Research Workflow in Plaintext

Arumoy Shome

2021-07-12

Contents

1 Introduction

In this talk I will go over how we can use Emacs and org-mode to craft a research workflow. We will look at how we can leverage the power of Emacs and org-mode to capture, store, search and retrieve research data, all in plain text! The talk will touch upon how org-mode can be used as an environment for literate programming and reproducible research. I do not assume any prior knowledge of emacs or org-mode and I want this to be more of a discussion rather than a talk. Please ask me questions as I go along and share your thoughts, tips and techniques with others!

1.1 Outline

Outline of this talk is as follows:

1. I will start with a brief history of Emacs and org-mode.
2. A brief background on plaintext.
3. We will follow with some motivation for why we would want to use plaintext to manage information and some of my supporting philosophies.
4. Next, we will jump into the building blocks of org-mode.
5. Which we will leverage to craft research workflows and introduce some automation into menial tasks.
6. I will highlight some additional features of org-mode that we can use for literate programming and reproducible research.
7. I will end this talk by pointing out additional resources that you may use to get started with Emacs.

1.2 A Note for the Audience

When I was preparing material for this talk, I thought: what better way to talk about emacs and org-mode than to do it in emacs itself. There is a lot of text compared to a traditional presentation, and there is justification for doing so:

1. This talk is highly interactive and there are demos throughout. It would have been a bit annoying having to constantly switch between emacs and my slides.

2. There is another reason however I will reveal this towards the end of the talk.

1.3 A Note for the Readers

This document is published on the Internet but can only be accessed if you have the url (you cannot access it via a web search engine). You can navigate back to this document by following the link on my website which is public and the url is easier to remember.

2 History & Background

When I say text editors, you are most likely are think of software such as Atom, Sublime Text and VS Code. That's what the normal folks use to get their job done. However, you may also be aware of this group of elitists who shun such tools, have a neckbeard, dwell in dungeons. . . and use Vim.

2.1 Emacs

Emacs is the **other** text editor that you may have heard of aside from Vim. There are several flavours of Emacs, the most popular being GNU Emacs which was created by Richard Stallman. The core, performance critical components of Emacs is written in C, however majority of it's codebase is written in emacs-lisp (elisp) which is a variant of Lisp.

The first public release of Emacs was in 1970 (it has 20 years on Vim which was released in 1991), it is fully open-source, has had 27 stable releases, has a vibrant and large community and continues to be in active development to this date.

2.2 org-mode

org-mode is an emacs package. created by Carsten Dominik (professor of Astronomy at UvA, faculty of natural sciences) in 2003. org-mode provides two core functionalities:

1. It provides a major-mode (Emacs lingo for syntax highlighting based on filetype) for plaintext files.

2. And a slew of tools to organize and edit information in the form of lists (I like to think of them as trees).

2.3 Plaintext

We have been talking about plaintext and I think it's important to clarify what I mean. Plaintext in this context means text files which are unencrypted and written in a format (.{txt,md,org}) that is readable through a terminal emulator.

3 Motivation & Philosophy

3.1 Why use Plaintext?

Why would we want to use plaintext to store information? There are several benefits, and the word "Freedom" encapsulates everything well. By freedom, I mean:

1. Freedom to do what we see fit with our data. Its stored as plaintext files on our system. We can choose to keep it that way, upload to our trusted cloud provider or even introduce encryption for additional security and privacy.

2. Freedom from yet another proprietary software. There are solutions that market themselves as systems for note-taking, knowledge management or a "second brain" such as Notion and Evernote. But the problem is that they are in control of your data (stored on their servers) and they often store this information in a proprietary format resulting in a vendor lock-in.

3. Freedom to choose (or craft) our own tools which are optimised for us and can scale with our ever-growing body of knowledge (remember, learning is a lifelong journey).

3.2 Plaintext & Productivity

Looking at the current landscape of productivity tools, I do not see anything that aids in exercising the one true productivity system that actually matters: our brain. Ultimately, it all comes down to effort and time taken to read papers, understanding concepts, learning, and forming connections which give rise to ideas. By takes away the "fluff", plaintext allows us to focus on the essential. The text. The ideas. The connections.

There is no "one size fits all" or "cookie cutter" solution when it comes to productivity. The problem with existing productivity software is that they enforce their own structure and constraints which does more harm than good. I have given this a fair amount of thought and experimentation. In my experience, a two step process of 1. Capturing information quickly and 2. The act of reviewing captured information and deliberate summarisation works best. To this end, we will see how org-mode can help us with this.

3.3 Importance of Structure & Standardisation

I made a profound observation while reflection on my current research workflow. I spent a lot of my time during my masters on finding tools and systems for managing knowledge. Extrinsic search for existing solutions/methods was not fruitful. Instead, I used whatever solution came naturally to me. Through several iterations and minor changes, I developed a set of rules to store the data in a consistent structure. From this structure, developing tools to automate the process developed organically. I think this is such a simple thing, which makes it so powerful. We see similar phenomenon in software as well. We have decades of research which boils down to doing things in a consistent and standardised manner.

3.4 Non-linear Nature of Research

The main take-away for me was realising that research is dynamic, non-linear and personal. Finding the right tools is a journey which we must make ourselves. Using plaintext grants us the freedom to explore and experiment different options and techniques. It allows us to craft our own set of tools, standards and techniques which are curated towards how we think and operate.

4 org-mode: First Principles

Text in org documents are stored in an hierarchical manner, in the form of outlines.

4.1 Document Structure

We can create a new header (or "tree" in org lingo) by prepending the header title with a `*`. Trees can be nested, a tree may have several other sub-trees within itself. Sub-trees are created by adding more `*`, the number indicates the depth. We can promote or demote headers quickly using the `M-left` and `M-right` keybindings.

We can hide or show specific sections of a document by pressing `TAB` while positioning the cursor (or "point" in emacs lingo) on the header. We can also cycle the visibility of the entire document using `S-TAB`.

Org provides a few handy keybindings to move between headers. `C-c n` (mnemonic: "next") takes us to the next header while `C-c p` (mnemonic: "previous") takes us to the previous header. `C-c u` (mnemonic: "up") can be used to navigate to the parent of the current sub-tree. `C-c f` (mnemonic: "forward") and `C-c b` (mnemonic: "back") navigates between headers of the same level.

We can edit the structure of a org document quickly by using `M-down` and `M-up` to move headers.

4.2 Todo States

Org was originally built to manage tasks in plaintext. A header can be converted to a task by prepending it with a "TODO" keyword. Org also understands the notion of various stages that a task may go through during it's lifecycle. We can change the todo state interactively using `C-c C-t`.

The todo states are customizable and we will talk more about the ones I have created and use regularly to organize scientific research.

4.3 Capturing & Refiling

NOTE this section requires some setup and plumbing before it can be replicated. Refer to the org manual on capture to get started.

Earlier I mentioned the notion of capturing information and reviewing them at a later stage as a system for productivity (see the section on Motivation & Philosophy). Let's dive deeper into the capturing segment and explore how org can help introduce some automation into this process.

We can invoke org-capture using `C-c c` which prompts us for a capture template. This is customizable and the most commonly used one is to capture tasks. Once we are satisfied with our content, we can use `C-c C-c` (this is also the universal keystroke you would use to tell emacs that you are "done" with something or to "confirm" something) to confirm, save and close the capture buffer. Later, I will also talk about the custom capture template that I have written for my research workflow.

By default, the captured text is saved in a predefined file, however we can also choose to save it somewhere else using org-refile (`C-c C-w` in the org-capture buffer).

5 Research Workflows

Let's see how we can leverage what org has to offer in order aid us in scientific work.

5.1 Planning & Structuring Papers

I find org to be particularly useful for planning and structuring scientific papers. I also use it to capture my thoughts, ideas and notes for various projects that I am working on. Once I have accumulated a large body of text, I find the the "narrow" functionality provided by emacs to help me focus on a particular section or region of text.

5.2 Bibliography Management

Besides keeping track of tasks, I use org to manage bibliographic information of scientific publications. Let's look at an example in a project that I worked on earlier: shokri2015privacy.

Org allows us to assign metadata to org headers known as org-properties. For instance, I like to store the first author, last author, source of publication, year of publication and a link to the pdf file on my disk as properties. Using the `org-sparse-tree` command (bound to `C-c /` in org-mode buffers), we can quickly find what we are looking for.

5.3 aocp.el

As you may expect, this task is repetitive and entering these properties manually becomes cumbersome. To automate this process, I wrote an emacs package: aocp.el. The package provides a few helper functions which are meant to be used in an org-capture template (refer to the project readme for more information).

6 Infinity & Beyond

Org let's use schedule tasks and assign deadlines, org-agende acts as the front-end and more.

We can write arbitrary pieces of code within org documents and use org-tangle to extract and create a source code file.

Using org-export, we can export org documents to other formats for simplicity of sharing and collaboration.

Using org-publish, we can run a website from within emacs.

Since we can mix text and code in org documents, it becomes trivial to document experiments - complete with code snippets - which are reproducible. In fact, org is a popular choice within the community to document their emacs config. Yes, org-mode is basically Jupyter notebooks in plain-text!

And the list goes on. Refer to the org manual for an exhaustive list of what org can do.

7 Closing Thoughts

"Premature optimisation is the root of all evil." — Donald Knuth

If you already have a system, methodology or tool that works for you, stick with it! There is no point reinventing the wheel, especially if you are content and productive with your existing system. You could also be on the other end of the spectrum, where you spend a lot of time configuring your system and tweaking your dotfiles!

That being said, if you are curious about emacs and want to learn more, I would strongly suggest to first get in the right mindset and set reasonable expectations. Emacs is not just another took, it's a way of being. Learning everything that emacs has to offer will take a long time (I have been using emacs full-time for over 3 years and I am still learning new things!). Instead, start small, learn a small part of the system which will help you get the job done. Iterate, and slowly craft an environment that works for you.

7.1 Additional Resources

A few resources and pointers to help you get started:

1. Start with the tutorial: `C-h t`.
2. Read the manual: `C-h r` or alternatively `C-h i` to open the documentation viewer (info) and select "Emacs".
3. Use the excellent and extensive emacs help system, some of the most frequent ones that I use are: `C-h f`, `C-h v`, `C-h o` and `C-h m`.
4. Consult the emacs wiki. I usually use it as reference once I have read about a particular topic in the manual.

5. Watch Harry Schwartz's excellent talk on getting started with org-mode.
6. Watch Howard Abrams' excellent talk on emacs introduction and demonstration.
7. Consult my emacs config files if you prefer code.