Getting Started With tmux

A terminal multiplexer

Wolf 14 September 2021

Agenda

- Why `tmux`
- `tmux`'s architecture
- A little about configuration
- Some important commands (of the many)
- Servers, clients, sessions, windows, and panes
- The status line
- A little about automation
- Modes, copy, and paste
- Pair-programming

There are a lot of live demos in this presentation, unfortunately the first one doesn't happen until slide 14 because I have some background to cover first. At the end, we'll use tmux to answer your "show me" questions: a live demo of whatever you want. And if I don't know how to do it, we'll figure it out together.

You **are** going to want a copy of the slide-deck. For long lists of things like pane tokens and the like, I'm not going to go into depth or just read the list. So if you want to go over them, get the deck.

```
man tmux
```

...and the most important thing I can teach you! It's the most complete, accurate, and up-to-date reference for `tmux`.

Who Am I?

- Wolf
- Long-time software engineer
- I've been using `tmux` for at least five years, not really sure
- I wouldn't call myself a `tmux` expert, rather I continue to be a student of `tmux`. This means there will be questions to which I don't know the answer
- I'm a strong fan of automation. That colors this presentation. It's one of the reasons why I use `tmux` instead of Gnu `screen`
- I have no problem with Emacs; but I also have almost no knowledge of Emacs. Vim is typically my editor of choice.That also colors this presentation

For those on YouTube, yes Wolf is my entire name.

Don't let my particular choice of editors stop you from checking out `tmux`. It's great for any editor.

Note: I'm typing on a crazy new keyboard layout that may slow me down a bit. I love my keyboard but I'm still adapting to my new layout.

`tmux` Background

What Is `tmux`?

And why do I need it?

- `tmux` is a terminal multiplexer, that is
  - It, itself, runs in a terminal
  - It manages multiple sub-terminals within itself (yes, like Gnu `screen`)
- You use it to
  - Protect running programs on remote servers from disconnects
  - Allow remote programs to be accessed by different local computers, even simultaneously
  - Bring programs and shells together into a single workspace, something like an IDE or a window manager
  - Facilitate pair-programming

Where Did `tmux` Come From?

- Created by Nicholas Marriott starting in 2007
- A heavy Gnu `screen` user at the time
- Who wanted something better. He felt
  - Gnu `screen` had a lot of baggage
  - Poor docs
  - Strange configuration; and an
  - Unintuitive command-line

How Is `tmux` Better Than Gnu `screen`?

- `tmux` is truly client/server. Gnu `screen` just emulates this
- `tmux` supports both Emacs and Vim key-bindings
- `tmux` has much better scripting support
- Window/pane splitting in `tmux` is more advanced (and lives through detaching/reattaching)
- `tmux` provides automatic window renaming
- `tmux` has a much richer set of commands and options
- `tmux` has better Unicode handling
- `tmux` is under active development: 6 month release cycle

Can Gnu `screen` Do Things `tmux` Can't?

- Gnu `screen` can directly make serial connections, which these days you might use to connect to your microcontroller board, e.g., an Adafruit Circuit Playground Express; however, you **could** just use `cu` for this, also by Nicholas Marriott. You can even run `cu` from within `tmux`
- Gnu `screen` has a single server per session. If you use multiple sessions and server stability has been a problem for you, this added isolation might be helpful
- But the bottom line is that `tmux` and Gnu `screen` are, very roughly, parallel in major features

Why you would use a terminal-multiplexer to do serial communication **instead** of using it to multiplex terminals is a mystery to me.

Gnu `screen` is considered by many to be “done”. That’s why it’s not really under active development. Pros of this are that **maybe** `screen` has fewer bugs. Cons are that while `tmux` got to learn what works and what doesn't for `screen`, `screen` didn't get to learn anything from `tmux`.

Why `tmux` Instead Of Just My GUI Terminal?

…I already have windows, tabs, splits, and a working mouse

- See the four points of my earlier slide:
  - Protect running programs on remote servers from disconnects
  - Allow remote programs to be accessed by different local computers, even simultaneously
  - Bring programs and shells together into a single workspace, something like an IDE or a window manager
  - Facilitate pair-programming
- Plus you're usually running `tmux` remotely over `ssh`
- `tmux` has better scripting than your GUI terminal
- You're faster when you keep your hands on the keyboard (at least that's what they tell me)
- …and you're not giving up your GUI. You're running `tmux` inside it

Availability & Installation

- An OpenBSD project, but the portable version is available for pretty much all Linuxes, Unixes, macOS, and Windows
- Install
  - With your favorite package manager
  - Build from source
  - Download a binary
  - https://blog.pjsen.eu/?p=440 explains how to install in Git Bash for Windows

To Mouse Or Not To Mouse

- `tmux` is specifically designed to do **everything** without the mouse
- …but it supports the mouse for those who want it
- In my experience, the best use of the mouse is fine-grained pane-resizing

You can turn mouse support on and off in `tmux` with `set-option -g mouse [on | off]`. The mouse works in Vim inside `tmux` either way, but scrolling with the scroll-wheel in `less` (e.g., for `man tmux`) only works with the mouse turned off.

`tmux` In Action

Live Demo

- Open a terminal and ssh into a server
- `new-session` (now we're running `tmux` **on the server**)
- Run `top`
- Split the window using a shortcut
- Split the window again using `split-window` at the command prompt
- Close the outer, local, terminal (simulating disconnect)
- Open a new terminal and reconnect
- `list-sessions`
- `attach-session` to the correct one, see `top` is still running

Note: a `tmux` server (and therefor, your sessions) will not survive a reboot. There is a plugin that helps with that, but obviously there's only so much it can do.

A Little About Commands

- Command names are pretty descriptive, but they all have shorter forms; and any unique string of starting characters can be used
- Commands can be used:
  - From any terminal, outside or inside `tmux` (we'll learn more about targeting existing sessions, windows, and panes a little later)
  - At `tmux`'s internal command-prompt
  - Within `tmux` configuration files and scripts
- Here are the commands we used:

`new-session`

- `-s` lets you provide a name for the new session; otherwise it just gets an automatically-assigned number
- `-n` lets you name the initial window in the session
- You are attached to the session immediately unless you specified `-d` (d for detached, daemon, or “don’t select” maybe?)
- If used from `tmux`'s command prompt, switches to the new session (does not nest, although `tmux` can be made to nest if that's what you want)

`split-window`

- Abbreviations: `split` or `splitw`. Alias: `split-pane` or `splitp`
- I disagree with the name. It targets (with `-t`) and operates on **panes**
- `-v` arranges the two resulting panes one above the other (shortcut: prefix ")
- `-h` side-by-side (shortcut: prefix %)
- Note this is the opposite of Vim, and I'm starting to appreciate it; but the shortcuts are terrible. We'll give it better shortcuts ahead
- Also read: https://gist.github.com/sdondley/b01cc5bb1169c8c83401e438a652b84e. Super Guide to the `split-window tmux` Subcommand (and Beyond)

The directionality of “vertical” and “horizontal” for split-window are confusing to some people, but they are not going to change. See https://github.com/tmux/tmux/issues/213. You'll get used to it.

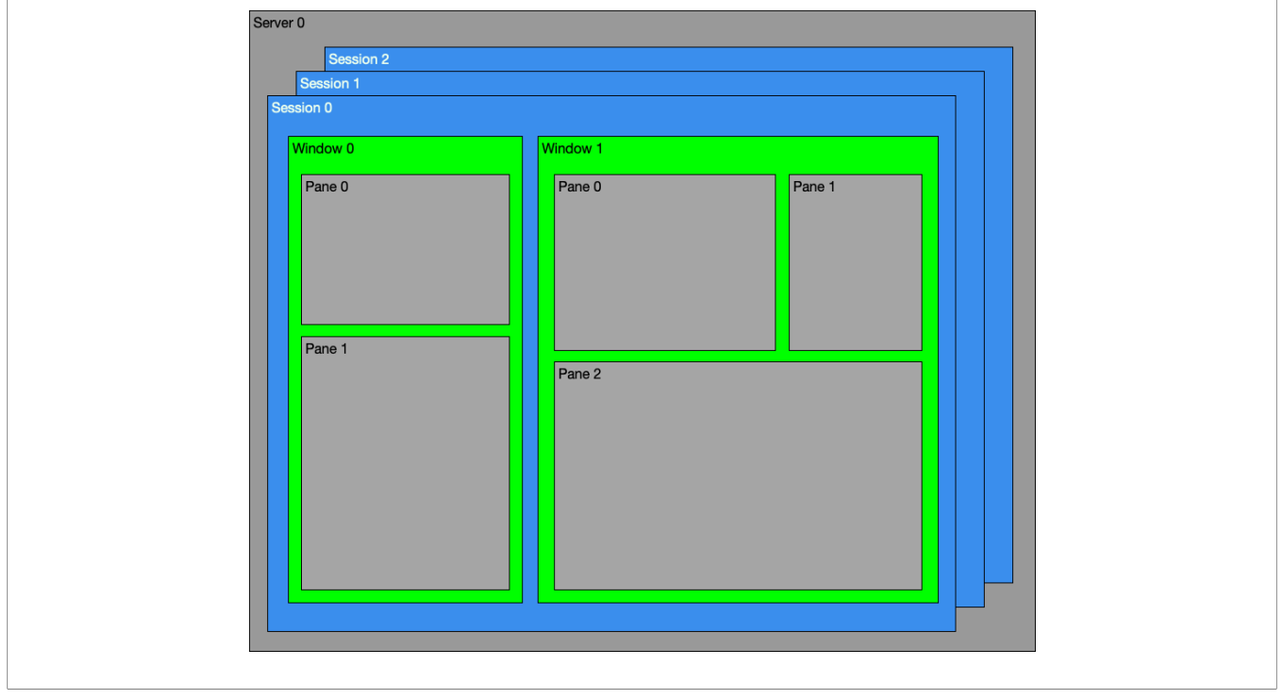
`list-sessions`

- Most-used abbreviation: `ls`
- From inside `tmux`, consider the alternative `choose-session`

`attach-session`

- Abbreviation: `attach`
- `-t` targets a specific session, else the last active session that is not currently attached to any client is chosen
- If used from `tmux`'s command prompt, switches to the targeted session. Does not try to nest
- But if you **are** inside `tmux`, again, consider the alternative `choose-session`

The Architecture Of `tmux`



Thanks to Jim McQuillan for this diagram!

Servers

- A server owns one or more sessions
- If all the sessions quit, the server quits
- If the server is killed, all its sessions are killed
- A server communicates with clients over a socket. A server listens for client connections on exactly one socket. If you use a different socket, you get a different server

Clients

- A client is running in a terminal app such as xterm, iTerm2, or even a pane in `tmux` itself (if you're running nested `tmux` sessions)
- A client is connected to exactly one session
- If the client is killed, the session is detached and goes on living in the background
- If the session is detached or killed, the client is killed
- Some commands take a `-t` *target-client* option. Clients are named for their underlying device, e.g. (on my system), `/dev/ttys005`, `/dev/ttys009`, or just `ttys006`. Names on your system will vary.

You can see client names by saying `choose-client` or `list-clients`

Sessions

- Sessions are the entry-point into `tmux`. You start your `tmux` day with `new-session`, `attach-session`, and/or `list-sessions`
- A session manages one or more windows
- It is attached to zero or more clients
- It belongs to exactly one server and cannot be moved to a different server
- If you start a session and no server is running, a server will be created

Windows

- A window contains one or more panes
- Windows show up in the `tmux` status line as "tabs"
- When the last pane in a window is closed, the window closes
- If a window is killed, all the panes in it are killed
- A window is linked into one or more sessions
- A window can be moved between sessions

Panes

- A pane is a single virtual terminal
- It belongs to exactly one window
- Inside a pane is where your actual work happens

In Gnu `screen`, what `tmux` calls a pane is a "region".

A Quick Look At Configuration

`~/.tmux.conf`

- Only read on **server** creation
- A different configuration file can be selected by using the `-f` option when the server is started
- Options can be updated in a running server using the `source-file` command. Rereading the config only overwrites existing options and adds new. It doesn't magically reset anything back to original defaults
- The two most-used commands in the configuration file:
  - `set-option`
  - `bind-key`

The Prefix Key

- Starts almost all tmux shortcuts, hence the name “the prefix key”
- Lets `tmux` know the next key is meant for `tmux` itself, not the app in the active pane
- Defaults to Control-b. Abbreviated C-b
- Can be changed in `.tmux.conf`. Many people use C-a instead, because that’s what Gnu `screen` uses (and it’s easier to reach)
- Hit twice to send it past `tmux` and into the app in the current pane
- Can be changed live at `tmux`’s command-prompt in case you are running nested `tmux` instances and don’t want to get confused

`tmux`'s Command Prompt

- Prefix :
- Execute any `tmux` command
- Uses Emacs-like key-bindings unless your `$EDITOR` contains the string "vi" or you explicitly set the `status-keys` option
- Get out of it by either executing a command, or executing an empty command with Return. With Emacs bindings, Escape. With Vi bindings, Escape to get into Vi "normal" mode, then q

Live Demo

- Splitting the window from the command-prompt
- Changing the prefix in .tmux.conf
- Sourcing the config
- Auto-complete at the command-prompt (by starting with `cho`)

`set-option`

- Abbreviation: `set`
- Options are set on the server, session, window, pane, or else "globally" from which they are inherited. For most options, `tmux` knows where to set it; but you can specify in the call to `set-option`
- `-a` lets you append text to an already set option
- Opposite: `unset-option`
- Show an (or all) option(s): `show-options`
- Interactively view and change options: `customize-mode` (shortcut: prefix `C`). This will also show you where an option belongs: server, session, window, or pane

The `customize-mode` command puts you into the options-mode mode. I know. That's weird. But also, I love options-mode. It's great for exploring.

`bind-key`

- Abbreviation: `bind`
- Opposite: `unbind-key`
- Show a (or all) key bindings: `list-keys`
- Use prefix `?` to show the current key bindings

No I'm not going to talk about key tables today. They're a more advanced topic and out-of-scope for this talk.

13 Commands: 90% of the Work

Some Important Commands

A few for outside of `tmux`. A few for inside.

Shortcut	Action	Actual Command
	Create a new session, window, & pane	`new-session -s name`
	List existing sessions	`list-sessions`
	Attach to existing session	`attach-session -t name`
prefix ?	List key bindings	`list-keys -N`
prefix d	Detach the current client	`detach-client`
prefix c	Create a new window in this session	`new-window`
prefix n	Move to the next window	`next-window`
prefix “	Split the current pane “vertically"	`split-window -v`
prefix %	Split the current pane “horizontally”	`split-window -h`
prefix o	Move to the next pane	`select-pane -t :.+`
prefix [	Enter copy mode	`copy-mode`
prefix]	Paste the most recent buffer	`paste-buffer -p`
prefix :	Enter the command-prompt	`command-prompt`

Just knowing these commands is enough to get around tmux like a pro.

Sessions In Depth

Targeting A Session

- Every session has a unique (server-wide) unchanging session ID. If you ever have to use it, it is prefixed with `$`, e.g., `$0`
- If you didn't supply a name when you created a session, it automatically gets a number for its name. Don't confuse this with its ID
- Several commands take a `-t` *target-session*. You can supply a session ID, a session name, or a pattern to be matched against the session name (in that order of preference)

Unlike with clients, neither `list-sessions` nor `choose-session` display the session ids. If you want to see one, you need to say `display-message "#{session_id}"`.

Some Session-Specific Commands

- `attach-session`
- `has-session`
- `kill-session`
- `list-sessions`
- `choose-session`
- `lock-session`
- `new-session`
- `rename-session`

This is not all of them. Consult `man tmux`.

Some Session Details

- A session may be connected to more than one client at a time; but every client shows the same active window (there's a way around this. I'll show you in the next slide)

Session Groups

- Why: you want multiple clients to work on the same windows/panes, but independently. You don't want them to share the same active window as they would if they were working on the same session
- What: a session group is a collection of sessions all of which contain the same windows and panes, but each of which has its own active window and active pane (and session ID, and session name)
- How: `new-session -t` *group-name*. You can supply the name of a new group; an existing group; or a session name, in which case you'll add your new session to the named session's group. If the named session has no group, a new one will be created, named after that session
- But: it's not magic. No matter how many sessions contain a pane, there's only one process in that pane, e.g., if you're running Vim in that pane, you won't have two different cursors. Sessions in a group can have different active **windows**, but if they have same active window, they will have the same active pane.

Sorry for the wall of text. This slide will be more useful to you when you have the slide-deck in your hands and are going over it slide-by-slide.

Windows In Depth

Targeting A Window

- Every window has a unique (server-wide) unchanging window ID. If you ever have to use it, it is prefixed by an `@`, e.g., `@1`
- Every window has an index within its session. That index can change if you reorder the windows. Window indexes start at 0 unless you specifically set the `base-index` option
- Several commands take a `-t` *target-window*. You can supply a special token (more on this ahead); an index; a window ID; an exact window name; or a pattern to match against the window name (in that order of preference)
- When targeting a window, you can start with the session if needed, and separate with a `:`, e.g. `mysession:@2`, `$6:Editor`, `:bash`
- If you didn't supply a name for the window, it will be dynamically assigned based on what's currently running in its active pane. That's the window's "title" by the way

You can see the window-id with `list-windows`. Unfortunately, `choose-window` **doesn't** show it.

Tokens Identifying Windows

- {start} or ^
- {end} or $
- {last} or !
- {next} or +
- {previous} or -

Some Window-Specific Commands

- `new-window`
- `rename-window`
- `select-window`
- `swap-window`
- `move-window`
- `link-window`
- `kill-window`
- `find-window`
- `list-windows`
- `choose-window`

There are plenty more. Again, consult `man tmux`.

Some Window Details

- A window may be linked into more than one session at a time. In fact, that's how session groups work
- A window may be moved from one session to another
- A window always contains at least one pane
- A window with multiple panes can cycle through some preset layouts

Live Demo By Special Request

Moving a window from one session to another

- Go to the iTerm window already showing two different sessions side-by-side
- Explain that this is an iTerm split, not a tmux split, and why: clients only show one session at a time and I needed two clients here
- Show the windows in the left session. Have only one window in the right session
- Use move-window from the tmux command-prompt
- Dissect the result

Panes In Depth

Targeting A Pane

- Every pane has a unique (server-wide) unchanging pane ID. If you ever have to use it, it is prefixed by a %, e.g., `%9`
- Every pane has an index within its window. That index changes as you reorder the panes. The lowest index is the top left. The highest index is the bottom right. Pane indexes start at 0 unless you specifically set `pane-base-index` option
- Several commands take a `-t` *target-pane*. You can supply a special token (more ahead); an index; or a pane ID
- When targeting a pane, you can start with the session and/or window if needed, and separate from them with a `.`, e.g. `mysession:@2.%2`, or `$6:Editor.!`, `:.+`
- Panes don’t have names, but they do have "titles"

Tokens Identifying Panes

- `{last}` or `!`
- `{next}` or `+` (can take an argument)
- `{previous}` or `-` (can take an argument)
- `{top}`
- `{bottom}`
- `{left}`
- `{right}`
- `{top-left}`
- `{top-right}`
- `{bottom-left}`
- `{bottom-right}`
- `{up-of}`
- `{down-of}`
- `{left-of}`
- `{right-of}`

Go back to slide 35 for a moment to analyze the command to go to the next pane.

Some Pane-Specific Commands

- `split-window` (also known as `split-pane`)
- `join-pane`
- `list-panes`
- `capture-pane`
- `pipe-pane`
- `swap-pane`
- `resize-pane` (especially with `-Z` for zoom)

There are plenty more.

Some Pane Details

- A pane is normally just a terminal. Sometimes, though, it will be in a special “mode”: copy-mode; buffer-mode; view-mode; client-mode; tree-mode; or option-mode
- You can see the pane indexes with prefix `q`
- Panes may be "zoomed" (try prefix `z` to toggle)
- Panes may be moved from one window to another, even creating a new window in the process

The Status Line

Let's Play With The Status Line

- The status line comprises three parts: stuff on the left, the window list in the middle, and stuff on the right
- Controlled by the options `status-left` and `status-right`
- You can run code to produce text in the status bar by enclosing it in `#()`, for example: `#(uptime)`
- There are options to control the maximum length of the left and right sides, the styles of all three parts, the refresh interval, whether the status line is at the top or bottom, and many more

Formats And Format Variables

- Formats are tmux's (rather minimal) templating language
- Formats are typically given to commands with the `-F` flag
- They are surrounded by double quotes (don't omit them!)
- They are built from format variables and options surrounded by `#{}`.
- For instance, here's a call to `list-windows` with a format:

```
tmux list-windows -F "#{window_id} #{window_name}"
```

Let's give that command a try.

Some Conditional Configuration

- `"#{?`*variable*`,`*string1*`,`*string2*`}"` produces string1 if variable is, for example

  ```
  "#{?automatic-rename,yes,no}"
  ```

- conditionals with "#{operator:string1,string2}", for example (from my config)

  ```
  %if "#{==:#{host},ubuntu}" ...
  ```

- conditionals with the `if-shell` command, for example

  ```
  if-shell "[[ $(name) == Darwin ]]" ...
  ```

Let's take another look at my configuration to see examples.

Modes

What Is A `tmux` Mode?

- Normally, in `tmux`, your keystrokes are all going into the app running in the active pane, be it a shell, an editor, a utility, whatever. When that pane is in one of `tmux`'s modes, though, your keystrokes are all going into `tmux` itself to control whatever `tmux` thing is happening in that pane.
- Various pane modes (at least according to the widget I added to my status bar):
  - copy-mode
  - buffer-mode
  - view-mode
  - client-mode
  - tree-mode
  - options-mode

Live Demo

- Use `list-keys` to get into view-mode; navigate around
- Use `choose-tree` to get into tree-mode; navigate around
- Use `customize-mode` to get into options-mode; navigate around and set some options

More about copy and buffer mode later

Copy & Paste

Your Favorite Key-Bindings

- Unlike other modes, copy-mode key bindings are under your control
- Emacs-like bindings is the default unless the value of `$EDITOR` contains the string "vi"
- You can explicitly control it by setting the `mode-keys` option

Note that **all** key-bindings are under your control when you're **not** in a mode.

Copy Mode

- Get into copy mode with prefix `[`
- You know you're in copy mode because of the number/number widget in the upper right-hand corner of the pane
- Maneuver around with the key bindings you've chosen
- Scrolling all the way to the bottom generally takes you out of copy mode
- Start selecting text with Space (Vi) or C-Space (Emacs)
- Copy the selected text with Enter (Vi) or M-w (Emacs)
- Go to the session, window, and pane where you want to paste and hit prefix `]`

Consult `man tmux` for all the keys.

Live demo.

Buffer Mode

- Copying adds another buffer to the stack
- Executing the command `choose-buffer`, or using prefix = puts you into buffer mode, where you select any of the previously copied buffers to be pasted into the current pane

Live demo.

Scripting

The Good (But Short) News

- Because `tmux` commands can be run from the shell and target any session, window, or pane, this is just shell scripting
- Everything you know and love about your favorite shell applies

tmuxinator

What Is `tmuxinator`?

- It's a wrapper for `tmux` that helps you easily build complicated setups: automatically creating the windows, splitting the panes, and starting the programs to assemble your development environment
- Install with your package manager, or as a ruby gem
- Where scripting is imperative, `tmuxinator` is declarative
- Uses YAML files

Live Demo

- Edit `mug-demo.yml`; talk about the pieces
- Start, stop, and restart mug-demo
- Detach and restart
- Debug
- List
- Edit a more complicated project: `mug-angular.yml`
- Start mug-angular

Sharing & Pairing

My Favorite Technique

- On the machine you will be sharing:
  - Make a special group for this, e.g., named `tmux-sharing`
  - Make a special directory, owned by that group and read/write for anyone in the group, for sharing the underlying socket, e.g., `/var/tmux-sharing`
  - Have accounts for the users who will be sharing, e.g., `wolf` and `jam`, and make them members of the group
- The users log in to the shared box, and
  - The first one, here `wolf`, creates a new server, session, and session group by saying:
    `tmux -S /var/tmux-sharing/wolf-jam new-session -s wolf -t wolf-jam-sharing`
  - The second one, here `jam`, creates a new session in that same session group by saying:
    `tmux -S /var/tmux-sharing/wolf-jam new-session -s jam -t wolf-jam-sharing`

Live Demo

- Attach to our already running the demo with:
  `tmux -S /var/tmux-sharing/wolf-jam attach-session -t wolf`
- Have Jim attach from his machine:
  `tmux -S /var/tmux-sharing/wolf-jam attach-session -t jam`
- Be typing in the window
- …while Jim creates a second window, launches Vim, and starts typing there
- Point out when the window tab appears
- When Jim says he’s got content, switch over to Jim’s window

Okay, for time's sake, these aren't exactly the commands we're using. We're just demonstrating simple sharing of a single session, not a session group.

Conclusions

What We Covered

- What `tmux` is good for
- The architecture of `tmux`
- Configuration
- Common commands
- Servers, clients, sessions, windows, and panes
- The status line
- A little automation
- Modes, copy, and paste
- Pair-programming

Where To Go From Here

Things I didn’t talk about

- Plugins
- Automatic window renaming
- Linking one window to multiple sessions
- Hooks
- Only briefly touched on "formats"
- Pop-ups and pop-up menus
- "Control mode" and `-CC`
- Command parsing vs. execution
- Key tables, and for instance using them to support multi-key commands; or your own prefix key specifically for a subset of your favorite commands

Books And Links

- The `tmux` repo: https://github.com/tmux/tmux/
- `man tmux` (absolutely the most complete and up-to-date reference)
- The `tmux` wiki: https://github.com/tmux/tmux/wiki
- "Tmux 2: Productive Mouse-Free Development" (get the 2018 update)
- "The Tao of tmux: and Terminal Tricks"
- PJSen Blog: How to run Tmux in GIT Bash on Windows https://blog.pjsen.eu/?p=440
- My `.tmux.conf`: https://github.com/wolf/dotfiles/blob/main/tmux/.tmux.conf
- Super Guide to the `split-window` tmux Subcommand (and Beyond) https://gist.github.com/sdondley/b01cc5bb1169c8c83401e438a652b84e.

Colophon

You probably don't want to know this but I always get asked

- I'm doing this presentation on a Mac and connecting to an Ubuntu server
- I'm using the open source tool KeyCastr to show on-screen what keys I'm hitting
- Locally, I'm using `tmux` version next-3.3 and `tmuxinator` 3.0.1, both installed by Homebrew
- On my server I'm running Ubuntu 20.04.3, and `tmux` 3.0a as installed by `apt`
- I am starting with my existing `tmux` config, which we will dissect during the presentation
- I'm giving this presentation in Apple's Keynote application; but I'm providing the slides (with presenter's notes) as a PDF
- My terminal app is iTerm2.
- iTerm2 is using the open source font JetBrains Mono; these slides use Andale Mono to distinguish terminal commands
- For the pair-programming demo, Jim is connecting to the Ubuntu server from Canada through his MacBook Air