Increase your productivity with neovim, language protocol servers, and modern terminal tools

Jan Eitzinger, NHR@FAU

NHR@FAU HPC Café, July 15, 2025

Introduction

This talk is about

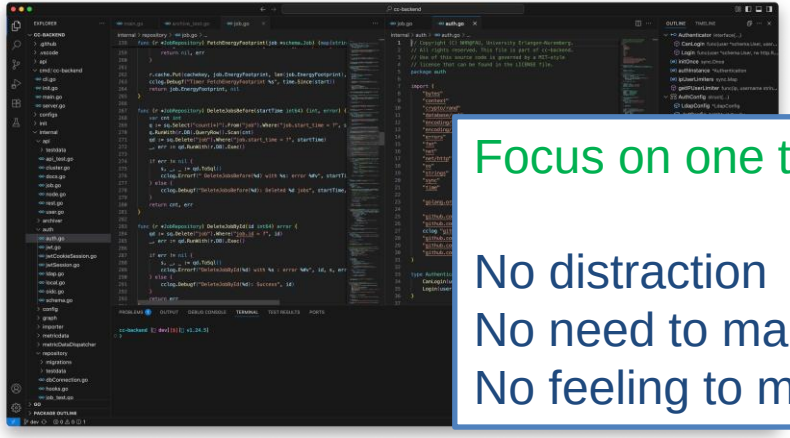
- Tips and tricks about TUIs (terminal user interfaces)
- Workflow suggestions and tools
- Personal preferences and tastes

- Take this as an inspiration and pick what suits you best

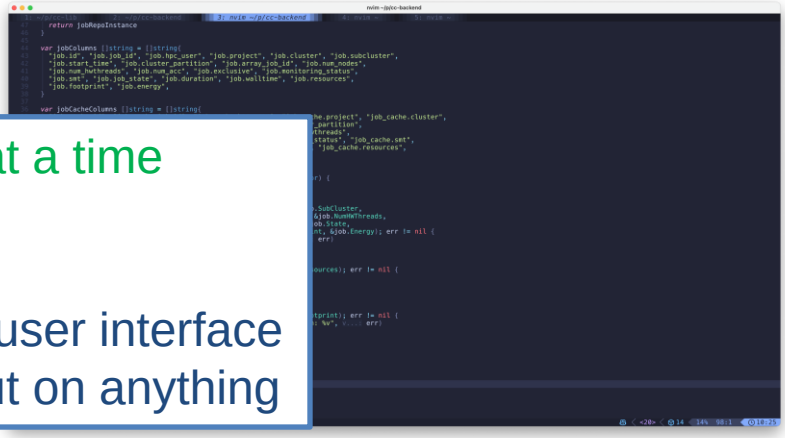
- What is not covered
  - Remote editing with VS Code
  - HPC specific workflows

Personal UI preferences

IDE



UNIX way



Focus on one thing at a time

No distraction
No need to manage user interface
No feeling to miss out on anything

- Everything is integrated
- You never leave the IDE Window
- Lots of things shown at once

- One tool for each task
- Only show what you currently use
- Switch between views for tasks

Use it or loose it

- I am really bad at remembering stuff
- If I do not frequently use something I forget about it

- Consequence
  - Make things as simple as possible
  - Use contextual help
  - Use intuitive interfaces where common things are easy to do

- Pain limit to switch to fancy but less common functionality
- I do not introduce fancy functionality without a real use case

Options for ssh remote access

- Policy: No private key on shared multi-user system!
- On all mobile systems: Store private key on YubiKey

- Access all internal systems via proxy-jump
- No ssh agent running:
  - Initial connection with YubiKey pin
  - Use persistent ssh sockets for subsequent logins (Control master feature)

```
ControlPersist 4h
ControlMaster auto
ControlPath ~/.ssh/sockets/socket-%r@%h:%p
```

Requirements for the terminal emulator

- The terminal user interface has evolved over the years
  Unicode, true color, bold/italic fonts, text formatting, colored and styled (curly) underlines, image rendering
- UTF-8 Nerd Fonts. (Ab)use UTF-8 fonts for icons, graphical elements
  https://www.nerdfonts.com/

- Why tmux/screen is a bad idea?
  - Unnecessary overhead and complexity cascade
  - Escape codes need to be parsed, translated, and modified
  - Hinders innovation in terminal interfaces
- Only useful feature: Remote persistence
- Persistence functionality can be provided by **`shpool`**, **`abduco`**

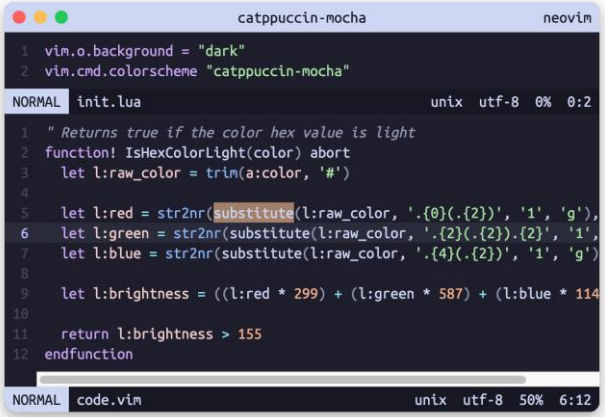
Kitty, the terminal emulator I use

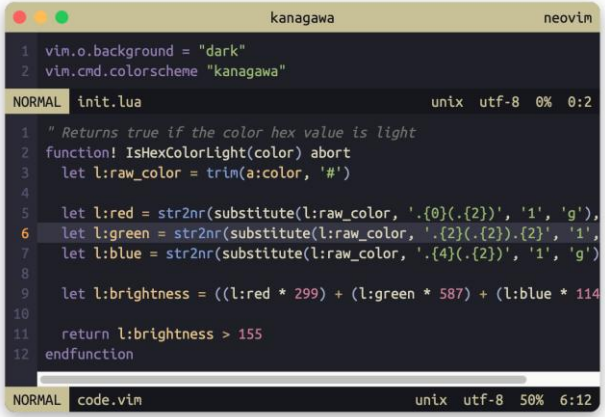
- Works on Linux, MacOS, and BSDs

- GPU accelerated for great performance
- Configured with text file, but configuration helpers available
- Sane defaults, feature rich, and highly configurable
  - Tabs
  - Startup session layouts
  - Configurable font rendering
  - Font overlays (useful to load Nerdfonts separately)
  - ssh wrapper to propagate termcap info and integrate remote session
  - Sane scroll back buffer and copy-and-paste functionality

Colorschemes: The most important choice!

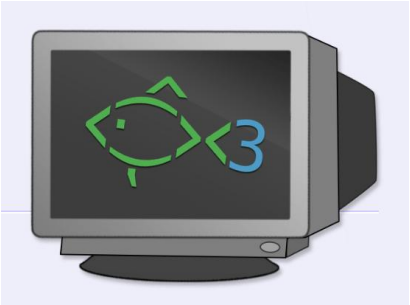
- Good colorschemes come with wide range of plugin and tool integration
- Good overview: https://vimcolorschemes.com
- Recommended:
  - [Catppuccin]
  - [Tokyonight]
- Based on look:
  - [Kanagawa]
  - [Nord]





The shell

- Up front: I do not use any advances shell features. For anything more complex than a single command I use Perl scripts.



- Common options: **`bash`**, **`zsh`**

- My choice: fish
  - Built-in fast autocompletion and syntax highlighting
  - Sane shell script syntax (though I still don’t use shell scripting)
  - No need for complex configuration, sane defaults, gets out of your way
  - Did I mention its fast

How to install tools

- Many novel terminal tools are implemented in Rust or Golang

Rust

Install toolchain

```
curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh
```

Will install all tools in `~/.cargo/bin`

Install applications:

```
cargo install <appname>
```

Golang

Install toolchain, 75MB (500MB on disk) (e.g. into `~/.local`):

```
tar xzf go1.24.5.linux-amd64.tar.gz
export PATH=$PATH:~/.local/go/bin
```

Install applications:

```
go install <URL>
```

- Often you can just download a binary and put it in your path!

NHR@FAU specific notes

- Neovim release distribution for old GLIBC versions:
  - https://github.com/neovim/neovim-releases
- Build everything on a cluster frontend, especially cargo builds
- Option for Kitty kitten ssh to enable fish shell on login
  - `~/.config/kitty/ssh.conf` : `login_shell fish`

Options in `~/.config/fish/config.fish` to enable modules in fish shell (and setup other tools):

```
source /apps/modules/5.0.1/init/fish
atuin init fish | source
zoxide init fish | source
starship init fish | source
```

Manage dotfiles: Chezmoi

Use case: You have multiple computers with potentially different operating systems and want to backup, track, synchronize your configuration files

Solution: https://www.chezmoi.io/

Command line application

- Stores your dotfiles in a separate git sandbox
- Templates (to handle small differences between machines)
- Password manager support (to store your secrets securely)
- Full file encryption (optional, using gpg or age)
- Very good documentation

Command prompt

- Shell prompt: https://starship.rs/
  - Fast
  - Configurable
  - Shows relevant infos based on context
  - Cross shell
- Install

```
curl -sS https://starship.rs/install.sh | sh
```

- Enable in shell

```
starship init fish | source
```

```
unrz254@ssh fritz1:~ [ v8.5.0-gcc][ v5.26.3][ v3.6.8]
❯ ls .cargo/bin
```

```
cc-backend [ dev][$✘!?⇣][ v1.24.4][ 42s]
❯ lazygit
```

Shell history (aka admins second brain)

- https://atuin.sh/
  - Stores shell history in SQLite database
  - Records additional context for your commands
  - With this context, produces faster and better search
  - (optionally) Sync shell history between multiple machines. Fully end-to-end encrypted.

```
Atuin v18.6.1                                   <esc>: exit, <tab>: edit, <enter>: run, <ctrl-o>: inspect
 Search | Inspect
   43ms   7d ago z cc-me
   29ms   7d ago ls
   25ms   7d ago go get -u
   3s     7d ago go get -u ./...
   4m     7d ago ssh monitoring
   17ms   7d ago got get
```

```
 5 4m     8h ago brew upgrade
 4 0s     8h ago exit
 3 6h     6h ago nvim
 2 10m    4h ago nvim ~/.ssh/config
 1 3m     3h ago nvim ~/Downloads/uiconfig_brainstorm_reduced.json
 > 0s     1h ago ssh fritz
[    GLOBAL    ]
ssh fritz
```

Fuzzy search everything

- fzf (https://junegunn.github.io/fzf/)
  - General-purpose command-line fuzzy finder
  - Includes live preview UI
  - Can be used to design many use-cases using simple shell scripts
  - Inspirations: https://github.com/junegunn/everything.fzf

Graphical search tool interfaces for the Desktop

- Apple Spotlight (Part of MacOS)
- Alfred (https://www.alfredapp.com/)
- Rofi https://davatorium.github.io/rofi/ (window switcher and application launcher)

Basic UNIX tools on steroids

- **`ls`** : eza (https://eza.rocks/) Coloring and more information. Better defaults.
- **`cd`** : zoxide (https://github.com/ajeetdsouza/zoxide) Remembers most frequent directories and jump with typing only a few characters
- **`find`** : fd (https://github.com/sharkdp/fd) A simple, fast and user-friendly alternative to 'find'
- **`grep`** : rg (https://github.com/BurntSushi/ripgrep) Recursively searches directories for a regex pattern
- **`top`** : btop (https://github.com/aristocratos/btop) Awesome resource monitor
- **`cat`** : bat (https://github.com/sharkdp/bat) Feature rich cat replacement. Used by many other tools. Great as colored manpage viewer.

Application classes **I don't use** in terminal

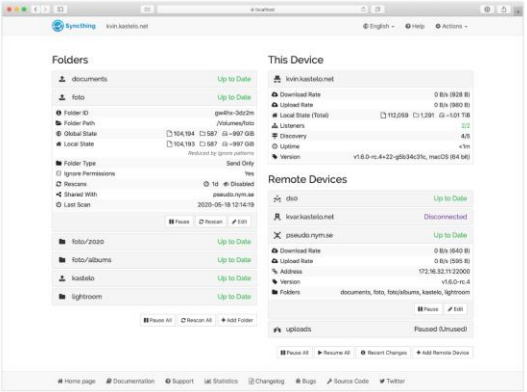
- Email
  - I use Apple Mail for convenience
  - Terminal options: **`mutt`**, **`neomutt`**, **`aerc`**

- File Manager
  - I use Forklift two-pane GUI file manager (MacOS only)
  - Terminal options: **`broot`**, **`yazi`**, **`ranger`**, **`mc`**, **`vifm`**

- Merge/Diff tool
  - I use Araxis Merge (commercial with academic teaching license)
  - Terminal options: **`vimdiff`**, **`neovim -d`**
- Password Manager
  - I use KeepassXC
  - Terminal options: **`pass`**, **`gopass`**

Synchronization and backup

- Syncthing (https://syncthing.net/)
  - Continuous decentralized file synchronization
  - Fully encrypted and authenticated
  - Very fast
  - Easy to setup and use



- Restic (https://restic.net/)
  - Fast and secure backup program
  - Supports many targets (local FS, SFTP, S3, and many more)
  - Easy to setup and use
  - Efficient incremental backups with de-duplication

More tools: lazygit, zk

- LazyGit git tui (https://github.com/jesseduffield/lazygit)
  - Awesome git user interface
  - Context sensitive help
  - Easy to use defaults to otherwise complex features
  - Makes git actually easy and fun to use

- Zettelkasten notetaking (https://github.com/zk-org/zk)
  - Markdown based notetaking
  - Link and tag notes
  - Easy search and navigation of notes
  - Very good editor (nvim and vscode) integration via zk LSP

Modern text editor features

- Autocompletion (Textual, LSP, AI, Snippets)
- Autoformatting (and linting) (format on save, fix includes/imports)
- Language Server Protocol (LSP)
  - Navigation (Jump to definition, list references)
  - Information (Show function prototype and type information)
  - Faster feedback loop while editing
  - Refactoring (Rename variables or functions and linting advices)

- Snippets (have templates with placeholders for common code patterns)
- Fuzzy search with preview everything
- Integrated testing and debugging

My Neovim Setup

- Configuring Neovim is tedious and requires constant attention to keep up to date
  - Use a maintained Neovim distribution: LazyVim (https://www.lazyvim.org/)
  - Builds on plugin manager with incremental overwrites
  - Can be easily tailored to your requirements
  - Mason LSP, formatter, linter installer builtin

- Additional plugins I use
  - Oil File Manager (https://github.com/stevearc/oil.nvim)
  - zk-nvim: Plugin for zk Zettelkasten notes
  - nvim-autopairs: Autopair plugin (https://github.com/windwp/nvim-autopairs)

Essential Neovim plugins (part of lazyvim)

- Built-in neovim: LSP, Snippets, syntax highlighting, and much more

- Package and tools management: **`lazy.nvim`**, Mason
- Picker UI: **`snacks.nvim`**
- Autoformat: **`conform.nvim`**
- Autocompletion: **`blink.cmp`**
- Context keyboard shortcut help: **`which-key.nvim`**
- Issue aggregator: **`trouble.nvim`**
- AI based code completion and chat: Codeium/Windsurf, GitHub Copilot
- Fast keyboard driven movement: **`flash.nvim`**

LSP setup for HPC programming languages

- Very good C/C++ LSP: LLVM clangd
- But you need to
  - Setup flags, defines, and include paths manually
  - Only opened files are considered unless you setup compile database file `compile_commands.json`
- https://github.com/rizsotto/Bear Generate `compile_commands.json` from build output

- Fortran language server: https://fortls.fortran-lang.org
- Python language server: https://github.com/microsoft/pyright

The role of AI in programming (to me)

AI fan boys: Programming is now a solved activity the AI can easily take over.

My take (from my limited experience): AI based tools are an additional puzzle piece in what a software developer can use to get the job done

Common applications

- Use ai snippets in autocompletion
- Talk to AI chat interface to answer specific questions and generate code examples

Things to consider

- The underlying technology did not change
- You still must understand how things work

Further thoughts

- AI chat bots are today faster to find relevant information, also because standard web search has become really bad
- AI is effective if you can judge and validate the generated code is correct
- If you are a good programmer, you can use the AI code as starting point and continue from there
- If you have no clue what this all means and how things work the outcome is purely coincidence and luck

Demo