An Orgmode Note Workflow

Rohit Goswami · 2020-05-10 · tools · emacs · workflow · orgmode

Background

One of the main reasons to use `orgmode` is definitely to get a better note taking workflow. Closely related to blogging or writing, the ideal note workflow is one which lets you keep a bunch of throwaway ideas and also somehow have access to them in a coherent manner. This will be a long post, and it is a work-in-progress, so, keep that in mind. Since this is mainly me[^fn:1] work-shopping my technique, the philosophy will come in a later post probably. This workflow is documented more sparsely in my config file here, in the `noteYoda` section[^fn:2]. Some parts of this post also include mini video clips for clarity[^fn:3].

The entire workflow will end up being something like this[^fn:4]:

Concept

While working through ideas, it actually was more useful to describe the workflow I want, and then implement it, instead of relying on the canned approaches of each package. So the basics of the ideology are listed below.

Reference Management

Reference management is one of the main reasons to consider a plain-text setup, and mine is no different. The options most commonly seen are:

Mendeley
: This is a great option, and the most mobile friendly of the bunch. Sadly, the price tiers aren't very friendly so I have to give it a hard pass.

Jabref
: This is fun, but really more of a per-project management system, but it works well for that. The fact that it is Java based was a major issue for me.

Zotero
: This is what I personally use and recommend[^fn:5]. More on that in a later post.

Notes

The idea is to be able to create notes for all kinds of content. Specifically, papers or books, along with webpages. This then requires a separate system for each which is described by:

Search Engine
: The search engine is key, both in terms of accessibility and scalability. It is assumed that there will be many notes, and that they will have a wide variety of content. The search interface must then simply allow us to narrow down our candidates in a meaningful manner.

Contextual Representation
: This aspect of the workflow deals with representations, which should transcend the usage of tags or categories. In particular, it would be nice to be able to visualize the flow of ideas, each represented by a

note.

Backlinks

In particular, by backlinks at this point we are referring to the ability to link to a `pdf` or a website with a unique key such that notes can be added or removed at will.

Storage

Not actually part of the workflow in the same way, since it will be handled at the system level, it is worth nothing, that in this workflow Zotero is used to export a master **bib** file and keeps it updated, while the notes themselves are version controlled[^fn:6].

The concepts above will be handled by the following packages.

Concept	Package	Note
Search	deft	Has a great interface
Context	org-roam	Allows the export of graphiz mindmaps
Backlinks	org-roam, org-ref, org-noter	Covers websites, bibliographies, and pdfs respectively

A key component in this workflow is actually facilitated by the fabulous `org-roam-bibtex` or ORB. The basic idea is to ensure meaningful templates which interpolate smoothly with `org-roam`, `org-ref`, `helm-bibtex`, and `org-capture`.

Basic Variables

Given the packages we will be using, some variable settings are in order, namely:

```
(setq
   org_notes (concat (getenv "HOME") "/Git/Gitlab/Mine/Notes/")
   zot_bib (concat (getenv "HOME") "/GDrive/zotLib.bib")
   org-directory org_notes
   deft-directory org_notes
   org-roam-directory org_notes
   )
```

Search

For the search setup, the `doom-emacs` `deft` setup, by adding `+deft` in my `init.el`, worked out of the box for me. For those who do not use `doom`[^fn:7], the following should suffice:

```
(use-package deft
  :commands deft
  :init
  (setq deft-default-extension "org"
        ;; de-couples filename and note title:
        deft-use-filename-as-title nil
        deft-use-filter-string-for-filename t
        ;; disable auto-save
        deft-auto-save-interval -1.0
        ;; converts the filter string into a readable file-name using kebab-case:
        deft-file-naming-rules
        '((noslash . "-")
          (nospace . "-")
          (case-fn . downcase)))
  :config
  (add-to-list 'deft-extensions "tex")
  )
```

For more about the `doom-emacs` defaults, check the Github repo. The other aspect of interacting with the notes is via the `org-roam` interface and will be covered below.

Bibliography

Since I will be using `org-ref`, it makes no sense to load or work with the `+biblio` module at the moment. Thus this section is actually `doom` agnostic. The basic tools of bibliographic management from the `emacs` end are the venerable `helm-bibtex` (repo here) and `org-ref` (repo here). In order to make this guide complete, I will also describe the Zotero settings I have.

Zotero

Without getting too deep into the weeds here, the basic requirements are:

- Zotero
- The better bibtex extension
- A WebDAV provider for sync (TurtleTech works well for this)

The idea is to then have one top level `.bib` file in some handy location which you will set up to sync automatically. To make life easier, there is a tiny recording of the next steps.

Helm-Bibtex

This venerable package is really good at interfacing with a variety of externally formatted bibliographic managers.

```
(setq
 bibtex-completion-notes-path "/home/haozeke/Git/Gitlab/Mine/Notes/"
 bibtex-completion-bibliography "/home/haozeke/GDrive/zotLib.bib"
 bibtex-completion-pdf-field "file"
 bibtex-completion-notes-template-multiple-files
 (concat
  "#+TITLE: ${title}\n"
  "#+ROAM_KEY: cite:${=key=}\n"
  "* TODO Notes\n"
  ":PROPERTIES:\n"
  ":Custom_ID: ${=key=}\n"
  ":NOTER_DOCUMENT: %(orb-process-file-field \"${=key=}\")\n"
  ":AUTHOR: ${author-abbrev}\n"
  ":JOURNAL: ${journaltitle}\n"
  ":DATE: ${date}\n"
  ":YEAR: ${year}\n"
  ":DOI: ${doi}\n"
  ":URL: ${url}\n"
  ":END:\n\n"
  )
 )
```

`doom-emacs` users like me might want to wrap the above in a nice `after! org-ref` expression, but it doesn't really matter.

Explanation

To break-down aspects of the configuration snippet above:

- The template includes the `orb-process-file-field` function to allow selecting the `pdf` to be used with `org-noter`
- The `file` field is specified to work with the `.bib` file generated by Zotero

- `helm-bibtex` allows for any of the keys in a `.bib` file to be used in a template, and an overly expressive one is more useful
- The `ROAM_KEY` is defined to ensure that cite backlinks work correctly with `org-roam`
- As I prefer to have one notes file per `pdf`, I have only configured the `bibtex-completion-notes-template-multiple-files` variable

Org-Ref

As discussed above, this just makes citations much more meaningful in `orgmode`.

```
(use-package org-ref
    :config
    (setq
         org-ref-completion-library 'org-ref-ivy-cite
         org-ref-get-pdf-filename-function 'org-ref-get-pdf-filename-helm-bibtex
         org-ref-default-bibliography (list "/home/haozeke/GDrive/zotLib.bib")
         org-ref-bibliography-notes "/home/haozeke/Git/Gitlab/Mine/Notes/bibnotes.org"
         org-ref-note-title-format "* TODO %y - %t\n :PROPERTIES:\n  :Custom_ID: %k\n  :NOTER_DOCUMENT: %F\n
:ROAM_KEY: cite:%k\n  :AUTHOR: %9a\n  :JOURNAL: %j\n  :YEAR: %y\n  :VOLUME: %v\n  :PAGES: %p\n  :DOI: %D\n
:URL: %U\n :END:\n\n"
         org-ref-notes-directory "/home/haozeke/Git/Gitlab/Mine/Notes/"
         org-ref-notes-function 'orb-edit-notes
    ))
```

An essential aspect of this configuration is just that most of heavy lifting in terms of the notes are palmed off to `helm-bibtex`.

Explanation

To break-down aspects of the configuration snippet above:

- The `org-ref-get-pdf-filename-function` simply uses the `helm-bibtex` settings to find the `pdf`
- The default bibliography and notes directory are set to the same location as all the `org-roam` files, to encourage a flat hierarchy
- The `org-ref-notes-function` simply ensures that, like the `helm-bibtex` settings, I expect one file per `pdf`, and that I would like to use my `org-roam` template instead of the `org-ref` or `helm-bibtex` one

Note that for some reason, the format specifiers for `org-ref` are **not** the keys in `.bib` but are instead, the following[^fn:8]:

```
In the format, the following percent escapes will be expanded.
%l   The BibTeX label of the citation.
%a   List of author names, see also `reftex-cite-punctuation'.
%2a  Like %a, but abbreviate more than 2 authors like Jones et al.
%A   First author name only.
%e   Works like %a, but on list of editor names.  (%2e and %E work as well)
It is also possible to access all other BibTeX database fields:
%b booktitle     %c chapter        %d edition    %h howpublished
%i institution   %j journal        %k key        %m month
%n number        %o organization   %p pages      %P first page
%r address       %s school         %u publisher  %t title
%v volume        %y year
%B booktitle, abbreviated          %T title, abbreviated
%U url
%D doi
%S series        %N note
%f pdf filename
%F absolute pdf filename
Usually, only %l is needed.  The other stuff is mainly for the echo area
display, and for (setq reftex-comment-citations t).
%< as a special operator kills punctuation and space around it after the
string has been formatted.
A pair of square brackets indicates an optional argument, and RefTeX
will prompt for the values of these arguments.
```

Indexing Notes

This part of the workflow builds on the concepts best known as the Zettelkasten method. More details about the philosophy behind `org-roam` is here.

Org-Roam

The first part of this interface is essentially just the `doom-emacs` configuration, adapted for those who don't believe in the dark side below.

```
(use-package org-roam
  :hook (org-load . org-roam-mode)
  :commands (org-roam-buffer-toggle-display
             org-roam-find-file
             org-roam-graph
             org-roam-insert
             org-roam-switch-to-buffer
             org-roam-dailies-date
             org-roam-dailies-today
             org-roam-dailies-tomorrow
             org-roam-dailies-yesterday)
  :preface
  ;; Set this to nil so we can later detect whether the user has set a custom
  ;; directory for it, and default to `org-directory' if they haven't.
  (defvar org-roam-directory nil)
  :init
  :config
  (setq org-roam-directory (expand-file-name (or org-roam-directory "roam")
                                             org-directory)
        org-roam-verbose nil  ; https://youtu.be/fn4jIlFwuLU
        org-roam-buffer-no-delete-other-windows t ; make org-roam buffer sticky
        org-roam-completion-system 'default
)

  ;; Normally, the org-roam buffer doesn't open until you explicitly call
  ;; `org-roam'. If `+org-roam-open-buffer-on-find-file' is non-nil, the
  ;; org-roam buffer will be opened for you when you use `org-roam-find-file'
  ;; (but not `find-file', to limit the scope of this behavior).
  (add-hook 'find-file-hook
    (defun +org-roam-open-buffer-maybe-h ()
      (and +org-roam-open-buffer-on-find-file
           (memq 'org-roam-buffer--update-maybe post-command-hook)
           (not (window-parameter nil 'window-side)) ; don't proc for popups
           (not (eq 'visible (org-roam-buffer--visibility)))
           (with-current-buffer (window-buffer)
             (org-roam-buffer--get-create)))))

  ;; Hide the mode line in the org-roam buffer, since it serves no purpose. This
  ;; makes it easier to distinguish among other org buffers.
  (add-hook 'org-roam-buffer-prepare-hook #'hide-mode-line-mode))

;; Since the org module lazy loads org-protocol (waits until an org URL is
;; detected), we can safely chain `org-roam-protocol' to it.
(use-package org-roam-protocol
  :after org-protocol)

(use-package company-org-roam
  :after org-roam
  :config
  (set-company-backend! 'org-mode '(company-org-roam company-yasnippet company-dabbrev)))
```

Once again, for more details, check the Github repo.

Org-Roam-Bibtex

The configuration required is:

```
 (use-package org-roam-bibtex
  :after (org-roam)
  :hook (org-roam-mode . org-roam-bibtex-mode)
  :config
  (setq org-roam-bibtex-preformat-keywords
   '("=key=" "title" "url" "file" "author-or-editor" "keywords"))
  (setq orb-templates
        '(("r" "ref" plain (function org-roam-capture--get-point)
           ""
           :file-name "${slug}"
           :head "#+TITLE: ${=key=}: ${title}\n#+ROAM_KEY: ${ref}

- tags ::
- keywords :: ${keywords}

\n* ${title}\n  :PROPERTIES:\n  :Custom_ID: ${=key=}\n  :URL: ${url}\n  :AUTHOR: ${author-or-editor}\n
:NOTER_DOCUMENT: %(orb-process-file-field \"${=key=}\")\n  :NOTER_PAGE: \n  :END:\n\n"

           :unnarrowed t))))
```

Where most of the configuration is essentially the template again. Like `helm-bibtex`, ORB allows taking arbitrary keys from the `.bib` file.

Org Noter

The final aspect of a `pdf` workflow is simply ensuring that every `pdf` is associated with notes. The philosophy of `org-noter` is best described here. Only minor tweaks should be required to get this working with `interleave` as well.

```
(use-package org-noter
  :after (:any org pdf-view)
  :config
  (setq
   ;; The WM can handle splits
   org-noter-notes-window-location 'other-frame
   ;; Please stop opening frames
   org-noter-always-create-frame nil
   ;; I want to see the whole file
   org-noter-hide-other nil
   ;; Everything is relative to the main notes file
   org-noter-notes-search-path (list org_notes)
   )
  )
```

Evidently, from my configuration, it appears that I decided to use org-noter over the more commonly described interleave because it has better support for working with multiple documents linked to one file.

Org-Protocol

I will only cover the bare minimum relating to the use of `org-capture` here, because eventually I intend to handle a lot more cases with orca. Note that this part of the workflow has more to do with using `org-roam` with websites than `pdf` files.

Templates

This might get complicated but I am only trying to get the bare minimum for `org-protocol` right now.

```
;; Actually start using templates
(after! org-capture
  ;; Firefox and Chrome
  (add-to-list 'org-capture-templates
               '("P" "Protocol" entry ; key, name, type
                 (file+headline +org-capture-notes-file "Inbox") ; target
                 "* %^{Title}\nSource: %u, %c\n #+BEGIN_QUOTE\n%i\n#+END_QUOTE\n\n\n%?"
                 :prepend t ; properties
                 :kill-buffer t))
  (add-to-list 'org-capture-templates
               '("L" "Protocol Link" entry
                 (file+headline +org-capture-notes-file "Inbox")
                 "* %? [[%:link][%(transform-square-brackets-to-round-ones \"%:description\")]]\n"
                 :prepend t
                 :kill-buffer t))
)
```

Conclusions

At this point, many might argue that since by the end, only one template is called, defining the rest were pointless. They would be right, however, this is just how my configuration evolved. Feel free to cannibalize this for your personal benefit. Eventually I plan to expand this into something with `org-journal` as well, but not right now.
