My (ideal) setup

September 2022

I have been tinkering my TUI for quite a bit of time actually, and I believe I reached a close to ideal setup I could possibly expect from all the tools I configured along the way.[1] At some point, it's time to stop tinkering and rather optimize what we have at hand. It all evolved over time, and it is somewhat still evolving, but at a safer pace which I interpret as me finding an equilibrium in the tools I need for my working habits. The core software I use the most are Zsh, Tmux and Neovim. Someone on the internet[2] coined them the *text triumvirate*. Indeed, "text streams are a valuable universal format because they're easy for human beings to read, write, and edit without specialized tools", according to Eric S. Raymond [2]. I spend a lot of time dealing with text, in the form of manipulating input or output data and writing hard coded handouts or short notes like this one. Tmux helps me manage the multiple working directories and projects I interact with, while Zsh gives me access to load of utilities that help manipulating and piping text streams. And, well, Neovim let me read and write most of what I consume every day when in front of my computer. Above all, Neovim (or Emacs) is an advanced text editors which gives you some extra power at little cost. Past the first times of discovery, you start to get comfortable with the tooling

> Our work is repetitive by nature. Whether we're making the same small change in several places or moving around between similar regions of a document, we repeat many actions. Anything that can streamline a repetitive workflow will save our time multifold. Vim is optimized for repetition. Its efficiency stems from the way it tracks our most recent actions. We can always replay the last change with a single keystroke. Powerful as this sounds, it's useless unless we learn to craft our actions so that they perform a useful unit of work when replayed. Mastering this concept is the key to becoming effective with Vim. Drew Neil [1]

In this note, I present various TUIs, especially those largely capable of replacing the default applications installed on, say, Linux Ubuntu. Applications suggested in Table 1 also proved to work well on OpenBSD and macOS. This list shows the essential software I run every day, but it is far from being exhaustive of course. Also, many terminals are available and your mileage may vary, depending on whether you like having ligatures enabled in your terminal, or if you like to have some kind of multiplexing capabilities. For instance, Ubuntu ships with Gnome terminal as the default, which does not support ligatures but allows to open different tab in the same window. OpenBSD and OSX support different terminal applications by default.

[1] See, e.g., one review or another on my website.

[2] Seth Brown, aka drbunsen. Unfortunately, the blog post is no longer available.

If you are looking for a universal terminal, I suggest trying out Kitty, Alacritty, or the good old Xterm with Unicode support.

Default	Alternative
bash	zsh
screen	tmux
gedit	(neo)vim
evince	zathura
eog	feh
firefox	nyxt
thunderbird	(neo)mutt
rhythmbox	cmus
gfeeds	newsboat

Table 1: Alternatives to common applications on Linux distros (Ubuntu)

Sidenote: If you want to change default applications in Ubuntu, you first need to query the relevant filetype of a given file, or you can ask what default application is used for a given filetype. This can be done using either `xdg-mime query filetype` or `xdg-mime query default application/pdf`. Then, you can update the user database with `xdg-mime default org.pwmt.zathura.desktop application/pdf`, for instance. The `desktop` file are usually located in `/usr/share/applications` or `$HOME/.local/share/applications`. Be sure to check the correct filetype first; for instance, a Postscript file has filetype `application/postscript`, not `application/ps`.

Interacting with the shell

My preferred shell is Zsh. I have been using it for years, and this is usually the first tool I install on external server I am interacting with. I could probably be happy with Bash, but it would mean a lot more customizations than what I have here with Zsh. Fish is great too, but now that I finally manage to get everything working in Zsh like it works in Fish I think I prefer to stay with a POSIX-compliant shell, if only for my daily job where I use to use a lot of shell-oriented workflows. I don't use any external framework for managing my plugins, like OhMyZsh and the like. My current Zsh configuration includes auto-suggestions and auto-pairing of matching delimiters,[3] but I disabled syntax highlighting. The setup is simple:

[3] Note that I keep local copies of those projects under version control (Git) but I never updated them.

```
source ~/.local/share/zsh/zsh-suggest/zsh-autosuggestions.zsh
ZSH_AUTOSUGGEST_HIGHLIGHT_STYLE="fg=#d0d0d0"
ZSH_AUTOSUGGEST_STRATEGY=history
ZSH_AUTOSUGGEST_HISTORY_IGNORE="(git|ls|less|rm) *"
ZSH_AUTOSUGGEST_USE_ASYNC=1
source ~/.local/share/zsh/zsh-autopair/autopair.zsh
autopair-init
```

I have a set of dedicated Zsh files for managing aliases, functions, and the prompt, although in the later case I now rely on starship to get an unified (and optimized) prompt across the machines and shell I happen to interact with. In addition to starship, I use Fzf for many things: fuzzy searching in the history, finding files in a given directory, managing my BibTeX entries, interacting with Tmux, or looking for some commits in a Git repository. I use the following settings:

```
source /usr/share/doc/fzf/examples/key-bindings.zsh
source /usr/share/doc/fzf/examples/completion.zsh

export FZF_DEFAULT_COMMAND='rg --files --follow'
export FZF_CTRL_R_OPTS="--layout=reverse-list --height 100%"
export FZF_CTRL_T_OPTS="--layout=reverse-list \
  --info=default --height 20% --preview='head {}' \
  --preview-window right:50%"
export FZF_CTRL_T_COMMAND="$FZF_DEFAULT_COMMAND"
export FZF_DEFAULT_OPTS="--no-mouse --height 20% \
  --layout=reverse --no-info --color=light \
  --color=bg+:#2E3440,fg+:#ffffff,hl:#81a1c1, \
  hl+:#88c0d0,marker:#5f87af,pointer:#81a1c1, \
  prompt:#96522b"
```

Editing code, email and prose

All text-related stuff is managed with Neovim. I have been a hard time Emacs user for years, but now I prefer the Vim approach to text editing. I use the nightly build version of Neovim, which I update every week via Ubuntu package manager system. I use packer to manage my plugins, but it's only because I'm to lazy to autoload plugins for specific filetypes. In fact, packer's lazy loading feature is the only thing that makes me keep this plugin in my config. I came to appreciate builtin features of Vim, then Neovim, and tend to favor their use over more complex workflows, especially those involving load of external plugins. I tend to favor all kind of minimalism these days, and in the end I think I could probably live with less than 10 plugins. This is low standards compared to some of the settings I find regularly when browsing GitHub.

The mappings I use are described on a separate page on my website. Previously I was using the wonderful fzf plugin for fuzzy searching in Vim, and now I use Telescope, which is a Neovim only thing. The builtin LSP server in Neovim makes coding easier than before when we have to install lot of plugins for each programming language. However, I use very few of its capabilities: Python has no code ac-

tion or reliable refactoring tools, Scheme does not have a language server (only Racket does), and I do not write that much Haskell code. However, I appreciate the tight integration of Telescope and LSP commands like go to definition or references, renaming, or (range) formatting.

For prose I tend to favor Org files, although I have been using Markdown for more than 10 years before that. I simply find Org syntax more pleasant, and I often end up with the results I want, even in case of nested lists or when an image is inserted right after an item in a list, while with Markdown the result may vary (from two to four spaces, usually). For a long time, Pandoc has been my exporter of choice for Markdown files. I could also use a two-pass export (Org → Md → PDF), but then I realized it is easier to manage everything from within Org mode tools, and then using `org-export`. This assumes you are not too picky with table design, though. Now, with a single key press from within Neovim, I can export a buffer to a PDF handout which pop up in Zathura right away, and I never had to revert to Emacs to edit my Org files.[4] Org syntax supports TEX expressions[5], which means I rarely need to write plain TEX documents nowadays. When I do, I use the excellent vimtex plugin, which offers synctex support for Zathura PDF viewer.

I'm not a GTD guy, and I store all my personal or work notes in plain Org files in one or two dedicated folders on my hard drive.

[4] You really only need to call Emacs from a shell script to process your document.
[5] which solves the problem of designing good looking tables, at the price of extra syntax in your Org file

Managing emails

I use Neomutt for all things related to email (private, work, and mailing-list).

Media viewer

For PDF, I settled upon Zathura, mostly because of its synctex capabilities (forward and backward synchronization with Neovim) when working with TEX processing and its keyboard-driven features. It is even possible to specify your own color scheme, and in case you wonder documents rendered using a dark theme are still perfectly readable (Figure 1).

References

[1] Drew Neil. *Practical Vim: Edit Text at the Speed of Thought*. 2nd ed. The Pragmatic Bookshelf, 2015.

[2] Eric Steven Raymond. *The Art of Unix Programming*. 2003.

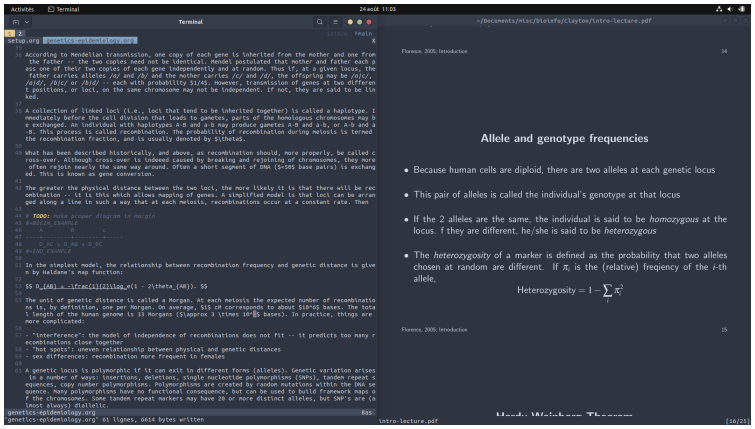

Figure 1: Neovim in Gnome terminal (left) and Zathura viewer (right)