non-exists under the socket name "moo"

`tmux -L moo attach` try to re-attach to session if one exists

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`.

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
```

There commands in practice are rather rare. As well as the other `list-` commands for that matter. But they are part of the tools that make tmux highly scriptable should you want to get creative. You can learn more about that in formats.

Clipboard

tmux clients wield a powerful clipboard feature you can use to copy and paste across sessions, windows and panes.

Much like vi, tmux handles clipboard as a mode (or a state) which a pane is temporarily placed in while text can be copied.

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

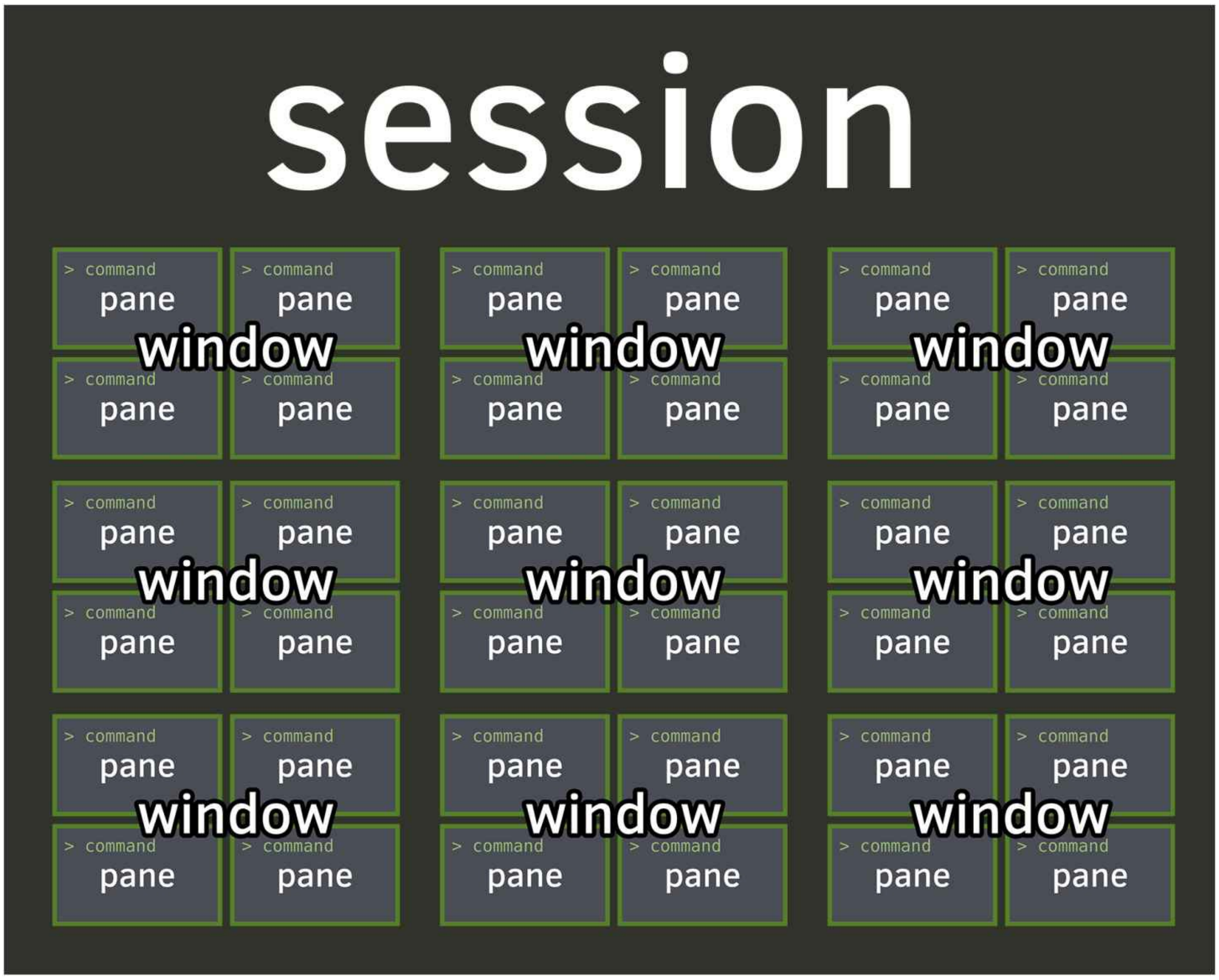
In your config, put this:

```
# 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
```

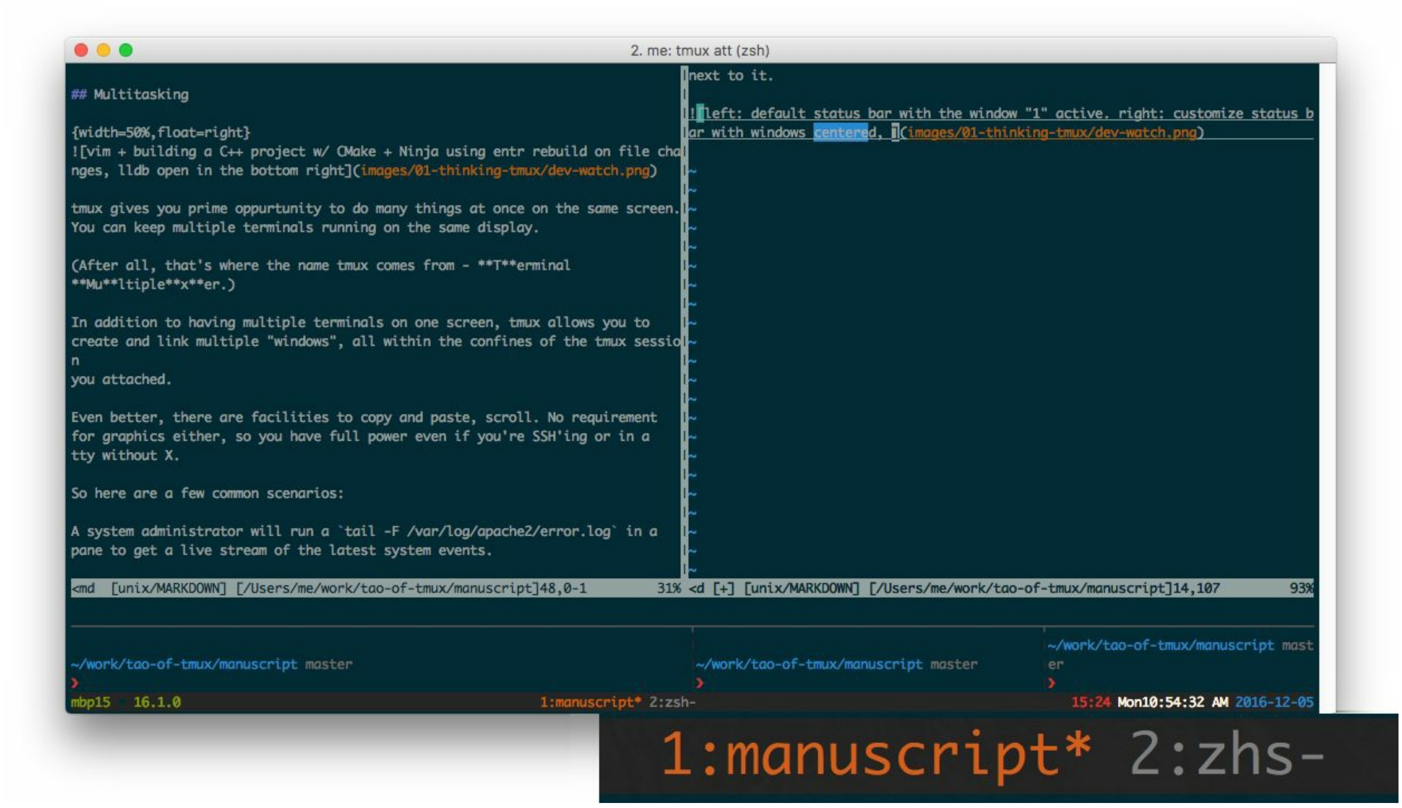
5. Sessions

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.

A session holds one or more windows.



The active window will have a * symbol next to it.



```
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:zhs-
```

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

Creating a session

The simplest command to create a new session is just typing `tmux`:

```
$ tmux
```

The `$ tmux` application without any commands is equivalent to `$ tmux new-session`. Nifty!

By default, your session name will be given a number. Which isn't too descriptive. What would be better is:

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

Switching sessions within tmux

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

Shortcut	Action
`Prefix` + `(`	Switch the attached client to the previous session.
`Prefix` + `)`	Switch the attached client to the next session.
`Prefix` + `L`	Switch the attached client back to the last session.
`Prefix` + `s`	Select a new session for the attached client interactively.

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

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

Example usage:

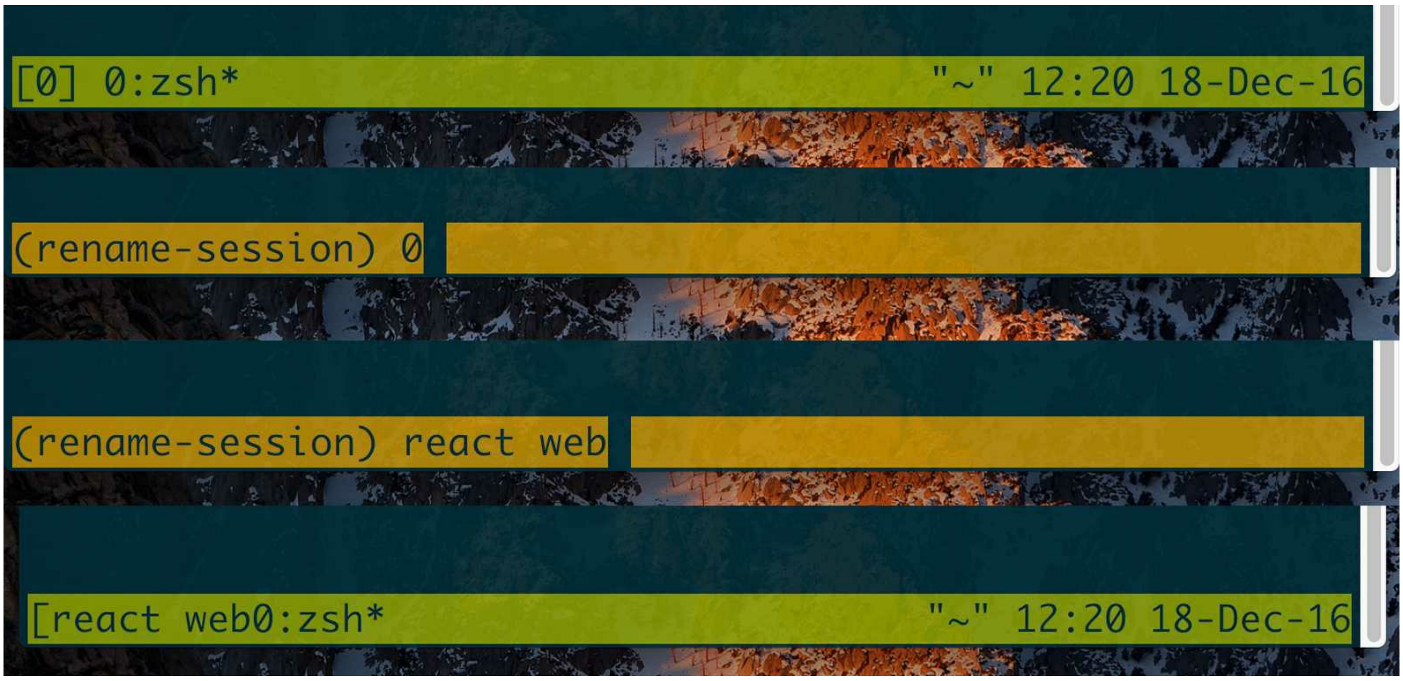
```
$ tmux switch-client -t dev
```

This will switch to a session named "dev", if it exists. No need to enter the `target-client` if you're in a client already.

Naming sessions

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 pretty much 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 'zsh' to 'renamed'

If you don't name your sessions, it'll be difficult to keep track of what is inside that session from the outside. Sometimes you may forget you already have a project opened that is a few days old and you can just re-attach or switch to that.

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 -t1 "my session"
```

Does my session exist?

If you're scripting tmux, you will want to be able to see if a session already 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 -t1
```

It's assume 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 -t1
> can't find session 4
```

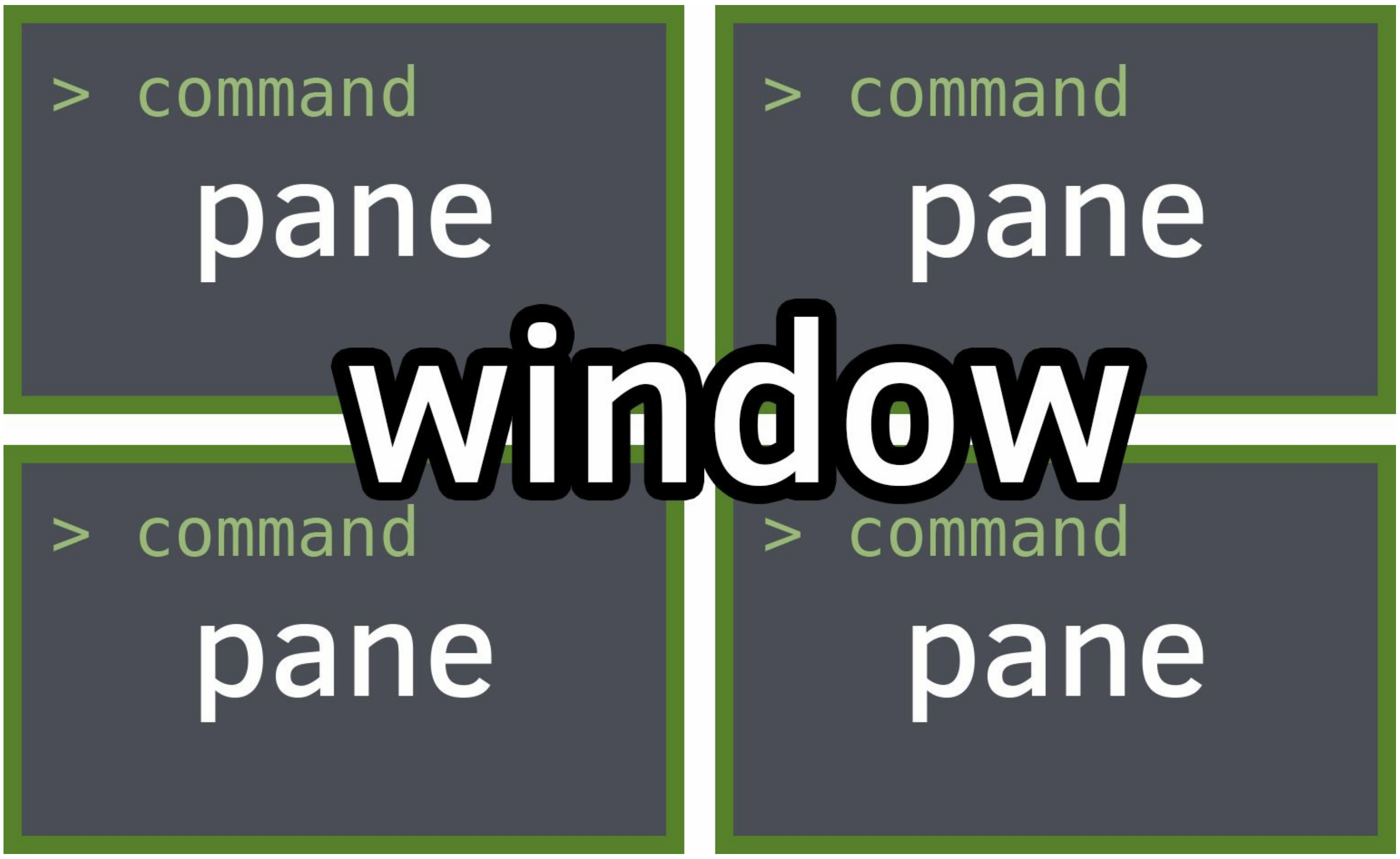
To try it in a shell script:

```
if tmux has-session -t0 ; then
    echo "has session 0"
fi
```

6. Windows

Windows hold panes. They reside within a session.

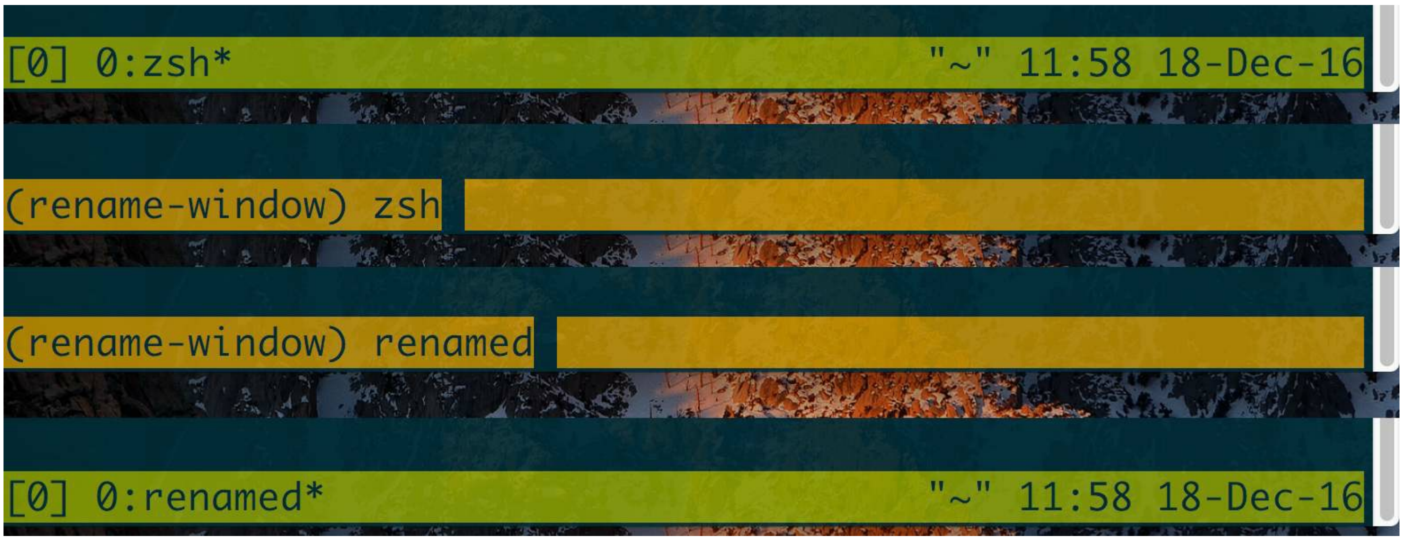
They also have layouts, which can be one of many preset dimensions or a custom one done through pane resizing.



You can see the current open windows through the status bar.

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

When inside tmux, the most common way of doing that is `Prefix` + `,`. This will open a prompt in the tmux status line where you can alter the name of the current window.

Traversing Windows

`Prefix` + `1`, `Prefix` + `2`, and so on… will get you to navigate to windows by their index.

Prompt for a window index (useful for indexes greater than 9) with `Prefix` + `'`.

POWER MOVE: Search + Traverse Windows for Text

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

You can move to the last selected window with `Prefix` + `l`.

You can bring up a list of current windows with `Prefix` + `w`. The benefit of this is it also gives you some info on what's inside the window. Helpful if you're juggling around a lot of things!

Moving Windows

`$ 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:

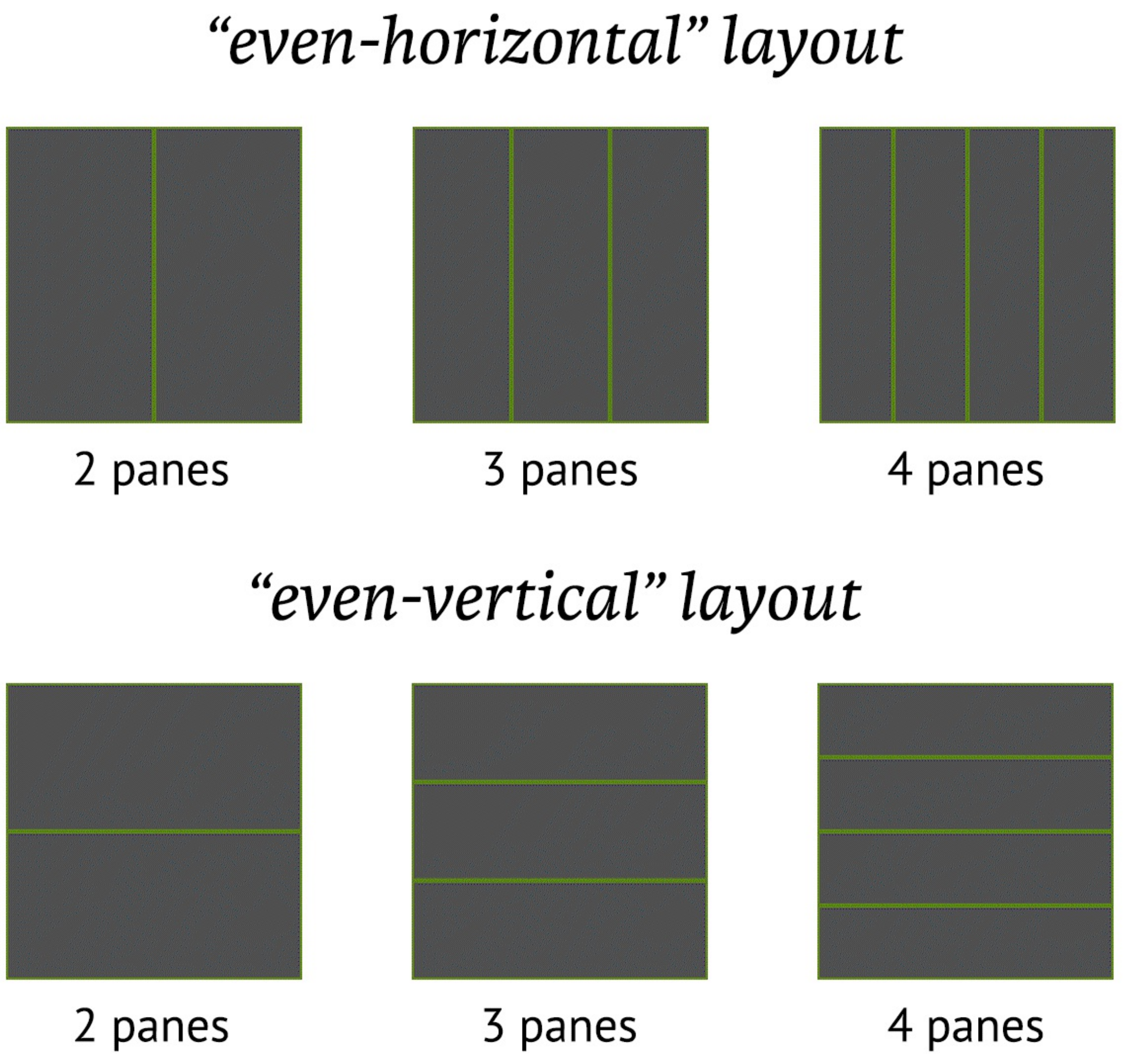
```
$ tmux movew -s2 -t1
```

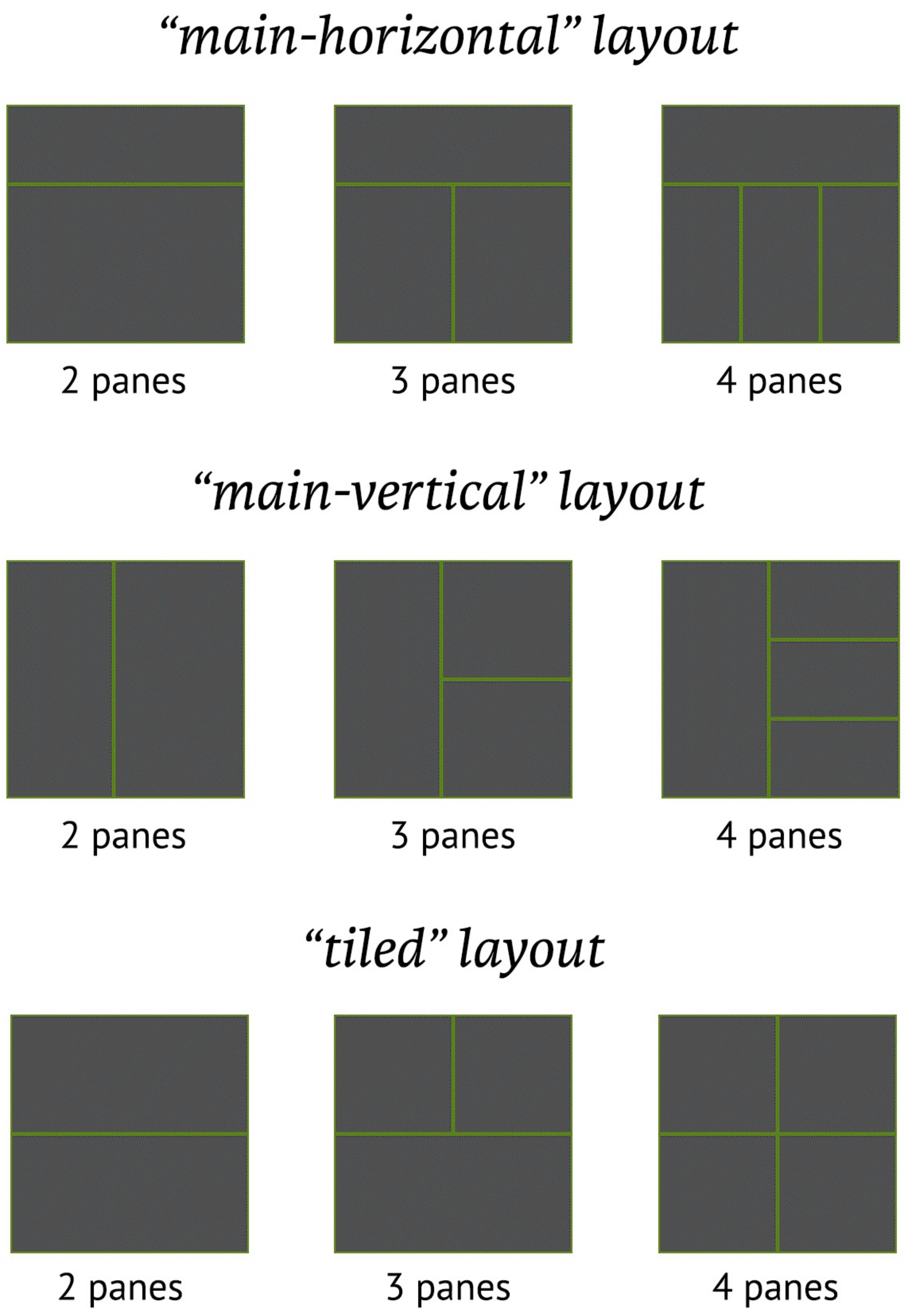
The shortcut to prompt for an index to move the current window to is `Prefix` + `.`.

Layouts

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

As of tmux 2.3, the supported layouts are:





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 easily 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 everytime you do `Prefix` + `m`.

Layouts can also be totally 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 that layout, do this:

```
$ 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}]"
```

Closing windows

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
```

You can easily find the window index through the middle section of the status line.

7. Panes

Panes are pseudoterminals that contain your shell (e.g. Bash, Zsh). They reside within a window.

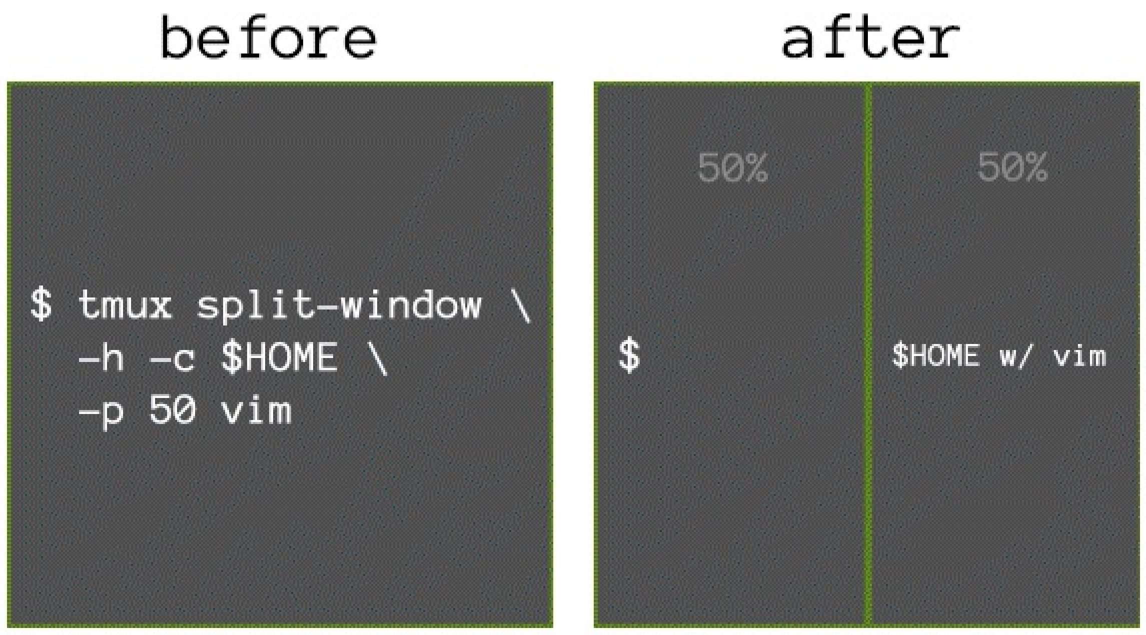
Creating new panes

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

Shortcut	Action
`Prefix` + `%`	`split-window -h` (split horizontally)
`Prefix` + `"`	`split-window -v` (split vertically)

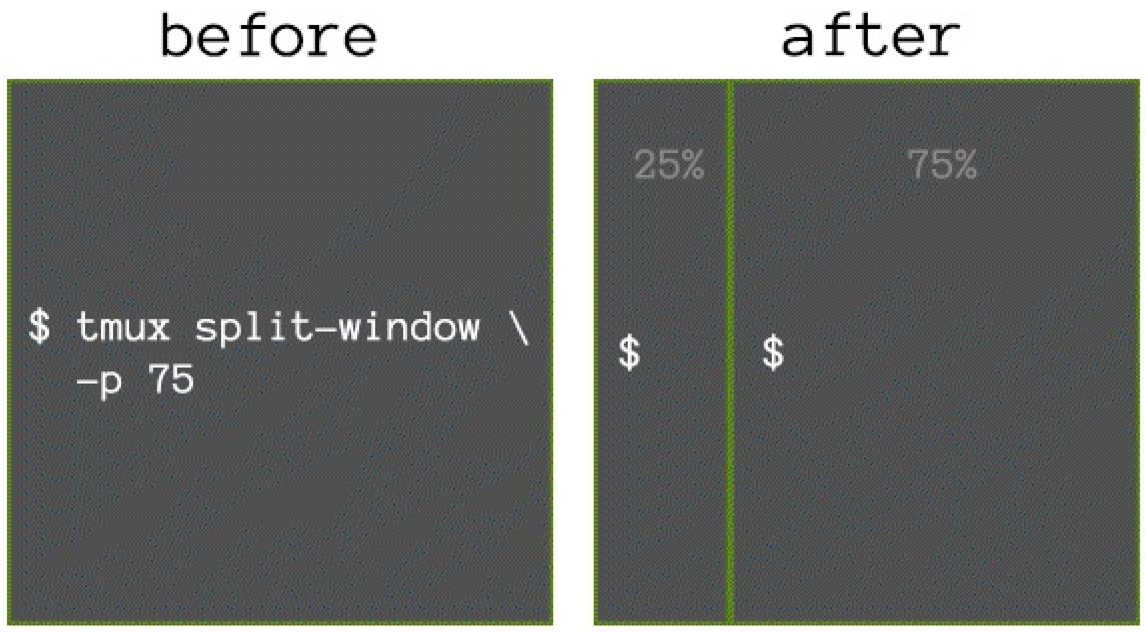
Example usage:

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



```
$ tmux split-window \
  -h -c $HOME \
  -p 50 vim
```

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



```
$ tmux split-window \
  -p 75
```

Traversing Panes

Shortcut	Action
`Prefix` + `;`	Move to the previously active pane.
`Prefix` + `Up` / `Down` / `Left` / `Right`	Change to the pane above, below, to the left, or to the the right of the current pane.
`Prefix` + `o`	Select the next pane in the current window.

Movin 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
```

Zoom in

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

You can use any pane traversal to unzoom and move a pane at the same time.

Resizing panes

Panes can be resized within windows.

Another technique that resizes panes is window layouts. The difference is a window layout switch the proportions and order of the panes. Resizing the panes target a specific pane inside that window.

Resizing a pane in a specific layout may subsequently resize that whole row.

Shortcut	Action
`Prefix M-Up`	`resize-pane -U 5`
`Prefix M-Down`	`resize-pane -D 5`
`Prefix M-Left`	`resize-pane -L 5`
`Prefix M-Right`	`resize-pane -R 5`
`Prefix C-Up`	`resize-pane -U`
`Prefix C-Down`	`resize-pane -D`
`Prefix C-Left`	`resize-pane -L`

Prefix C-Right	resize-pane -R

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. Configuration

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.

For a sample config, I maintain a pretty decked out one at https://github.com/tony/tmux-config.

Custom Configs

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

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

Note that if a tmux server is already 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
```

Updating configs in current sessions

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

`Prefix` + `:` will open up 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.

Easy reloadin'

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.

How configs work

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 that prefixes `tmux` in front of every entry and run that file 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
```

Common options

Tweak wait time between key sequences:

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

`-s` sets the option server wide.

Set the starting number (base index) for windows:

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

Will make newly created windows start at 1 and count upwards.

Customize your prefix key:

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

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 '%%'"
```

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.

9. Status bar

The status bar, or *status line* lies in the bottom of the screen. It is customizable through the [.tmux.conf config]() and live through `set-option`.

Finding your current status line settings

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

The status line is compromised of 3 sections. The status fields on either side of the status line are customizable. The center field is a window list.

The `status-left` and `status-right` option can be configured to accept a variety of variables.

The symbology behind windows

The center part of the status line contains a list of windows, each of which can be followed by a symbol:

Symbol	Meaning
*	Denotes the current window.
-	Marks the last window (previously selected).
#	Window is monitored and activity has been detected.
!	A bell has occurred in the window.
~	The window has been silent for the monitor-silence interval.
M	The window contains the marked pane.
Z	The window's active pane is zoomed.

Date and time

`status-left` and `status-right` accepts variables for the date.

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

For a full list of the variables you can use, view the documentation for `strftime(3)`. You find that in the link above, or through your manpages by typing `$ man strftime`.

Shell command output

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

Styling

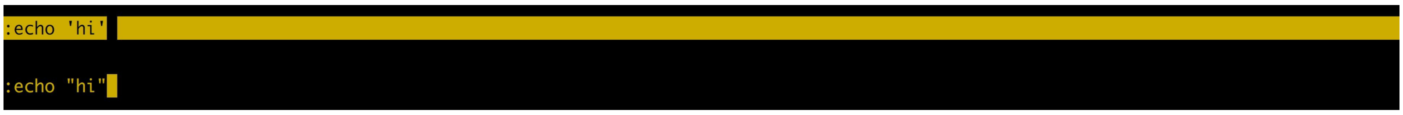
You can use `[bg=color]` and `[fg=color]` to adjust the text color and background within for status line text.

Prompt colors

The benefit of wrapping your around this type of 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.

Tweaking your status bar, live!

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

First 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 the variable off 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"
```

Turn your status line off

Turn it off:

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

Turn it on:

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

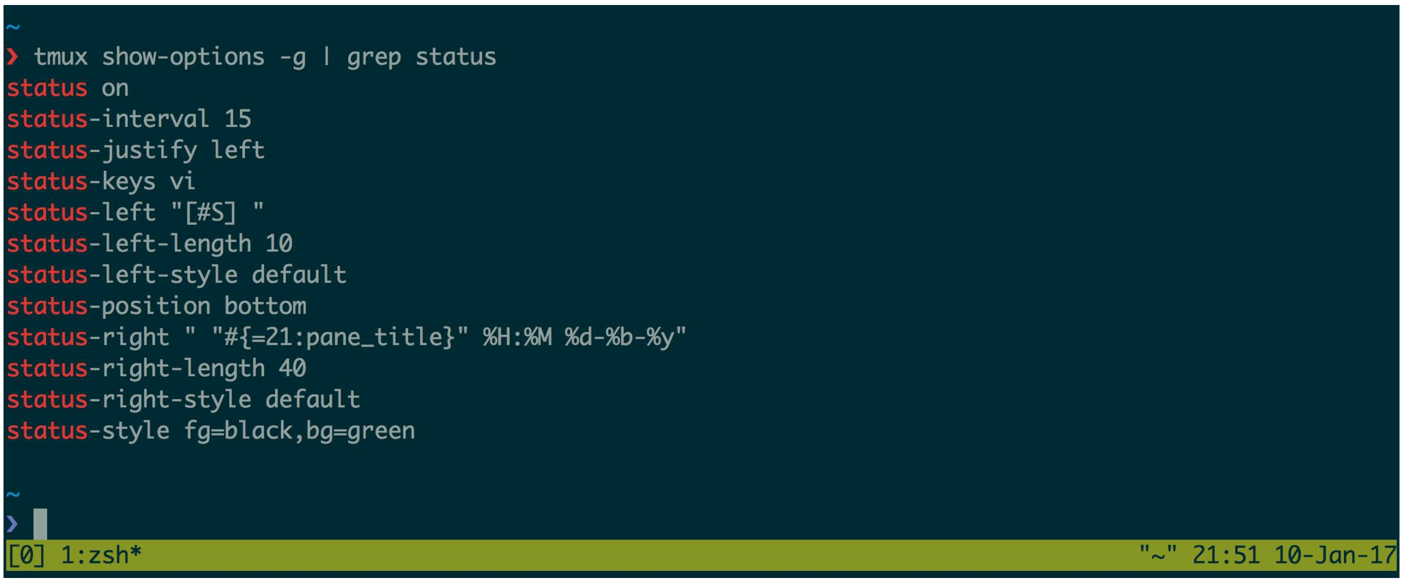
Toggle it (regardless or current state):

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

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

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

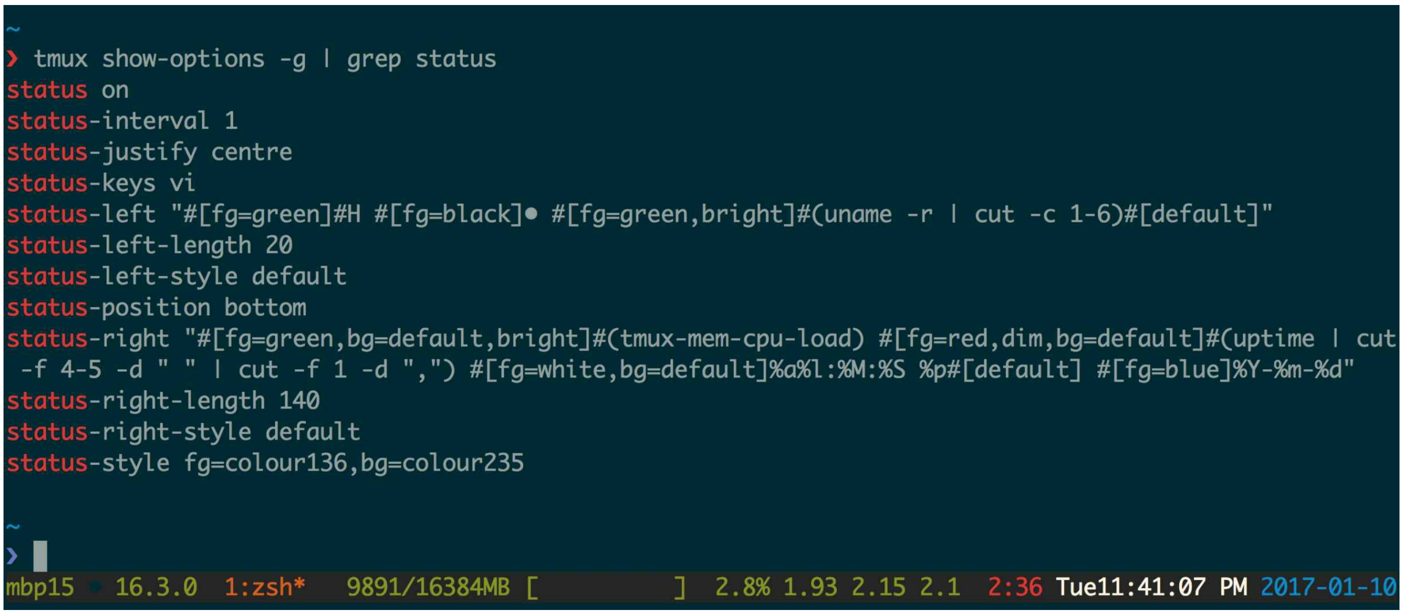
Example: default config



```
~
❯ tmux show-options -g | grep status
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

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

```
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
```

Example: Dressed up



```
~
❯ tmux show-options -g | grep status
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

~
❯ █
mbp15 • 16.3.0  1:zsh*   9891/16384MB [          ]  2.8% 1.93 2.15 2.1  2:36 Tue11:41:07 PM 2017-01-10
```

```
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=de\
fault]%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
```

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.

In order to get tmux-mem-cpu-load built 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

Example: Powerline

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

To get these to work correctly, you have to set your fonts up to handle the powerline symbols. The easiest way to use this is to install powerline fonts, which are a great collection of fixed width coder fonts which look great in terminal.

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

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

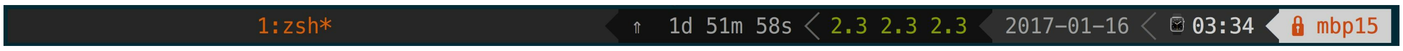
psutil is a cross-platform tool powerline uses to help gather system information.

The first option to get powerline working with tmux is sourcing `powerline.conf` from your config, the only difficulty is nailing down the location across systems and python versions. As a way to try getting powerline found across varying installations, I use `if-shell`:

```
# pip install --user git+git://github.com/powerline/powerline
if-shell 'test -f ~/.local/lib/python2.7/site-packages/powerline/bindings/tm\
ux/powerline.conf' 'source-file ~/.local/lib/python2.7/site-packages/powerli\
ne/bindings/tmux/powerline.conf'

# [sudo] pip install git+git://github.com/powerline/powerline
if-shell 'test -f /usr/local/lib/python2.7/site-packages/powerline/bindings/\
tmux/powerline.conf' 'source-file /usr/local/lib/python2.7/site-packages/pow\
erline/bindings/tmux/powerline.conf'

# [sudo] pip install git+git://github.com/powerline/powerline
if-shell 'test -f /usr/local/lib/python2.7/dist-packages/powerline/bindings/\
tmux/powerline.conf' 'source-file /usr/local/lib/python2.7/dist-packages/pow\
erline/bindings/tmux/powerline.conf'

# python 3.4
# if-shell 'test -f /usr/local/lib/python3.4/dist-packages/powerline/binding\
s/tmux/powerline.conf' 'source-file /usr/local/lib/python3.4/dist-packages/p\
owerline/bindings/tmux/powerline.conf'

# python 3.5
# if-shell 'test -f /usr/local/lib/python3.5/dist-packages/powerline/binding\
s/tmux/powerline.conf' 'source-file /usr/local/lib/python3.5/dist-packages/p\
owerline/bindings/tmux/powerline.conf'

# python 3.6
# if-shell 'test -f /usr/local/lib/python3.6/dist-packages/powerline/binding\
s/tmux/powerline.conf' 'source-file /usr/local/lib/python3.6/dist-packages/p\
owerline/bindings/tmux/powerline.conf'
```

A simpler method, after you assured properly adding python to your PATH, try adding this to your config:

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



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

Powerline requires a specialized font

10. Scripting tmux

The command line in tmux is one of those areas often uncharted.

I will use some tables in this chapter. Never get a feeling that you have to commit a table to memory immediately. Not my intention, but every person's way of using tmux is slightly different, I want to be able to cover points most likely to benefit people's flows. Full tables are in the cheatsheets.

Aliases

tmux supports a variety of alias commands. So don't feel you always have to type `tmux attach`. *Aliases*, alongside fnmatch-style pattern commands make it very intuitive to type those commands in a pinch.

Most of these aliases come to mind via intuition and are a lot friendlier than typing the full hyphenated commands.

Command	Alias
attach-session	attach
break-pane	breakp
capture-pane	capturep
display-panes	displayp
find-window	findw
join-pane	joinp
kill-pane	killp
kill-window	killw
last-pane	lastp
last-window	last
link-window	linkw
list-panes	lsp
list-windows	lsw
move-pane	movep
move-window	movew
new-session	new
new-window	neww
next-layout	nextl
next-window	next
pipe-pane	pipep

previous-layout	prevl
previous-window	prev
rename-window	renamew
resize-pane	resizep
respawn-pane	respawnp
respawn-window	respawnw
rotate-window	rotatew
select-layout	selectl
select-pane	selectp
set-option	set
set-window-option	setw
show-options	show
show-window-options	showw
split-window	splitw
swap-pane	swapp
swap-window	swapw
unlink-window	unlinkw

If you already 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.

Pattern matching

tmux commands and arguments may all be accessed via `fnmatch(3)` patterns.

For instance, you don't need to type `$ tmux attach-session` every time. First there's the alias of `$ tmux attach`, but *in addition* to that, you can pattern match `$ tmux attac`, `$ tmux att`, `$ tmux at` and `$ tmux a` work as well.

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

```
$ tmux new-session

$ tmux new-sessio

# ...

$ 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. There are multiple commands which 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 better than the matching in that case.

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.

Entity	Prefix	Example
server	n/a	n/a, uses socket-name and socket-path
client	n/a	n/a, uses /dev/tty{p,s}[000-9999]
session	$	$13
window	@	@2313
pane	%	%5432

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 kind of 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!

To give you an idea of the possibilities of where you can use targets, here are the commands with you can use targets:

```
$ tmux attach-session [-t target-session]

$ tmux detach-client [-s target-session] [-t target-client]

$ tmux has-session [-t target-session]

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

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

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

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

$ tmux new-session [-t target-session]
$ tmux refresh-client [-t target-client]

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

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

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

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

Formats

tmux provides a minimal template language and set of variables you can use 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 amount of `FORMATS` (variables) made available by tmux has 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, 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
```

Panes

Variable name	Description
cursor_x	Cursor X position in pane
cursor_y	Cursor Y position in pane
pane_active	1 if active pane
pane_current_command	Current command if available
pane_current_path	Current path if available
pane_dead	1 if pane is dead
pane_dead_status	Exit status of process in dead pane

pane_height	Height of pane
pane_id	Unique pane ID (Alias: #D)
pane_in_mode	If pane is in a mode
pane_index	Index of pane (Alias: #P)
pane_pid	PID of first process in pane
pane_start_command	Command pane started with
pane_title	Title of pane (Alias: #T)
pane_tty	Pseudo terminal of pane
pane_width	Width of pane

Sessions

Variable name	Description
session_attached	Number of clients session is attached to
session_activity	Integer time of session last activity
session_created	Integer time session created
session_last_attached	Integer time session last attached
session_group	Number of session group
session_grouped	1 if session in a group
session_height	Height of session
session_id	Unique session ID
session_many_attached	1 if multiple clients attached
session_name	Name of session (Alias: #S)
session_width	Width of session
session_windows	Number of windows in session

Windows

Variable name	Description
window_activity	Integer time of window last activity
window_activity_flag	1 if window has activity
window_active	1 if window active
window_bell_flag	1 if window has bell
window_flags	Window flags (Alias: #F)
window_height	Height of window
window_id	Unique window ID
window_index	Index of window (Alias: #I)
window_layout	Window layout description, ignoring zoomed window panes
window_linked	1 if window is linked across sessions
window_name	Name of window (Alias: #W)
window_panes	Number of panes in window

	zoomed window panes
window_width	Width of window
window_zoomed_flag	1 if window is zoomed

Servers

Variable name	Description
host	Hostname of local host (alias: #H)
host_short	Hostname of local host (no domain name) (alias: #h)
socket_path	Server socket path
start_time	Server start time
pid	Server PID

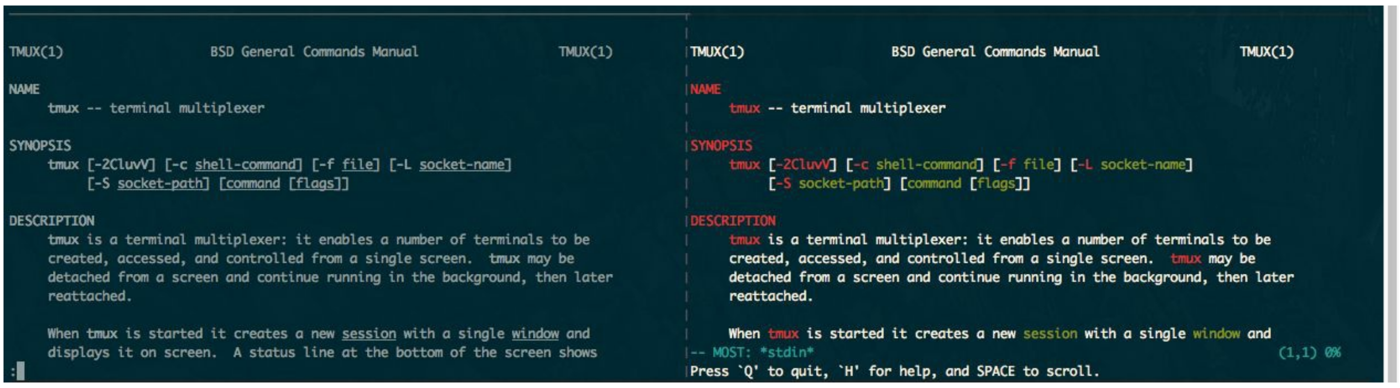
11. Tips and Tricks

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 just about any comment, here’s a couple of fun ones:

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

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



```
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

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

```
$ export PAGER=more man ls
```

If you found that 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"
```

In my configurations, I often reuse configs and some machines 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
```

So save that to a file, let's say `~/.dot-config/most.sh`.

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

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

If you keep that pattern (or something close to it), you're on your way to have a cross-platform, modular dot config. If you need some inspiration you can check my public permissively licensed config at https://github.com/tony/.dot-config. I document it pretty well and welcome you to copy/paste from it too.

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 that open in a pane. It's kind of like a Facebook newsfeed, except for programmers and system administrators.

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

File watching

In my never ending conquest to get software projects working in symphony with code changes, I've come to taste test many file watching applications and patterns. In an effort to get that perfect feedback look upon files changing, I've gradually become the internet's unofficial connoisseur on them.

What this kind of application does is wait for a file to be updated, then executes a custom command, like restarting a server, rebuilding an application, running tests, linters and so on. It gives you as a developer instant feedback in the terminal and is one of those things that can trick out a tmux workspace into an IDE-like environment.

I eventually settled on `entr(1)`, which works superbly across Linux distros, BSD's 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, and we need to do that in a cross-platform way. We can’t depend on `**` existing to grab files recursively. So something more POSIX friendly would be `find . -print | grep -i '.*[.]go'`:

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

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' | e\
ntr -c go test ./...; fi
```

And have it fallback to `go test` in the event `entr` isn’t installed (you’ll thank me when you end up scripting this command in a [session manager](#):

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

Show a notice message to user to install entr if not installed on the system:

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

Here’s why you want patterns like that: 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 that certain program installed.

Note: You may have to convert the indentation within the `Makefile`s from spaces to tabs.

So let’s go ahead 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 t\
asks when files change. \nSee http://entrproject.org/"; fi
```

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

But that was 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 up into variables and `make` subcommands. So let’s try that:

```
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
```

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

Session Managers

For those who use tmux regularly to perform repetitive tasks, such as opening the same software project, view the same logs, etc. Applications that store your layouts declaratively in a YAML or JSON file and help you boot up your session fast.

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. I written it already having used teamocil and tmuxinator but with many more features. For one, it builds on top of libtmux, a library which abstracts tmux server, sessions, windows and panes to actively 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 sessions that already exist instead of redundantly running script commands against the session if it already is 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
  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* that has 4 panes in it. 3 of them are 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.

More code and examples

I've decided to dust 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)
- golang tmux web frontend - display current tmux session and window information via browser (gin, bower)

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

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

tmux-plugins and tpm

tmux-plugins and tmux package manager are a suite of tools dedicated toward 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.

12. Cheatsheets

These are taken directly from tmux's manual pages, tabled and organized by hand into sections for convenience.

Commands

Session

Command	Action
no command	Short-cut for `new-session`
`attach-session`	Attach or switch to a session
`choose-session`	Put a window into session choice mode
`has-session`	Check and report if a session exists on the server
`kill-session`	Destroy a given session
`list-sessions`	List sessions managed by server
`lock-session`	Lock all clients attached to a session
`new-session`	Create a new session
`rename-session`	Rename a session

Window

Command	Action
`choose-window`	Put a window into window choice
`find-window`	Search for a pattern in windows
`kill-window`	Destroy a given window
`last-window`	Select the previously selected
`link-window`	Link a window to another
`list-windows`	List windows of a session
`move-window`	Move a window to another
`new-window`	Create a new window
`next-window`	Move to the next window in a sesssion
`previous-window`	Move to the previous window in session
`rename-window`	Rename a window
`respawn-window`	Reuse a window in which a command has exited
`rotate-window`	Rotate positions of panes in a window
`select-window`	Select a window
`set-window-option`	Set a window option
`show-window-options`	Show window options

`split-window`	Splits a pane into two
`swap-window`	Swap two windows
`unlink-window`	Unlink a window

Pane

Command	**Action**
`break-pane`	Break a pane from an existing into a new window
`capture-pane`	Capture the contents of a pane to a buffer
`display-panes`	Display an indicator for each visible pane
`join-pane`	Split a pane and move an existing one into the new space
`kill-pane`	Destroy a given pane
`last-pane`	Select the previously selected pane
`list-panes`	List panes of a window
`move-pane`	Move a pane into a new space
`pipe-pane`	Pipe output from a pane to a shell command
`resize-pane`	Resize a pane
`respawn-pane`	Reuse a pane in which a command has exited
`select-pane`	Make a pane the active one in the window
`swap-pane`	Swap two panes

Keybindings

Shortcut	Action
`C-b`	Send the prefix key (C-b) through to the application.

Miscellaneous

Shortcut	Action
`C-z`	Suspend the tmux client.
`r`	Force redraw of the attached client.
`t`	Show the time.
`~`	Show previous messages from tmux, if any.
`f`	Prompt to search for text in open windows.
`d`	Detach the current client.
`D`	Choose a client to detach.
`?`	List all key bindings.
`:`	Enter the tmux command prompt.

Copy/Paste

Shortcut	Action
`#`	List all paste buffers.
`[`	Enter copy mode to copy text or view the history.
`]`	Paste the most recently copied buffer of text.
`Page Up`	Enter copy mode and scroll one page up.
`=`	Choose which buffer to paste interactively from a list.
`-`	Delete the most recently copied buffer of text.

Session

Shortcut	Action
`$`	Rename the current session.

Session Traversal

Shortcut	Action
`L`	Switch the attached client back to the last session.
`s`	Select a new session for the attached client interactively.

Window

Shortcut	Action
`c`	Create a new window.
`&`	Kill the current window.
`i`	Display some information about the current window.
`,`	Rename the current window.

Window Traversal

Shortcut	Action
`0 to 9`	Select windows 0 to 9.
`w`	Choose the current window interactively.
`M-n`	Move to the next window with a bell or activity marker.
`M-p`	Move to the previous window with a bell or activity marker.
`p`	Change to the previous window.
`n`	Change to the next window.
`l`	Move to the previously selected window.
`'`	Prompt for a window index to select.

Window Moving

Shortcut	Action
`.`	Prompt for an index to move the current window.

Pane

Shortcut	Action
`x`	Kill the current pane.
`q`	Briefly display pane indexes.
`%`	Split the current pane into two, left and right.
`"`	Split the current pane into two, top and bottom.

Pane Traversal

Shortcut	Action
`;`	Move to the previously active pane.
`Up, Down` `Left, Right`	Change to the pane above, below, to the left, or to the right of the current pane.
`o`	Select the next pane in the current window.

Pane Moving

Shortcut	Action
`C-o`	Rotate the panes in the current window forwards.
`M-o`	Rotate the panes in the current window backwards.
`{`	Swap the current pane with the previous pane.
`}`	Swap the current pane with the next pane.
`!`	Break the current pane out of the window.

Pane Resizing

Shortcut	Action
`M-1 to M-5`	Arrange panes in one of the five preset layouts: even-horizontal, even-vertical, main-horizontal, main-vertical, or tiled.
`C-Up, C-Down` `C-Left, C-Right`	Resize the current pane in steps of one cell.
`M-Up, M-Down` `M-Left, M-Right`	Resize the current pane in steps of five cells.

Formats

Copy / paste

Variable name	Description
buffer_name	Name of buffer
buffer_sample	Sample of start of buffer
buffer_size	Size of the specified buffer in bytes

Clients

Variable name	Description
client_activity	Integer time client last had activity
client_created	Integer time client created
client_control_mode	1 if client is in control mode
client_height	Height of client
client_key_table	Current key table
client_last_session	Name of the client's last session
client_pid	PID of client process
client_prefix	1 if prefix key has been pressed
client_readonly	1 if client is readonly
client_session	Name of the client's session
client_termname	Terminal name of client
client_tty	Pseudo terminal of client
client_utf8	1 if client supports utf8
client_width	Width of client
line	Line number in the list

Panes

Variable name	Description
alternate_on	If pane is in alternate screen
alternate_saved_x	Saved cursor X in alternate screen
alternate_saved_y	Saved cursor Y in alternate screen
cursor_flag	Pane cursor flag
cursor_x	Cursor X position in pane
cursor_y	Cursor Y position in pane
insert_flag	Pane insert flag
keypad_cursor_flag	Pane keypad cursor flag
keypad_flag	Pane keypad flag
mouse_any_flag	Pane mouse any flag

mouse_button_flag	Pane mouse button flag
mouse_standard_flag	Pane mouse standard flag
pane_active	1 if active pane
pane_bottom	Bottom of pane
pane_current_command	Current command if available
pane_current_path	Current path if available
pane_dead	1 if pane is dead
pane_dead_status	Exit status of process in dead pane
pane_height	Height of pane
pane_id	Unique pane ID (Alias: #D)
pane_in_mode	If pane is in a mode
pane_input_off	If input to pane is disabled
pane_index	Index of pane (Alias: #P)
pane_left	Left of pane
pane_pid	PID of first process in pane
pane_right	Right of pane
pane_start_command	Command pane started with
pane_synchronized	If pane is synchronized
pane_tabs	Pane tab positions
pane_title	Title of pane (Alias: #T)
pane_top	Top of pane
pane_tty	Pseudo terminal of pane
pane_width	Width of pane
scroll_region_lower	Bottom of scroll region in pane
scroll_region_upper	Top of scroll region in pane
scroll_position	Scroll position in copy mode
wrap_flag	Pane wrap flag

Sessions

Variable name	**Description**
session_alerts	List of window indexes with alerts
session_attached	Number of clients session is attached to
session_activity	Integer time of session last activity
session_created	Integer time session created
session_last_attached	Integer time session last attached
session_group	Number of session group
session_grouped	1 if session in a group
session_height	Height of session
session_id	Unique session ID
session_many_attached	1 if multiple clients attached

session_name	Name of session (Alias: #S)
session_width	Width of session
session_windows	Number of windows in session

Windows

Variable name	Description
history_bytes	Number of bytes in window history
history_limit	Maximum window history lines
history_size	Size of history in bytes
window_activity	Integer time of window last activity
window_activity_flag	1 if window has activity
window_active	1 if window active
window_bell_flag	1 if window has bell
window_find_matches	Matched data from the find-window
window_flags	Window flags (Alias: #F)
window_height	Height of window
window_id	Unique window ID
window_index	Index of window (Alias: #I)
window_last_flag	1 if window is the last used
window_layout	Window layout description, ignoring zoomed window panes
window_linked	1 if window is linked across sessions
window_name	Name of window (Alias: #W)
window_panes	Number of panes in window
window_silence_flag	1 if window has silence alert
window_visible_layout	Window layout description, respecting zoomed window panes
window_width	Width of window
window_zoomed_flag	1 if window is zoomed

Servers

Variable name	Description
host	Hostname of local host (alias: #H)
host_short	Hostname of local host (no domain name) (alias: #h)
socket_path	Server socket path
start_time	Server start time
pid	Server PID

Commands

For `$ tmux list-commands`.

Variable name	Description
command_hooked	Name of command hooked, if any
command_name	Name of command in use, if any
command_list_name	Command name if listing commands
command_list_alias	Command alias if listing commands
command_list_usage	Command usage if listing commands

Appendix: Installing tmux

macOS / OS X

brew

```
$ brew install tmux
```

macports

```
$ sudo port install tmux
```

fink

```
$ fink install tmux
```

Linux

Ubuntu / Mint / Debian, etc.

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

CentOS / Fedora / Redhat, etc.

```
$ sudo yum install tmux
```

Arch Linux (pacman)

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

Gentoo (portage)

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

BSD

FreeBSD

pkg(1)

```
# pkg install tmux
```

pkg_add(1)

```
# pkg_add -r tmux
```

OpenBSD

As of OpenBSD 4.6, tmux is part of the base system.

If you are using an earlier version:

```
# pkg_add tmux
```

NetBSD

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

Windows 10

Check out the tmux on Windows 10 appendix section.

Appendix: tmux on Windows 10

As of Windows 10 build 14361 you can run tmux via the Linux Subsystem feature.

Usage requires enabling **Developer mode** via the “For Developers” tab in the “Update & security” settings.

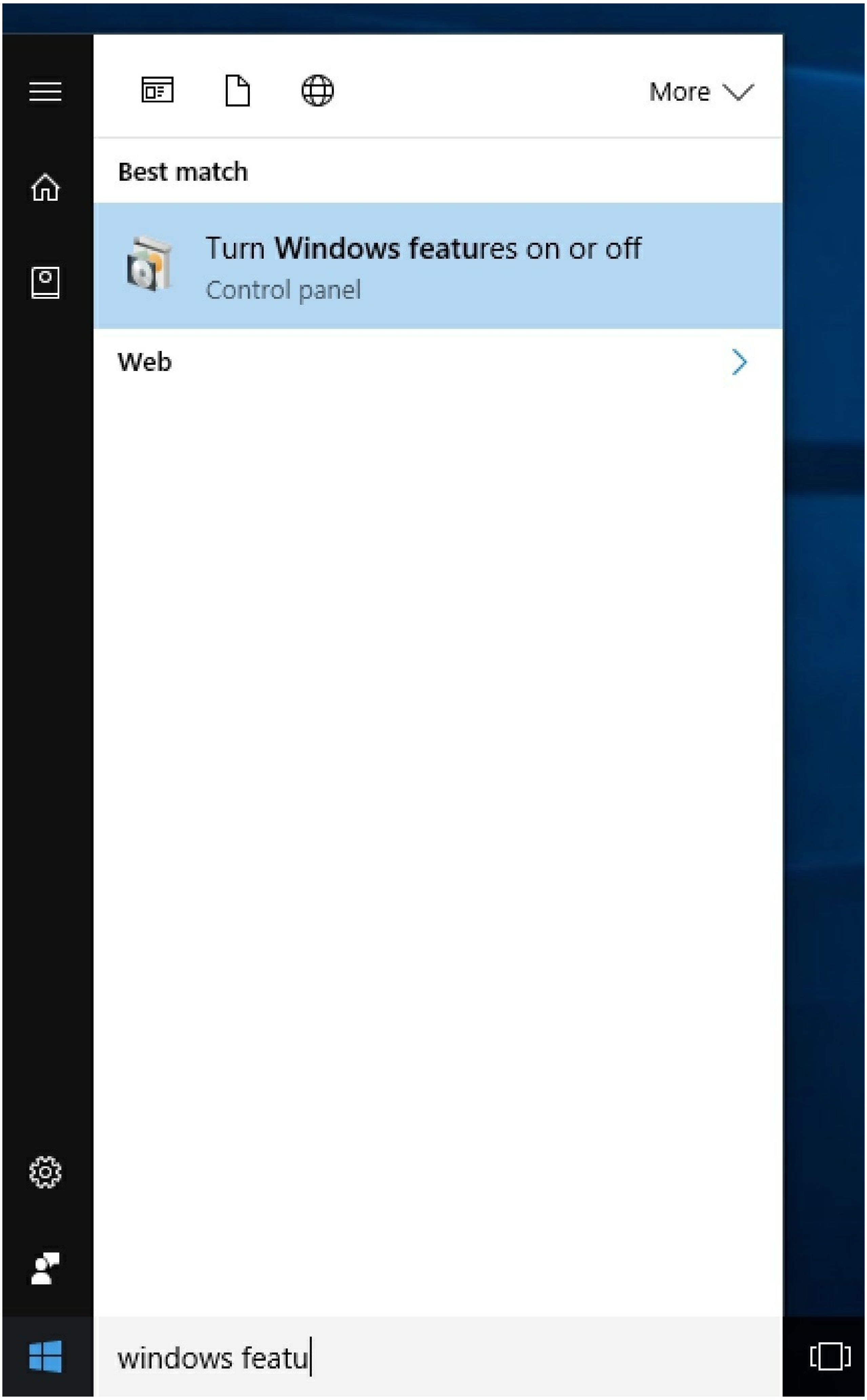
After you enable that, you open up “Windows Features”, you can find it by searching for “Turn Windows features on or off”. Then check “Windows Subsystem for Linux (Beta)”.

You may be asked to restart.

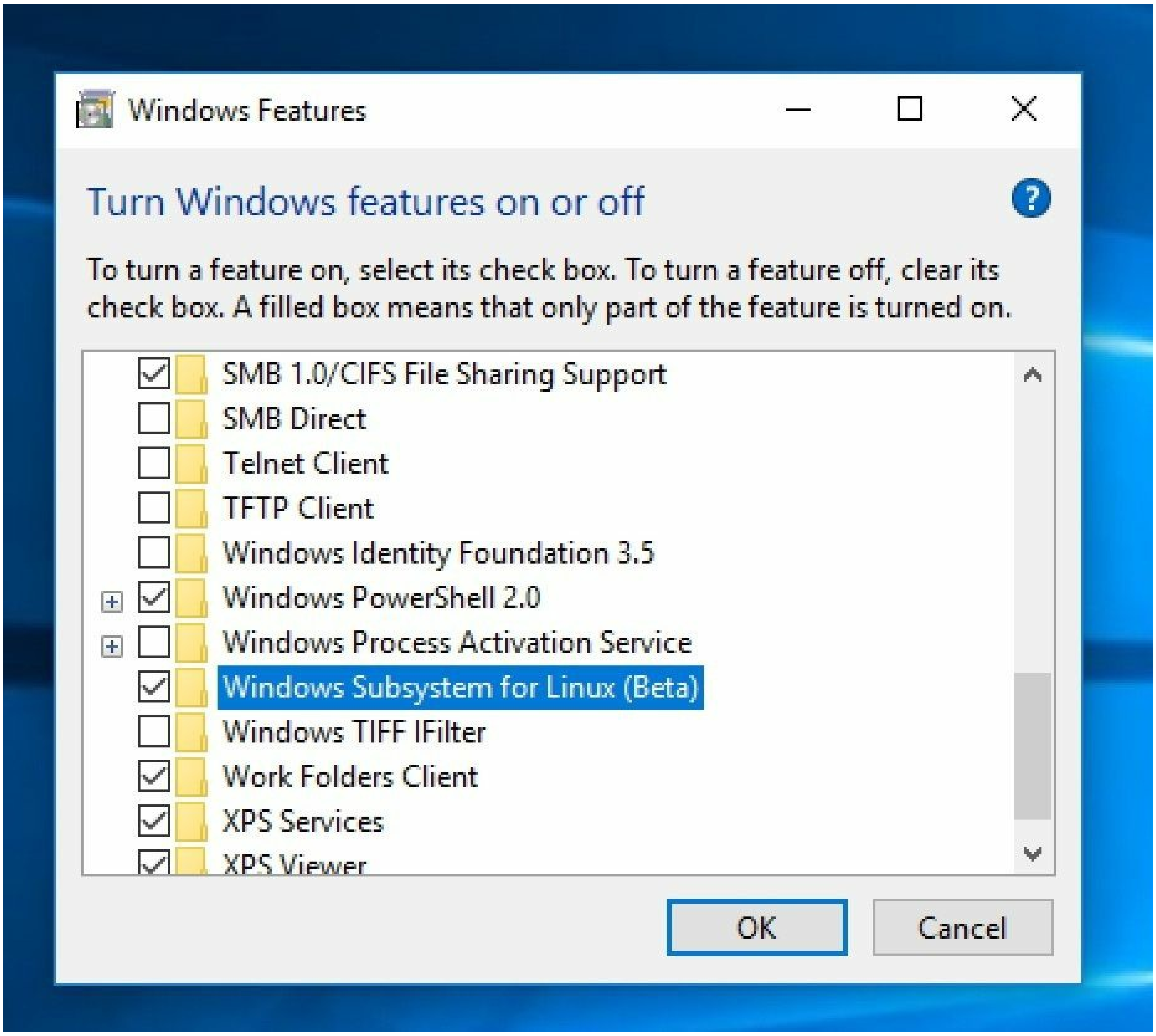
Then open up Command Prompt as you normally would (Run cli.exe). Then type

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

It will prompt you to agree to terms, create a user. In my build, tmux was already installed! But if it’s not, type `sudo apt-get install tmux`.


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

Find Turn Windows Features on or off

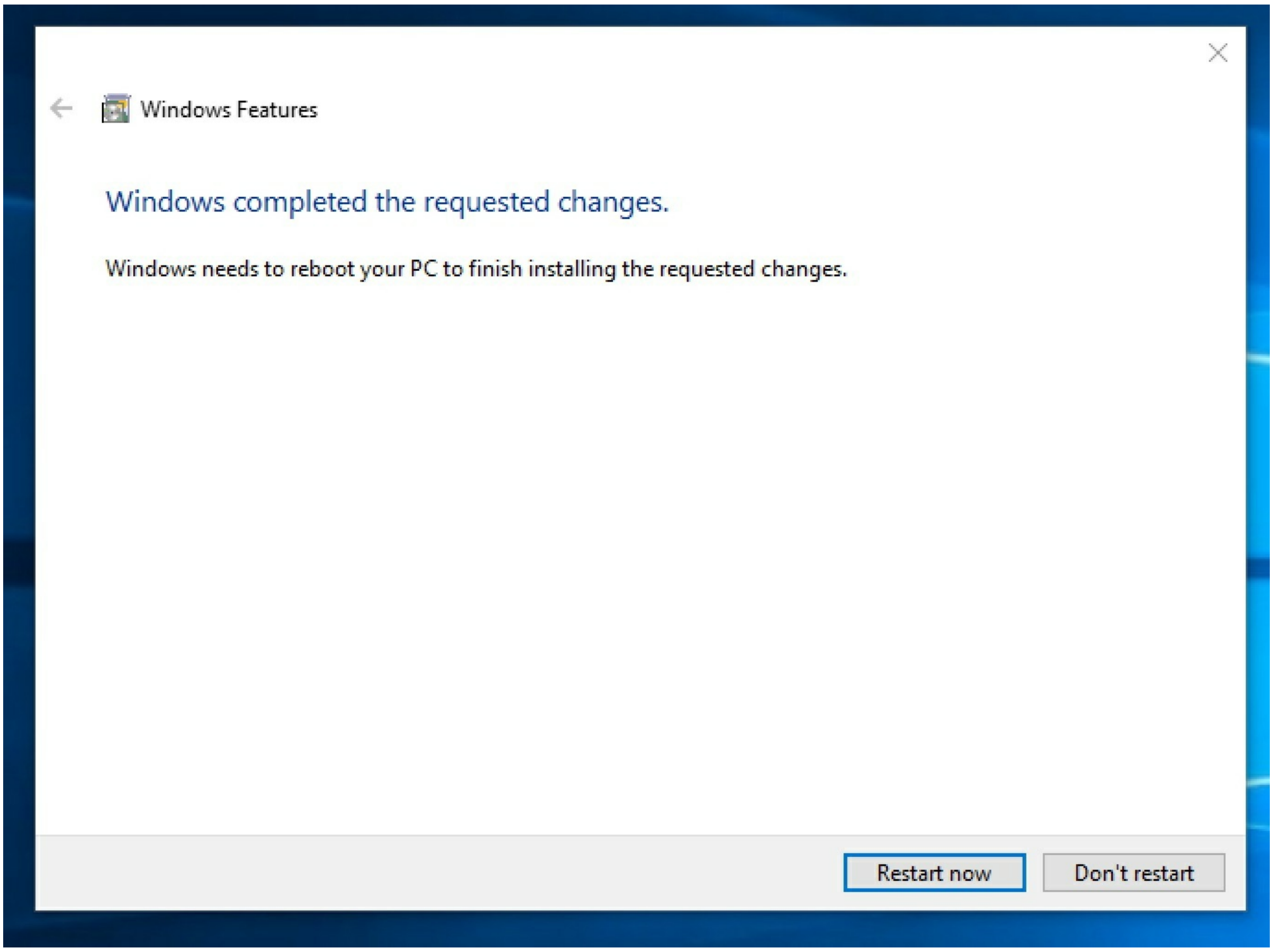

Windows Features

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

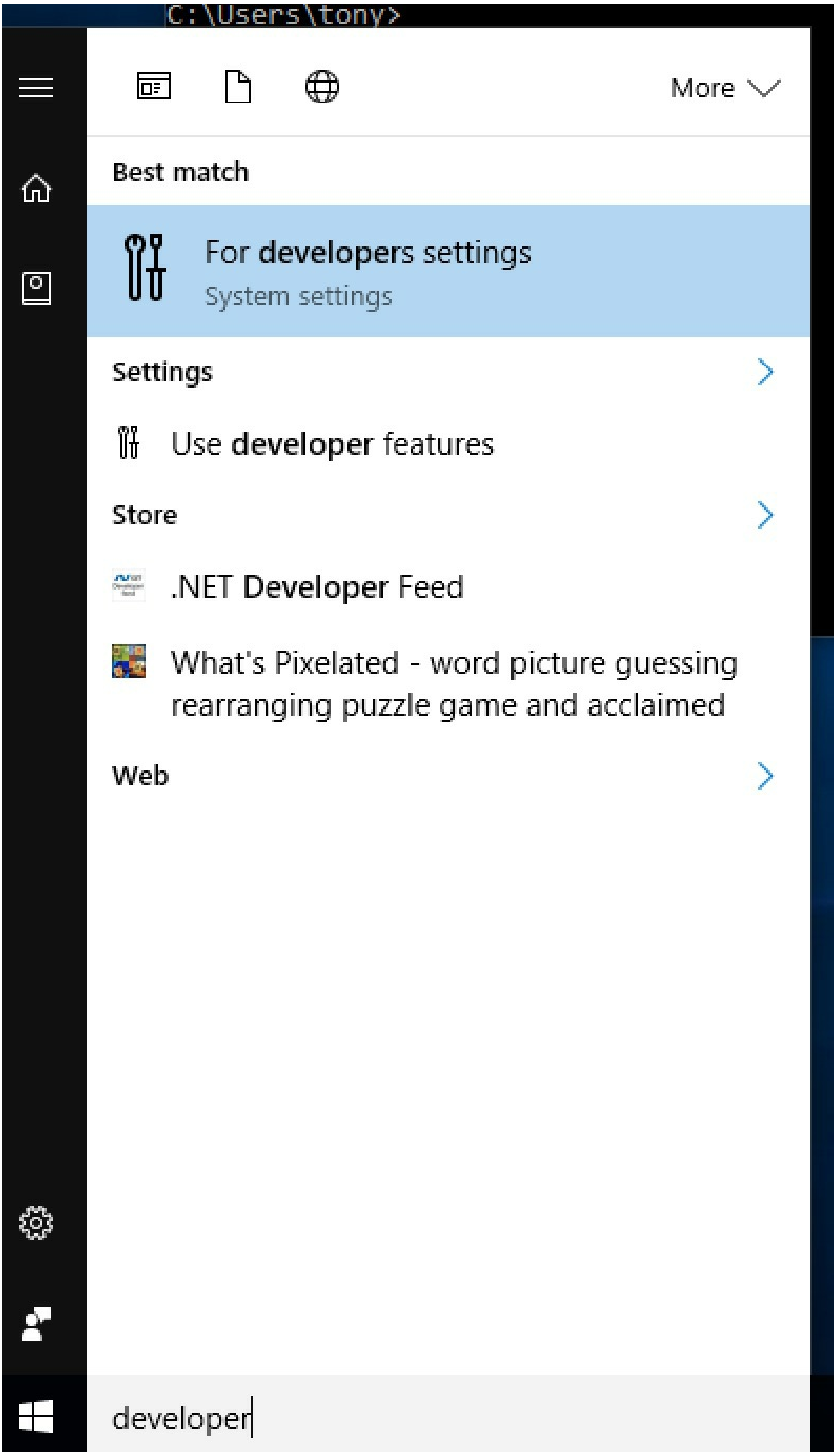
Check Windows Subsystem for Linux (Beta)


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>
```

Best match

For **developers** settings
System settings

Settings

Use **developer** features

Store

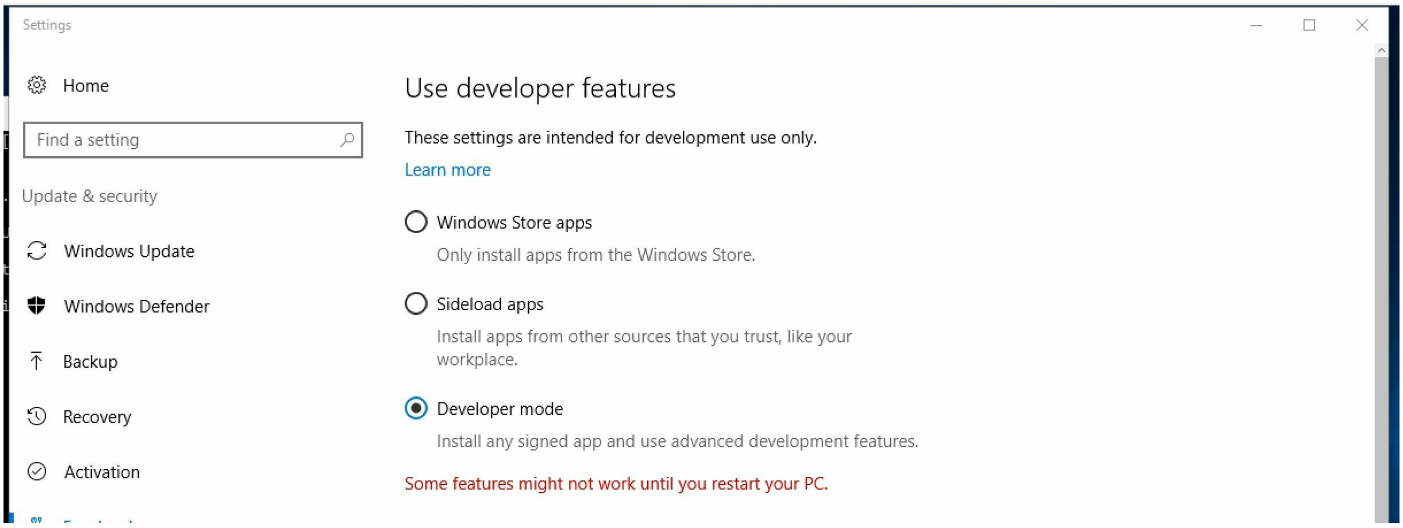
.NET **Developer** Feed

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

Web

developer

Use Developer features



Settings

Home

Find a setting

Update & security

Windows Update

Windows Defender

Backup

Recovery

Activation

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

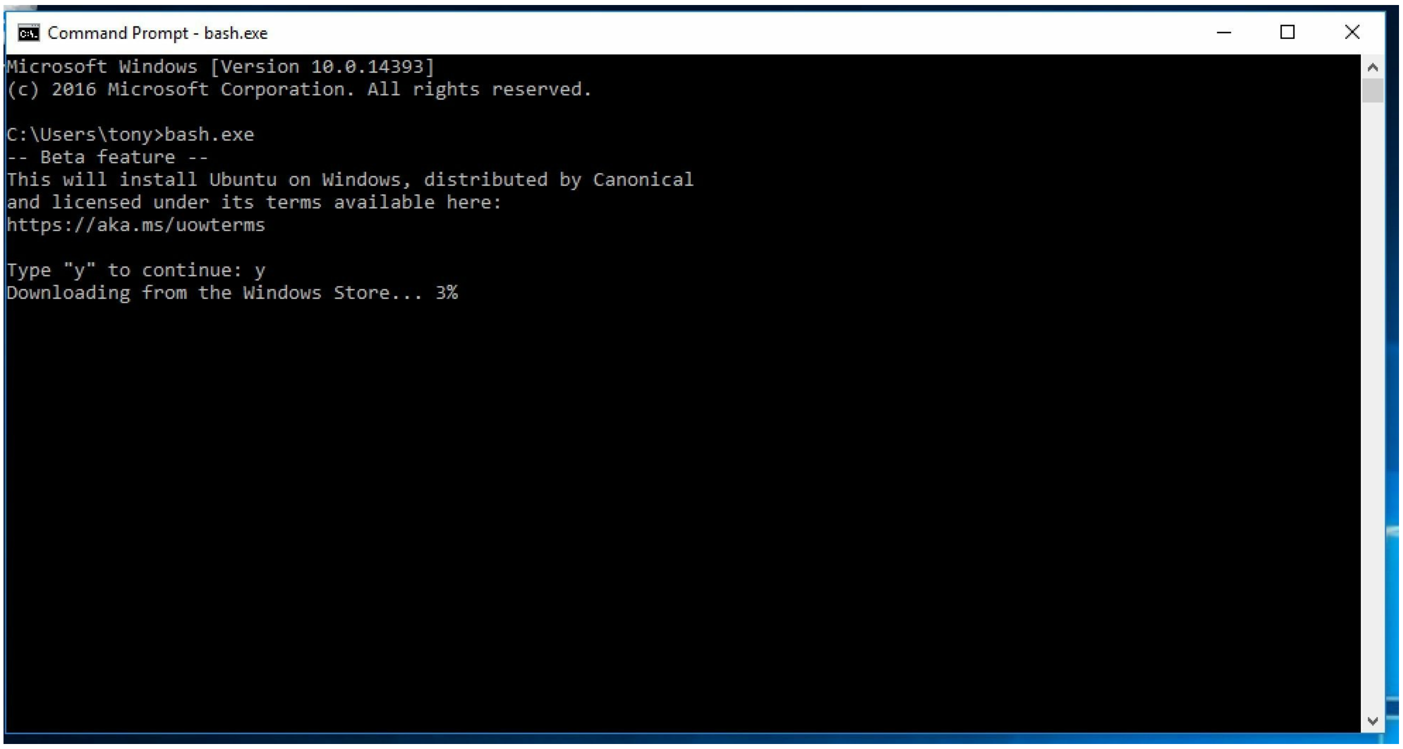
Install apps from other sources that you trust, like your workplace.

[✓ Developer mode]

Install any signed app and use advanced development features.

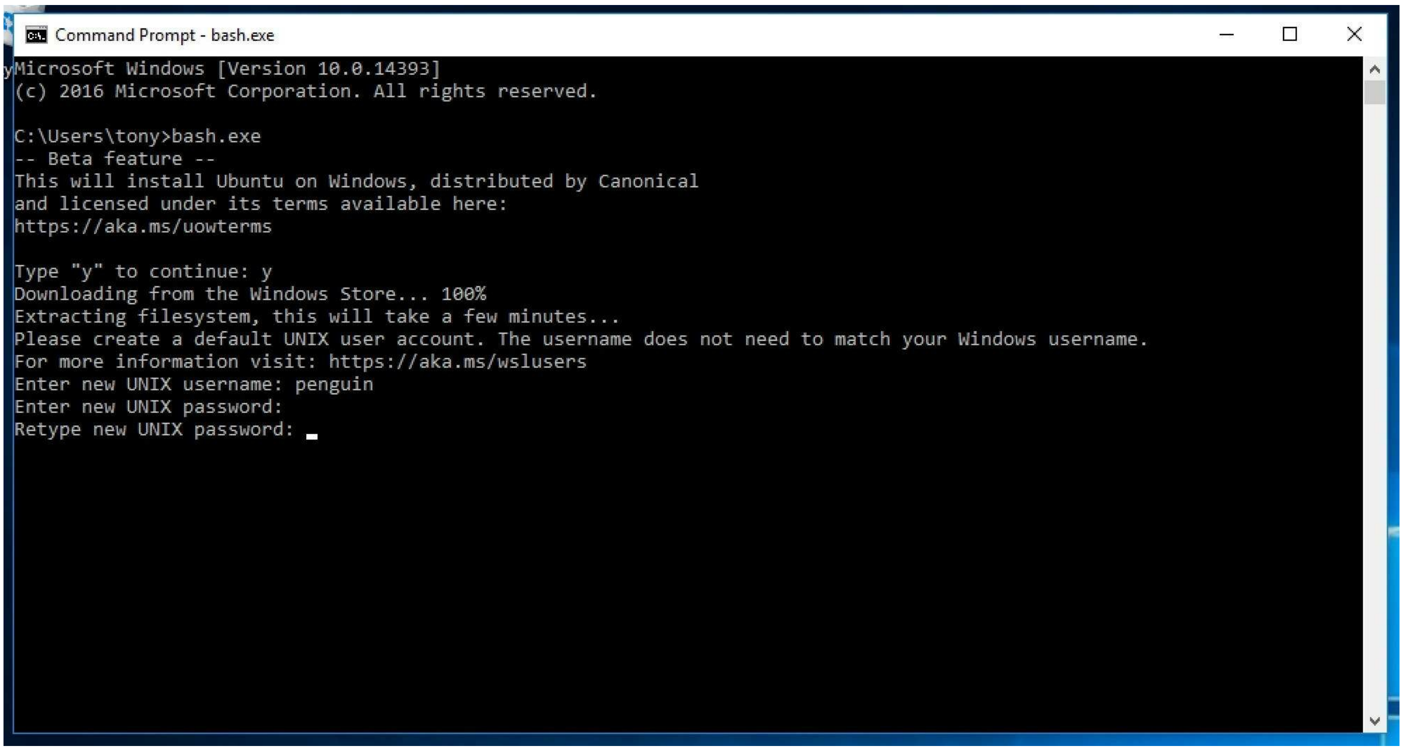
Some features might not work until you restart your PC.

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%
```

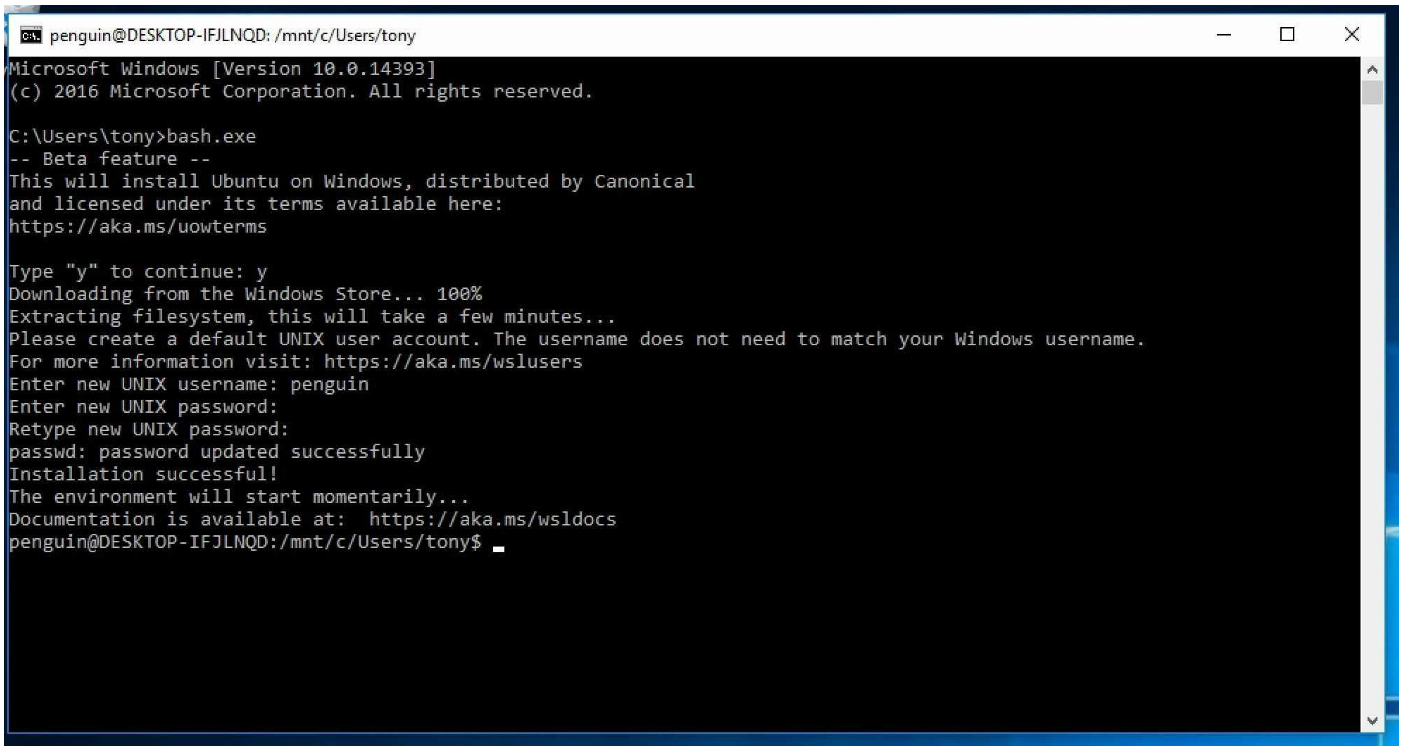
Installing Ubuntu from Windows Store



Command Prompt - bash.exe

```
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:
```

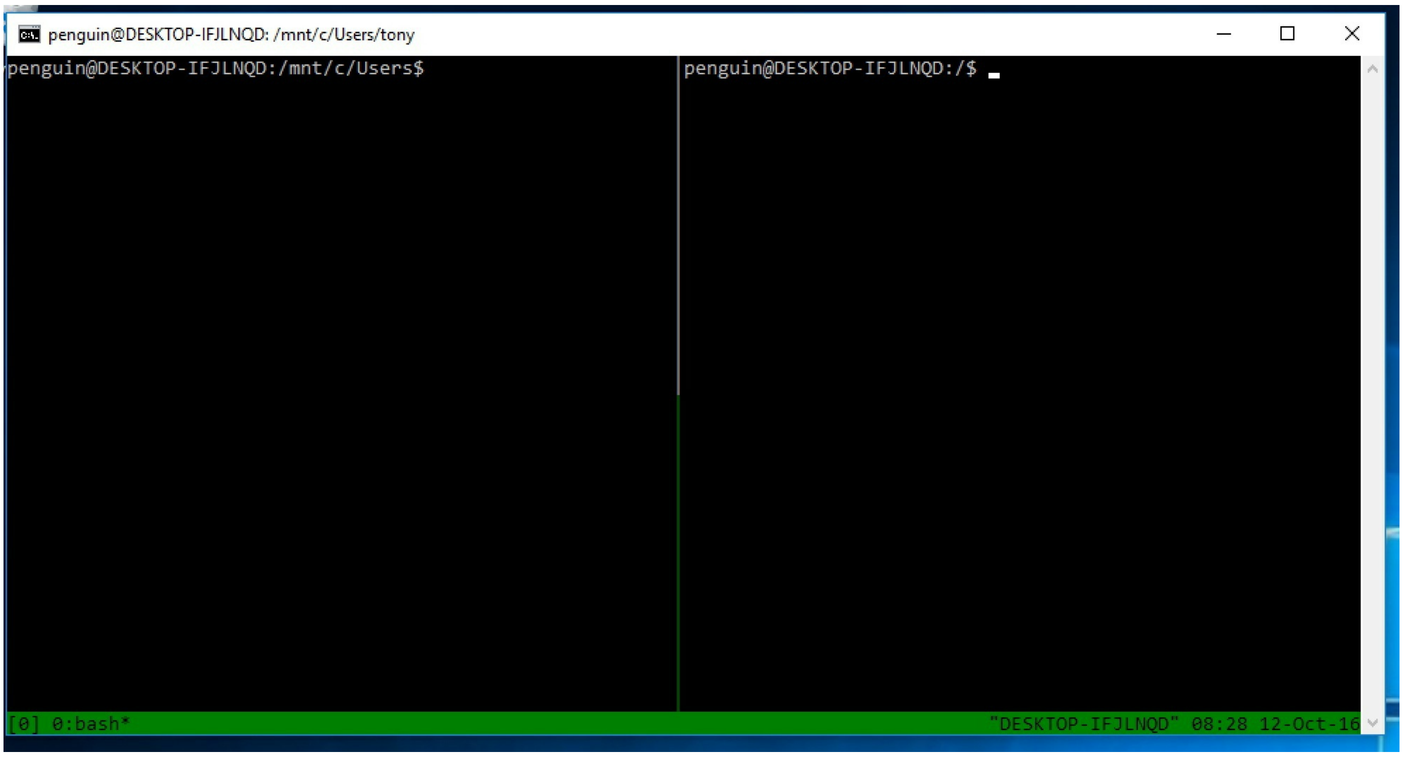
Create Linux user



```
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$
```

```
penguin@DESKTOP-IFJLNQD:/$
```

```
[0] 0:bash*                                   "DESKTOP-IFJLNQD" 08:28 12-Oct-16
```

In tmux!

This should allow you to run tmux within bash.exe.

This is a real ubuntu installation, so you can continue to install packages via `sudo apt-get install **packagename**` and update packages via `sudo apt-get update && sudo apt-get upgrade`.

Troubleshooting

`E353: Nothing in register *` when pasting on vim

If you are using macOS / OS X with vim inside tmux, you may get the error `E353: Nothing in register *` when trying to paste.

Try installing `reattach-to-user-namespace` via brew.

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

`tmuxp: command not found` and `powerline: command not found`

This is due to your site package bin path (where application entry points are installed to) not being in your paths. To find your user site packages base directory:

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

This will get you something like `/Users/me/Library/Python/2.7` on macOS with Python 2.7 or `/home/me/.local` on Linux/BSD boxes.

The applications are in the `bin/` folder inside that. So you need to concatenate the two and add them to your `PATH`. Try adding one of these in your `~/.bashrc` or `~/.zshrc`:

```
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
```

Then open a new terminal, or `. ~/.zshrc` / `. ~/.bashrc` in your current one. Then you can run `$ tmuxp -V`, `$ tmuxp load` and `$ powerline tmux right` commands.

Table of Contents

Foreword 7

About this book 7

Styles 8

How this book is structured 8

Donations 9

Available Formats 10

Errata 10

Thanks 10

Book Updates and tmux changes 10

1. Thinking in tmux 11

Window manager for the terminal 11

Multitasking 12

Keep your applications running in the background 14

Powerful combos 15

2. Terminal fundamentals 18

POSIX stuff 18

Terminal interface 19

Terminal emulators 20

Shell languages 20

Shell interpreters (Shells) 20

It’s about context 21

3. Practical Usage 22

The prefix key 22

Session persistence and the server model 23

It’s all commands 24

4. Server 25

What? tmux is a server? 26

Zero config needed 26

Stayin’ alive 27

Servers hold sessions 27

How servers are “named” 28

Clients 28

Clipboard 28

5. Sessions 30

Creating a session 32
Switching sessions within tmux 32
Naming sessions 33
Does my session exist? 34
6. Windows 35
Naming windows 35
Traversing Windows 36
Moving Windows 36
Layouts 37
Closing windows 39
7. Panes 40
Creating new panes 40
Traversing Panes 42
Zoom in 42
Resizing panes 42
Outputting pane to a file 43
8. Configuration 44
Updating configs in current sessions 44
How configs work 45
Common options 45
9. Status bar 47
The symbology behind windows 47
Date and time 47
Shell command output 48
Styling 48
Tweaking your status bar, live! 48
Turn your status line off 48
Example: default config 49
Example: Dressed up 50
Example: Powerline 51
10. Scripting tmux 53
Aliases 53
Pattern matching 54
Targets 55
Formats 56
11. Tips and Tricks 59
Read the tmux manual in style 59

Log tailing 60
File watching 60
Session Managers 62
More code and examples 63
tmux-plugins and tpm 64
12. Cheatsheets 65
Commands 65
Keybindings 67
Formats 71
Appendix: Installing tmux 75
macOS / OS X 75
Linux 75
BSD 75
Windows 10 76
Appendix: tmux on Windows 10 77
Troubleshooting 87
E353: Nothing in register * when pasting on vim 87
tmuxp: command not found and powerline: command not found 87