

Introduction

Are you still waiting for `grep` and `find` to show what you're looking for? Can't remember `find` syntax? Is your `ls` output bland? Tired of overconfiguring your `.zshrc` with too many plugins?

Let's explore a few must-have utilities for daily workflows.

fish – Finally, a command line shell for the 90s

- Smart autocompletion & syntax highlighting
- Now rewritten in Rust: [The Fish Of Theseus](#)
- No POSIX compliance – Be free of legacy constraints

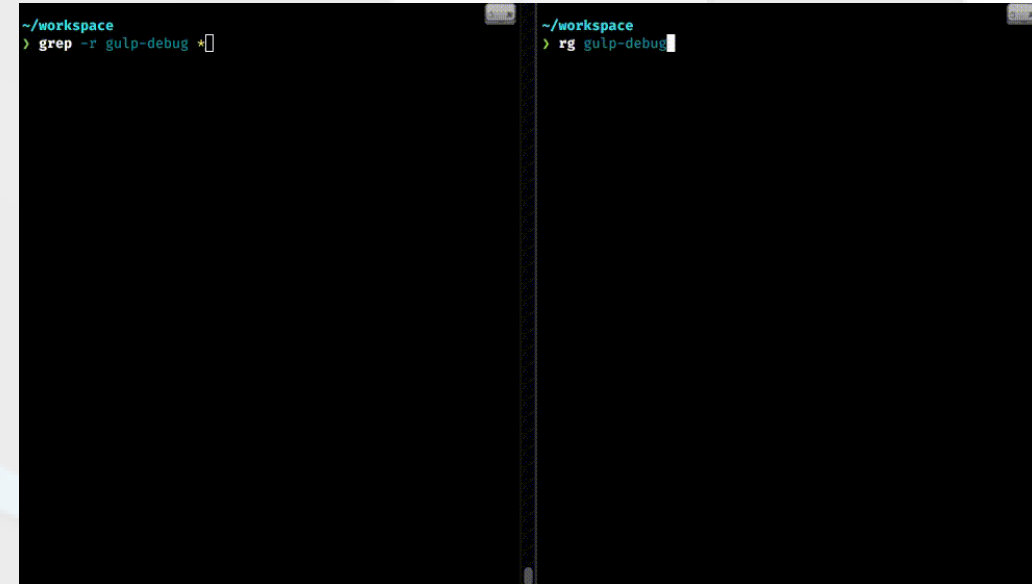
Reads Your Mind

```
fish /Users/demo (fish)
demo > cat explanation.txt
fish offers suggestions as you type, based on files, completions, and history.
Remember that long command to ssh into a host or rsync between servers? You run
it daily but never got around to making a function for it? fish will remember
it and suggest it for you again, when it sees that you've started to type it.
Just hit right arrow or control-F to accept it!
demo > ssh l demo SomeLongHost@SomeLongDomainIAalwaysMisspell.com
```

fish suggests commands as you type based on history and completions, just like a web browser. Watch out, Netscape Navigator 4.0!

ripgrep (rg) – A blazing-fast alternative to **grep**

- Fast search tool built on top of Rust's regex engine
- **.gitignore**-aware searching
- Example: **rg "error" src/** → searches recursively in source directory



The image shows two terminal windows side-by-side, both in a dark theme. The left window shows the command `grep -r gulp-debug *` being entered, with a cursor at the end. The right window shows the command `rg gulp-debug` being entered, with a cursor at the end. Both windows have a title bar that says `~/workspace`. The background of the slide features a large, faint, light green circular arrow graphic.

fdfind (fd) – A simpler alternative to `find`

- Fast due to parallelized directory traversal (and Rust, obviously)
- Smart filename matching
- Ignore patterns from `.gitignore`
- Example: `fd "config" --type f` → find config files

bat – A **cat** clone with wings

- Git integration for modified files
- Colorful syntax highlighting – your code looks so good, you won't want to run it
- Line numbering & pager support
- Example:

```
bat --style numbers file.js
```

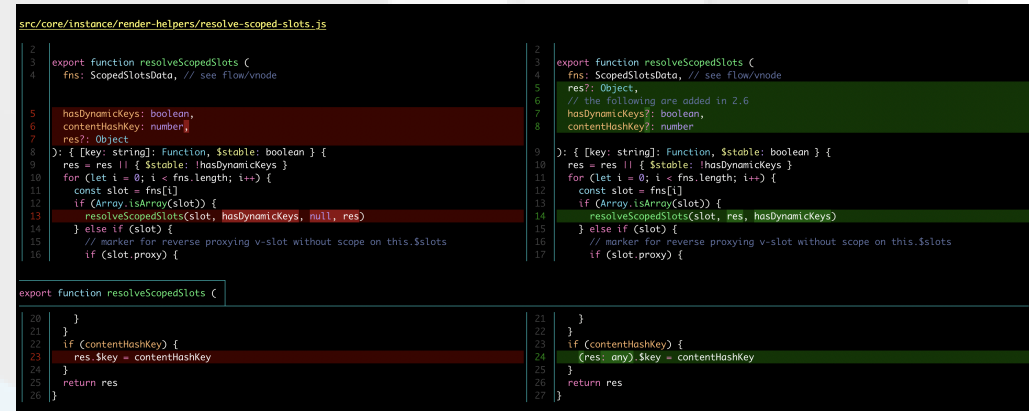
```
▶ bat src/index.html
```

	File src/index.html
1	<!DOCTYPE html>
2	<html lang="en">
3	<head>
4	<meta charset="utf-8">
5 ~	<title>a changed title</title>
6	<link rel="stylesheet" href="style.css">
7	</head>
8	<body>
9 +	New lines that have been
10 +	added since last commit.
11	</body>
12	</html>

delta – A syntax-highlighting pager for git, diff, grep, and blame output

- Beautiful syntax highlighting for git and diff output
- Side-by-side view for git diffs – perfectly blame your colleagues in PR reviews
- Customizable themes and colors

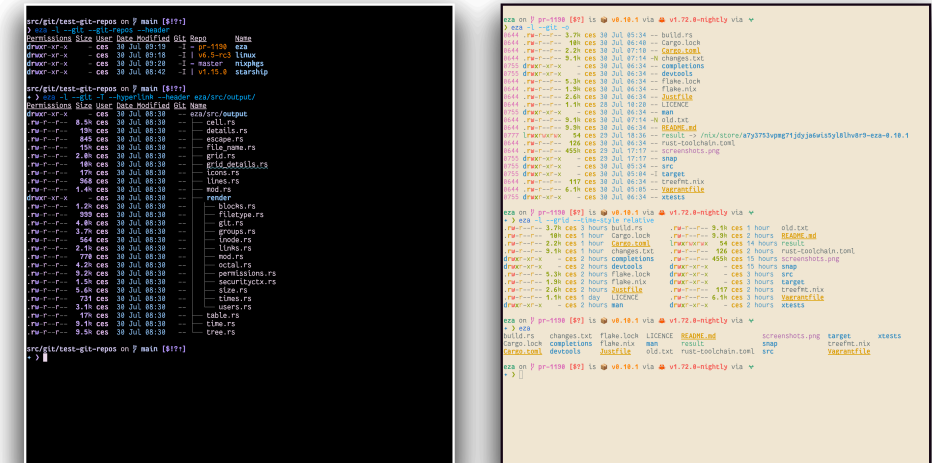
<https://github.com/dandavison/delta>



```
src/core/instance/render-helpers/resolve-scoped-slots.js
2
3 export function resolveScopedSlots (
4   fns: ScopedSlotsData, // see flow/vnode
5
6   hasDynamicKeys: boolean,
7   contentHashKey: number,
8   res?: Object
9 ): { [key: string]: Function, $stable: boolean } {
10   res = res || { $stable: hasDynamicKeys }
11   for (let i = 0; i < fns.length; i++) {
12     const slot = fns[i]
13     if (Array.isArray(slot)) {
14       resolveScopedSlots(slot, hasDynamicKeys, null, res)
15     } else if (slot) {
16       // marker for reverse proxying v-slot without scope on this.$slots
17       if (slot.proxy) {
20
21   }
22   if (contentHashKey) {
23     res.$key = contentHashKey
24   }
25   return res
26 }
```

eza – A feature-rich replacement to `ls`

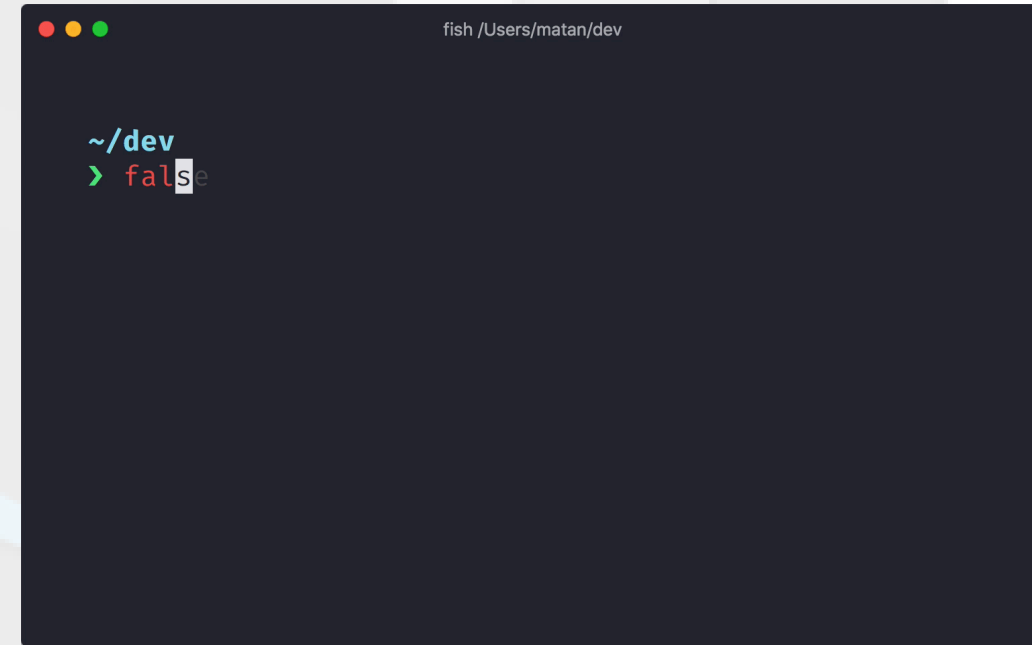
- Icons & colors to distinguish file types and metadata
- Human-readable sizes
- `eza --tree -L 2` – tree view with style and questionable performance on deep directories



The image displays three terminal windows demonstrating the capabilities of the `eza` command-line tool. The first window shows a standard `ls`-like output with columns for permissions, size, user, date, and filename. The second window shows a tree view output using `eza --tree -L 2`, displaying a hierarchical structure of files and directories. The third window shows a tree view output with icons and colors, using `eza --tree -L 2 --icons --color`, which visually distinguishes file types and metadata.

starship – The power of Rust in your prompt

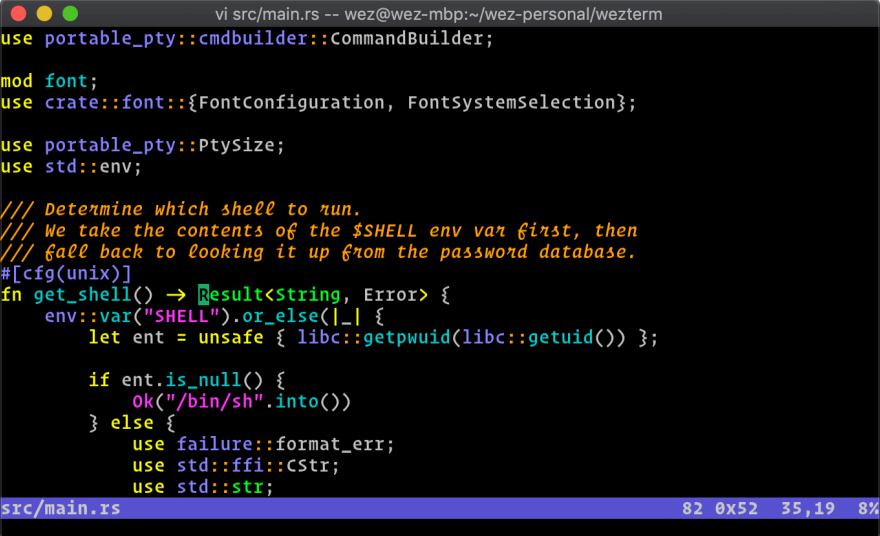
- Works with any shell (bash, zsh, fish, and even cmd.exe??)
- Shows information you need (git status, language runtimes, battery level, phase of the moon, etc.)
- TOML configuration – looks 0.01% more amazing after 10 hours of config



WezTerm - GPU-accelerated terminal emulator

- Built-in multiplexing (tabs, panes, windows) - no need for tmux/screen
- Configurable with Lua (spent nights customizing WoW addons? Now you can do the same during work hours)
- Cross-platform & written in Rust

<https://github.com/wezterm/wezterm>

A screenshot of a WezTerm terminal window. The window has a dark background with light-colored text. The title bar at the top shows the file path 'vi src/main.rs -- wez@wez-mbp:~/wez-personal/wezterm'. The code displayed is Rust code for a terminal emulator, featuring comments in orange and function definitions in green. The code includes imports for 'portable_pty', 'crate::font', 'portable_pty::PtySize', and 'std::env'. It also contains a function 'get_shell()' that determines the shell to run based on the '\$SHELL' environment variable or the system's password database. The bottom status bar shows 'src/main.rs' on the left and '82 0x52 35,19 8%' on the right.

```
vi src/main.rs -- wez@wez-mbp:~/wez-personal/wezterm
use portable_pty::cmdbuilder::CommandBuilder;

mod font;
use crate::font::{FontConfiguration, FontSystemSelection};

use portable_pty::PtySize;
use std::env;

/// Determine which shell to run.
/// We take the contents of the $SHELL env var first, then
/// fall back to looking it up from the password database.
#[cfg(unix)]
fn get_shell() -> Result<String, Error> {
    env::var("SHELL").or_else(|_| {
        let ent = unsafe { libc::getpwuid(libc::getuid()) };

        if ent.is_null() {
            Ok("/bin/sh".into())
        } else {
            use failure::format_err;
            use std::ffi::CStr;
            use std::str;
        }
    })
}
```

Conclusion

These tools will:

- ✓ Improve code discovery
- ✓ Enhance terminal experience
- ✓ Speed up your workflow (or not)

Use the right tools; don't be a tool 🐟