

Note Taking Manual

Ryan G

February 3, 2020

Contents

TODO Clean up and Replace previous Method	1
TODO Clear off Workload of attempts	1
Introduction	1
Org-Mode and Markdown	1
.1 Problems with org-mode	2
.2 Strengths to org-mode	2
.3 Use Case	3
.4 Using pandoc	4
Sharing Material	5
Markdown	5
Org-Mode	5
HTML and iOS	5
Writing Material	5
Diagrams	5
.1 Apple Pencil	5
.2 TiKz	5
.3 InkScape	6
Finding Material	6
Tags	6
.1 Yaml Tags	6
.2 #Tags	7
Viewing Material	10
Vim-Plugin	10
Firefox	10
Chrome Browser Extension	10
WYSIWYG Editor	10
Notable	10

TODO Clean up and Replace previous Method

+ Old Manual

TODO Clear off Workload of attempts

- [To Do List of NoteTaking strategies](#)

Introduction

The important features are:

- Writing Material
- Finding Material
- Reading Material
 - Mobile
 - Desktop
- Sharing Material

Org-Mode and Markdown

While org-mode is full featured and powerful, there are a few reasons I don't like it for the backbone of a note taking system:

Problems with org-mode

- Emacs with org-mode is just far too unstable, I've spent hours, well no weeks, literally 9-5 weeks trying to get it stable, you blink and it fails to export, or inline create math or whatever, whereas markdown just works, lacking in features, but, it just works.
- Emacs takes an enormous amount of configuration
 - my `~/.emacs.d/init.el` is already at like 500 lines and I'm using a starter kit (Spacemacs)
 - * With Emacs 28, this is now in `~/.config/emacs/init.el` where it should be but I don't really look forward to moving it.
- Emacs can get really slow
 - This is probably on my end, but again, a pain in the ass
- No mobile editors (see .2 below)
 - I don't like [beorg](#), it sucks for long notes, doesn't support inline math and it doesn't work well with git, using Dropbox is nothing but a headache when it comes to syncing.

- * I've heard good things about it's support for the agenda but I don't use `org-mode` like that, I use the `todo` features to track a specific project in an index but I use [ToDoist](#) because it's just simpler to use.
- It only works with emacs
 - The Vim Rendition is half-way there
 - markdown works in both and emacs doesn't scroll with the mouse which is a shame on the laptop.
 - You'll miss `fzf`, `helm-locate` and `helm-ag` don't even come close.
- No live preview for math sucks
- Math inline preview will break if there is a bug above
 - this is a dealbreaker honestly, it's ridiculous
- Exported Math depends on previous math, whereas markdown is compartmentalised
- `yasnippets` doesn't hold a candle to using [UltiSnips](#) to write `LaTeX` in vim. and `company-mode` isn't much better than [Deoplete](#), but i'll admit `YCM` is a buggy mess that fails when you blink
- bugs in `init.el` can break HTML export

Strengths to `org-mode`

Although it does have some features that justify using it:

- inline `LaTeX` preview is really great
 - if you're willing to compile `ghostscript v50` from source if you're not on Arch-Linux
 - * oh and you can tolerate opening your `init.el` to increase the font size as opposed to using `C-x C-+ C--`
- It really is more readable as raw text than Markdown
 - This means if you use [WorkingCopy](#) on *iOS* you can practically edit and work with your `org` set-up
 - * If you're using [org-wiki](#) you can even follow the HTML File through all the links which is great, because I think the tags and agenda suck for navigating notes/notebooks.¹
 - * Alternatively you could use `python3 -m http.server 8351 --bind 192.168.0.134` to look on the ipad, but this is annoying because you have to grab your ipv4 before doing that if you're out of the house unlike [iamcco's](#) preview which does it automatically.
- Foldable Code Boxes out of the box
 - with MD you have to use `set foldmethod=manual` and a [text-object](#) plugin.

[./images/WorkingCopyiOSwithOrg.gif](#)

¹Yes I've been patient, no it doesn't work for me.

- Once it is set up creating links with SPC a o l feels amazing
- Inline code is amazing with outshine mode
 - I cannot get outshine to play with R though so I don't know
- Literate Programming with it is amazing, so many of my bash scripts start as org-mode using org-edit-special and then I just pull the bash out [fn:9] and use =SPC m e e t u to give myself a manual/help dialog for it, it feels like magic!!
- Managing long documents with emacs is a **lot** easier, because the folding feels very natural and the syntax highlighting is brilliant.
- [citeproc-org](#) makes your HTML documents look just like they would with BibTeX
- org-mode supports the minted package out of the box (minted looks better than listing and actually supports most languages like vimscript and elisp)
- org-mode interprets \LaTeX properly and then passes it to mathjax, markdown is so confusing because you wrap everything in \$ even if you wouldn't in \LaTeX and I think that's ridiculous.
- you never need to leave emacs
 - I map C-c e/=v to vim emacs so it shouldn't matter.

Use Case

Basically the choice of format will depend on the note type,

- A long document like a manual will go into org-mode
 - because the interlinking is brilliant and the inbuild HTML export
- A short note will go into md because
 - fzf works in vim works with md
 - more apps support it ([MWeb](#)/[iaWriter](#))
 - More programs have support for smaller document notes:
 - * [Notable](#)
 - * [VsCode Nested Tags](#)
 - * [VsNotes](#)
 - live preview of math is excellent
 - Two features, missing by org-mode² but available for MD which are **crucial** to filtering through small notes (and available in *Evernote*) are:
 - * Something I call *fuzzy-search-with-live-preview* which is provided only by *Notable*
 - although [fzf --preview](#) comes pretty close
 - This is implemented in vim with [notational-fzf](#), which is amazing

²Atleast easily in a fashion I could make work

- * Recursively Filtering through tags
 - this isn't available in `md` but can be easily implemented with `#tags` and then you can use a live preview with something like [MarkText](#) or a vim auto live preview like [iamcco's](#) which is unavailable for

Using pandoc

Thankfully `pandoc` means you shouldn't really too much about this stuff though, you can literally paste `org-mode` out of the clipboard into Markdown and vice-versa if you map a bash script to a keybinding³

Actually with [pipes](#) you can do all sort of cool things, like this is an example of how to pull a website into markdown:

and reference it using [beautifulsoup](#):

1. Problems with links Links may become a pain in the ass to maintain, so what i do is copy the file name to the clipboard and use `fzf` with this bash script to make a relative link to a file:

Sharing Material

Markdown

-
- HackMD is an option
 - *MkDocs* and GitHub

Org-Mode

-
- put the HTML on [Github](#)

³OK, so I know this is a pain in [i3](#), and yes [i3-gaps](#) looks awesome, but you could probably get away with using [xmodmap](#) somehow, and this would be [WM](#) agnostic.

HTML and iOS

- Shortcuts
 - iCloud is quicker
 - Dropbox allows mathjax
 - WorkingCopy (best)
 - * phenomenal, uses mathml for math in MD
 - * uses MathJax to view Math in HTML

Writing Material

Diagrams

Apple Pencil

Tikz

Tikz is awesome but this is actually a bit of work to implement:

See Previous work on [Including Tikz Plots](#)

1. write in \LaTeX
2. export using a lua script
3. grab the svg
 - (a) be aware that getting an svg in \LaTeX is a real pain, so you can go from \TeX to HTML with the lua script but not really back, its madness.
 - (b) Or you could go from svg to png but then you lose scaling so the whole thing sucks
 - (c) You could also use \LaTeX and just export to HTML with Mathjax This is probably the way to go tbh.
 - i. You can open the HTML from notable :shrug:

InkScape

Finding Material

Tags

Tags revolve around:

- Entering the same tag
- Browsing through tags

Yaml Tags

These are basically Notebooks, they make more sense than folders because you can change them using `vim` and `sed` which scales better over many files and is easier to maintain than manually maintaining symlinks across directories.

These are supported by:

- Notable
 - [VSNotes](#)
 - [Nested Tags](#)

And they work really well for there rigid strucutre

1. Inserting YAML Tags ATTACH In order to insert a YAML tag into a note with `vim` you can use FZF to read a text file and make suggestions from a text file with those entries:



In order to get that text file you can use an **R** that leverages RMarkdown to pull the yaml out:

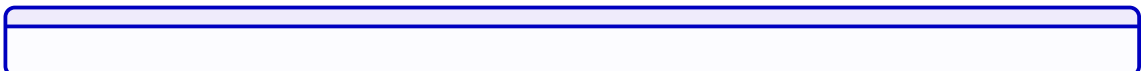


Then you can just regenerate the tags as needed with `$ Rscript makeTags.R` and insert them with `C-c C-y`.

This can be seen in this gif:

[./images/YAMLInsertion.gif](#)

2. Browsing Yaml Tags In order to browse YAML tags just search for the verbatim structure with `ripgrep`, in this context YAML tags are only used to set up notebook directories, so they will always be nested with `/` characters, hence the number of false positives will be small enough to justify this simplicity:



- (a) Folder Structure You can Also browse through the YAML tags like a folder structure, the above **R** Script at 1 uses a nested for loop to create a directory structure with symlinks for the corresponding notes.

#Tags

So the advantage to inline tags is that they are:

1. Really simple to implement and use

(a) for example using YAML basically requires a parser because:

- i. if you're working with something rigid like YAML you need to support it properly or the system will fall apart, but YAML can use:
 - A. python-style lists
 - B. mappings
 - C. New line sequences
- ii. Even if YAML used just one syntax, it would not be easy to implement in regex because look around features don't like to work with wild cards, moreover doing something like `\-\\-\\-\\n[\\w\\W]*tags:\\s` cannot be efficient.

2. Can appear anywhere in your document

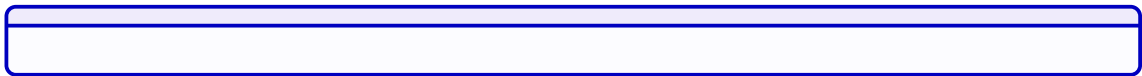
(a) If you use [iamcco's](#) preview, the scroll can be locked to once you jump to the tag it will be in the preview as well.

3. Because they are easy to implement they can be concurrently filtered

I've elected to use #tags rather than @⁴ because it's more common and is implemented by iaWriter, meaning I can use smart folders on the iPad, :tag: was another option I rejected ⁵.

In order to reduce false positives it's important to have a really clear definition of a tag, I've chosen to do `\s#TAGNAME\s`, this should work well, I just need to be mindful to include spaces around the tags.

1. Listing already Created tags This is where #tags really shine, because they are easy to parse and because ripgrep and fzf are both lightning fast all the notes can be searched and all the tags listed on the fly like so:



Then simply presing C-c C-t in vim will allow you to enter the matching tag.

2. Filtering by the desired tag This is where #tags are amazing, because they are so simple, they can be recursively filtered for, first create a temporary file to store any notes that have tags in them⁶, then use ripgrep with look-around to extract the tag name:

First find all the tags and offer the user

⁴This is used by Notes.vim

⁵This is used by VimWiki and org-mode, I don't like them because pymarkdown already uses : to delimit emojis. Also org-mode only uses them in [Headlines](#), according to the [manual](#).

⁶A variable would be quicker than a file because it's stored in memory, I did a file when I was playing in the terminal using for loops rather than pipes and it just made it's way into the script I'm using, there is a TODO next to it but the whole [script](#) needs to be rewritten anyway.

After the tags have been identified and passed to the user, save the `tagval` as a temp file and search through all the notes listed in the file for any matches:

Now you can re run those last two commands (which I've written into a bash script as `tagFilter.sh` [here](#), over and over again until the number of listed tags is down to a reasonable number⁷.

Once you're satisfied that the results are sufficiently filtered you can dump them as symlinks into a direcorey like so:

Then you can go through that directory using `fzf` and `--preview` with ⁸:

Or you you could preview the files in `ranger` using `glow` by appending the following to your [scope.sh](#):⁹

(a) Example This can be seen in the below gif:

[./images/hashtagfilter.gif](#)

3. Integrating with TMSU Alternatively you can leverage TMSU to make all the symlinks, both is probably the way to go because that way you're not tied to TMSU (plus the mounting is a little buggy) but you can still leverage it

TMSU wouldn't work so well with nested `yaml` TAGS because *TMSU* doesn't have any notion of nested tags, instead it would be easier to create symlinks directly from **R** using `R.utils::createLink()`

⁷This is why using descriptive file names is important, with all notes in one directory it will be necessary to have unique file names, but using a sample of 1000 words and never repeating a word and not using the same 3 words in two files there will be $\binom{1000}{3} > 10^6$ different combinations, so there is no need to use a UUID.

⁸I got this idea from the [Github Wiki](#)

⁹`ranger` crashes all the time though, there's some bug with the way the directories are mounted and `vim`, for example `vim` can only change into that directory after being loaded, so `vim -c 'cd 00tmsutags'` is fine but `vim 00tmsutags` is not fine.

4. Browsing through the Symlinks

Then you can mount the TMSU Virtual File System wherever you like and there is an easy to browse through tags.

What works really well though is to open nvim in the directory with:

and then search through the files using `:Files` from [FZF](#) (mapped to `C-p / SPC f f`) and/or [NerdTree](#), and use [iamcco's](#) preview in *Firefox* with [Tree Style Tab](#) to stay organised. If you'd rather work from bash you can use [fzf](#) with the `--preview` option from [before].

Or you could use vim with and a preview app like [iamcco's](#).

Or you could preview all the markdown in a rendered state by using [MarkText](#) / or [Typora](#), or [Zettlr](#), [Abricotine](#), [Typora](#), [MarkdownViewer](#) Chrome Plugin¹⁰, [MkDocs](#) or [Docsify](#), are other options that work in some way.

Another good option for navigating the '*tags are now folders*' structure is to use *VSCode* and/or *atom*, both are well suited to navigating through a dense structure.

[Dillinger](#) is also really cool, but, I don't know what I'd use it for, could I get it on the ipad by hosting my own server maybe?

5. Renaming Tags Just use a careful application of sed if this is necessary:

6. Renaming Files Somewhat integral to the Tag Filtering operation above is reasonable file names, if it's necessary to fix the file name just use `chdir "%p"; ! mv "%" newname.md`.

This will break links, but if a note is already linked from elsewhere you'd make a new note and/or delete the old one.

7. Searching Tag Under Cursor In order to search for the word under the cursor you could just `:Rg` by [FZF](#), `:NV` by [notational-fzf-vim](#) offers a seamless preview as well so I'll just use that.

Add the following to `.vimrc` and you'll be able to search for a tag under the cursor with `SPC f g` and if you forget keyboard shortcuts `SPC Tab` will list them.

Viewing Material

Vim-Plugin

¹⁰You might want to start a python server for this with `python3 -m http.server 8392`

Firefox

while Previewing MD in the browser, in order to keep your sanity, you're going to want to use this [Tree](#) add on to keep yourself organised.

Chrome Browser Extension

Firefox will not allow local MD files to be rendered with an add on as a security policy, instead you can Firefox doesn't allow markdown files to be rendered inside the browser if they are local (nor does it make it easy if you do it with a simple `python3 -m http.server 8089 --bind 192.168.0.137`) you'd be better off just doing it in chrome with [This extension](#)

WYSIWYG Editor

Notable
