Continuous integration and TEX with Org-Mode

Rohit Goswami

Abstract

Virtual or cloud computational resources in the form of build servers on public `git` hosting servers have become increasingly common. Herein we discuss how these may be leveraged for providing an asynchronous distributed collaborative workflow.

Additionally, we demonstrate how the Org-Mode markup language can be used to provide a user-friendly point of contact for novices and experts alike, and includes data analysis techniques.

1 Introduction

Since the acquisition of ShareLaTeX, and the integration with publishing houses, Overleaf has increased its prominence in academic circles. This has been accelerated by the global pandemic [8]. In the interests of preventing the promulgation of commercial monopolies we investigate the usage of continuous integration (CI) services to generate documents on the fly. At the same time, for many data-intensive studies, rapid analysis and plotting during the ideation phase has become de rigueur. This has led to the rise of Jupyter notebooks which meld analysis and context together in executable formats and have been a major driver in the rapid adoption of Python in scientific circles [5]. This can be seen also in the trend in publishing which has recently veered towards the adoption of such data-driven executable content [1, 6].

In these scenarios, the usage of TEX is often dependent on the programming language and environment used; e.g., R provides an almost complete format for preparing reports (RMarkdown and `knitr`) which exports to TEX [9, 12]. Here, we demonstrate the flexibility of the `org-mode` markup language, which can be extended in a straightforward manner to replace Jupyter notebooks for data analysis in a transparent, language agnostic manner.

2 CI setup

We will consider here the specific examples which are intended to run on the free "GitHub Actions" CI infrastructure. Given that most build-bots are configured in a similar manner, the key concepts are applicable to other setups as well. In broad strokes, a `texlive` installation controlled by a minimal `profile` is used for TEX, since the default build systems tend to be pinned to older distributions (e.g., Ubuntu 18.04 or even Alpine Linux).

2.1 Emacs and Org

To ensure that TEX exports are feasible on a CI without provisioning an entire `emacs` configuration in each repository, the following listing contains the minimal execution setup script. Links to more complete versions are given in section 4.4.

Listing 1: Minimal Lisp execution script

```
(require 'package)
(setq package-check-signature nil)
(add-to-list 'package-archives
  '("melpa" . "https://melpa.org/packages/") t)
(package-initialize)
(unless package-archive-contents
   (package-refresh-contents))
(package-install 'use-package)
(package-install 'org)

(dolist (package '(use-package))
(unless (package-installed-p package)
   (package-install package)))

(use-package org-ref
   :ensure t)

(require 'ox-latex)
(setq org-latex-packages-alist 'nil
 org-latex-minted-options 'nil
 org-latex-listings 'minted
 org-latex-default-packages-alist
  '((""     "graphicx"  t)
    (""     "lipsum"  t)))

(defun org-export-to-pdf-dir (files)
  "Export␣all␣FILES␣to␣latex."
  (interactive "ORG-->TEX")
  (save-excursion
    (let ((org-files-lst ))
      (dolist (org-file files)
        (message "***␣Exporting␣file
␣␣␣␣␣␣␣␣%␣s␣***" org-file)
        (find-file org-file)
        (org-latex-export-to-latex)
        (kill-buffer)))))

;; Export all org files from CLI
(org-export-to-pdf-dir argv)
```

This can now be executed on a CI as seen in Listing 2.

2.2 Caching and determinism

For continuous integration, control over dependencies is of paramount importance. In this workflow,

doi.org/10.47397/tb/42-2/tb131goswami-ci-org

Listing 2: Using Lisp exporter on the CI

```
- name: Generate TeX
  run: |
 emacs -q -nl -script \
 scripts/org2tex.el src/super.org
```

the primary dependency considered is the TEX installation. There have been many attempts to ensure reproducibility and automation of installation, ranging from language-specific measures like `tinytex` for R [11] to the alternate TEX-compatible systems like Tectonic [7]. There have also been recent CI-specific projects like the Island of TEX which provide Docker images for ensuring reproducibility [3]. Our approach to this problem is to leverage the standard TEX Live installation approach, along with caching to prevent unnecessary strain on the build servers. Packages are obtained on-the-fly from the Comprehensive TEX Archive Network (CTAN) via the `texliveonfly` package, which is a Python script for dependency resolution; it uses `tlmgr` to install packages needed for a given compilation.

3 GitHub bots

The workflow in the previous section can be augmented to use "Bots" to perform additional automated tasks. Beyond the convenience factor, the usage of GitHub allows for using the embedded rich text editor, and even Codespaces. Rendered documents can also be viewed on GitHub, which is a bonus. This section will assume the usage of GitHub as the platform of choice.

Other "Bots" not mentioned here include running `latexdiff` on pull requests to enhance the collaborative workflow.

3.1 Branches and deployments

To "deploy" the rendered document in a manner which is accessible from the web interface, a PDF may be deployed to an orphan branch which will be recreated each time, without history so as to not take up excessive space. Though this may be accomplished by manually adding a commit step, `actions-gh-pages` makes this even simpler, as shown in the Listing 3 fragment.

Pushing to a branch requires a personal access token. However, since each GitHub Actions runner automatically creates the `GITHUB_TOKEN` secret for the authentication of the workflow itself, this bot abstracts the authentication mechanism from the user.

Listing 3: PDF rendering fragment on CI

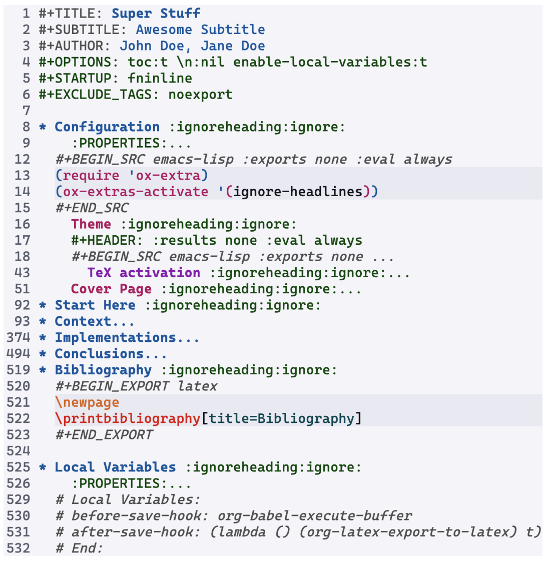
```
- name: Render to Branch
  uses: peaceiris/actions-gh-pages@v3
  with:
    github_token: |
        ${{ secrets.GITHUB_TOKEN }}
    publish_dir: ./pdfdir
    publish_branch: pdf
    force_orphan: true
```

4 Org markup and TEX

Generating a TEX layout with `org-mode` demands an understanding of the standard layout of LATEX documents, and also the Emacs Lisp variables and hooks which control and modify the export process from `org` to TEX. The `org-mode` syntax is described in great detail in its manual at `orgmode.org/manual/`. Furthermore, syntax highlighting of `org-mode` is provided by extensions to most editors, including Visual Studio Code. Finally, `org-mode` is rendered for HTML previews on GitHub, GitLab, and other servers, making its use and adoption easier.

4.1 Standard modifications

The basic setup for working with `org-mode` files in general is depicted in Figure 1, where importantly, the `:ignore:` tags can be used to demarcate the document into logically consistent parts. With code folding and narrowing to trees, this allows for an optimally efficient setup.



```
#+TITLE: Super Stuff
#+SUBTITLE: Awesome Subtitle
#+AUTHOR: John Doe, Jane Doe
#+OPTIONS: toc:t \n:nil enable-local-variables:t
#+STARTUP: fninline
#+EXCLUDE_TAGS: noexport

* Configuration :ignoreheading:ignore:
    :PROPERTIES:...
  #+BEGIN_SRC emacs-lisp :exports none :eval always
  (require 'ox-extra)
  (ox-extras-activate '(ignore-headlines))
  #+END_SRC
    Theme :ignoreheading:ignore:
    #+HEADER: :results none :eval always
    #+BEGIN_SRC emacs-lisp :exports none ...
      TeX activation :ignoreheading:ignore:...
    Cover Page :ignoreheading:ignore:...
* Start Here :ignoreheading:ignore:
* Context...
* Implementations...
* Conclusions...
* Bibliography :ignoreheading:ignore:
  #+BEGIN_EXPORT latex
  \newpage
  \printbibliography[title=Bibliography]
  #+END_EXPORT

* Local Variables :ignoreheading:ignore:
    :PROPERTIES:...
  # Local Variables:
  # before-save-hook: org-babel-execute-buffer
  # after-save-hook: (lambda () (org-latex-export-to-latex) t)
  # End:
```

Figure 1: Standard document layout with hook and code folding

Listing 4: Sample demonstrating the usage of a "super" class

```
#+HEADER: :eval always :results none
#+BEGIN_SRC emacs-lisp :exports none
(org-babel-tangle)
(add-to-list 'org-latex-classes
'("super" "\\documentclass{super}"
("\\part{%s}" . "\\part*{%s}")
("\\chapter{%s}" . "\\chapter*{%s}")
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" .
 "\\subsection*{%s}")
("\\subsubsection{%s}" .
 "\\subsubsection*{%s}")
("\\paragraph{%s}" .
 "\\paragraph*{%s}")
("\\subparagraph{%s}" .
 "\\subparagraph*{%s}")))
(setq org-latex-packages-alist 'nil)
(setq org-latex-minted-options 'nil)
#+END_SRC
```

Listing 5: Snippet of a custom class

```
#+header: :results none
#+header: :tangle super.cls
#+header: :exports none
#+begin_src latex :eval always
....
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{super}
....
#+end_src
```

4.2 Archive emulation

Perhaps the simplest approach is to leverage the `.org` file as a plain text archive, which expands to a layout defined by `#+begin_export latex` and `#+end_export` directives for layouts in the document itself. For the preamble and to define what traditionally falls under the purview of `.cls` files, it is easiest to literally export the TEX commands to a file before loading it.

To accomplish this, it is necessary to provide an Emacs Lisp snippet which disables much of the underlying machinery, while augmenting it with the desired `.cls` as seen in Listing 4.

While the class itself can be within the same file, as shown in Listing 5, the `:eval always` header and the `org-babel-tangle` call in Listing 4 ensures that the file is (re)generated during the export process.

Listing 6: Mathematica code to export an image (solutions for a hyperbolic equation)

```
ClearAll[u, x, t, p];
len = 1;
pde = D[u[x, t], t, t] == \
16*D[u[x, t], x, x];
(* initial conditions *)
ic = {u[x, 0] == Sin[Pi*x], \
Derivative[0, 1][u][x, 0] == 0};
(* boundary conditions *)
bc = {u[0, t] == 0, u[len, t] == 0};
sol = u[x, t] /. \
First@DSolve[{pde, ic, bc}, \
 u[x, t], {x, t}];
p = Plot3D[sol, {x, 0, 1}, \
{t, 0, 1}, PlotPoints -> 30];
Export["images/q2aM.png",p];
Print["images/q2aM.png"]
```

4.3 Data driven development

The previous sections used Lisp to configure the Org and LATEX configuration in a reproducible manner. The `org-babel` tangling works for arbitrary languages as well, with minor workflow modifications. Consider the Mathematica code in Listing 6.

The key element in Listing 6 from the perspective of the end user is that the image must exist on disk, and the filename must be returned from the code block for it to be picked up in the `orgmode` workflow. This has the added benefit of being more WYSIWYG (what you see is what you get) compared to Jupyter Notebooks or RStudio which often render images which must be saved to disk manually with additional configurational changes.

4.4 Workflow samples

For full examples of a report generated with a CI-based collaborative workflow, interested readers are referred to:
`github.com/HaoZeke/ipam21_tqc_wg_report`
or slides for TUG 2021 at:
`github.com/HaoZeke/haozeke.github.io/blob/src/presentations/TUG2021/tug21rgpres.org`
and for a more involved example that showcases `org-mode` and its integration with R, see:
`github.com/HaoZeke/haozeke.github.io/blob/src/content-org/solutions/SR2/sol03.Rorg`

5 Discussion

The workflow outlined in this short article is not without its rough edges. The content presented in this article are aimed at groups of collaborators with at least one intermediate user of both the Unix shell and `git` to set up the initial workflow. It is expected, with the adoption of Wizards on Windows [4] along with template repositories [2] and GitHub Codespaces, that this workflow will have an even lower barrier of entry. In its current form, it is expected to be of interest to asynchronously communicating authors who are unwilling or unable to leverage solutions in commercial domains or those invested in the `org-mode` ecosystem.

The CI aspects are independent of the underlying method used to generate the TEX, and plain TEX workflows are also supported with ease, along with alternative markup methods like the R ecosystem's `bookdown` [10], or even other TEX generation techniques using `pandoc` are viable approaches.

However, the utility of using `org-mode` cannot be discounted, given that variables are retained between code blocks during execution. This means that programming languages may be freely mixed in an `org` document and tangled and executed subsequently by `org-babel` while retaining desired TEX layouts. Additionally, `org-mode` is "closest to bare metal" in that there are far fewer defaults compared to other approaches.

6 Conclusions

We have outlined a mechanism by which task-specific ad hoc TEX templates can be generated from `org-mode` files, thus enhancing collaboration with non-TEXnical colleagues. This allows for a transparent workflow for data analysis and programming via `org-mode` extensions such as the `babel` ecosystem.

Additionally, the use and abuse of build servers for collaborative TEX editing has been explored. Avenues for further development of such cloud-based distributed collaboration mechanisms have also been identified. The CI discussion can also be extended to encompass the automated testing of CTAN packages themselves, which would aid package developers.

6.1 Acknowledgments

The author would like to thank the attendees of TUG 2021 for fruitful discussions which enhanced the article.

References

[1] C. Davidson-Pilon. *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference.* Addison-Wesley, 2016.

[2] GitHub Docs. Creating a repository from a template. `https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/creating-a-repository-on-github/creating-a-repository-from-a-template`

[3] Island of TEX. The Island of TEX: Developing abroad, your next destination. *TUGboat* 41(2):182–184, 2020. `https://tug.org/TUGboat/tb41-2/tb128island.pdf`

[4] Microsoft. Wizards - Win32 apps. `https://docs.microsoft.com/en-us/windows/win32/uxguide/win-wizards`

[5] T.E. Oliphant. Python for Scientific Computing. *Computing in Science Engineering* 9(3):10–20, May 2007. `10/fjzzc8`

[6] R. Peverati. Fitting elephants in the density functionals zoo: Statistical criteria for the evaluation of density functional theory methods as a suitable replacement for counting parameters. *International Journal of Quantum Chemistry* 121(1):e26379, 2020. `10.1002/qua.26379`

[7] Tectonic typesetting system, Aug. 2021. `https://tectonic-typesetting.github.io`

[8] B. Veytsman. Using Overleaf for collaborative projects: First impressions and lessons learned. *TUGboat* 41(2):179–181, July 2020. `https://tug.org/TUGboat/tb41-2/tb128veytsman-overleaf.pdf`

[9] Y. Xie. *Dynamic Documents with R and Knitr.* CRC Press, Boca Raton, FL, second edition ed., 2015.

[10] Y. Xie. *Bookdown: Authoring Books and Technical Publications with R Markdown.* CRC Press, Boca Raton, FL, 2017.

[11] Y. Xie. TinyTeX: A lightweight, cross-platform, and easy-to-maintain LATEX distribution based on TEX Live. *TUGboat* 40(1):30–32, 2019. `https://tug.org/TUGboat/tb40-1/tb124xie-tinytex.pdf`

[12] Y. Xie, J.J. Allaire, G. Grolemund. *R Markdown: The Definitive Guide.* CRC Press, Boca Raton, FL, 2019.

◇ Rohit Goswami
Science Institute
& Faculty of Physical Sciences
University of Iceland VR-III
107 Reykjavík, Iceland
& Quansight Labs
`rog32 (at) hi dot is`
`https://rgoswami.me`
ORCID 0000-0002-2393-8056