Editing published documents

Ahmad Ammar

16 November 2026

Contents

1 The editing cycle **3**

1.1 Four questions that tell projects apart . . . 3

1.2 Why the answers matter . . . 3

2 A teaching site built with Sphinx **4**

2.1 Source and tools . . . 4

2.2 Build and publish . . . 4

2.3 Editing after publication . . . 4

2.4 What it does well . . . 4

3 How other projects do it **6**

3.1 Quarto books: R for Data Science and Geocomputation with Python . . . 6

3.2 Jupyter Book and MyST: The Turing Way . . . 6

3.3 Manubot: papers written in the open . . . 6

3.4 The Carpentries Workbench: lessons . . . 6

3.5 LaTeX with a hosted editor . . . 7

3.6 Side by side . . . 7

4 Whole workflows, step by step **8**

4.1 A Sphinx teaching site (Chapter 2) . . . 8

4.2 A Quarto book . . . 8

4.3 A Jupyter Book or MyST project . . . 9

4.4 A Manubot manuscript . . . 9

4.5 A LaTeX paper or thesis . . . 9

4.6 Our stack . . . 9

4.7 Practices nearly everyone shares . . . 10

5 Our stack, compared **11**

5.1 Where we are ahead . . . 11

5.2 What the site does now . . . 11

5.3 Still to do . . . 12

6 What to adopt **13**

6.1 Small and worth doing now . . . 13

6.2 Larger, worth planning . . . 13

6.3 What not to copy . . . 13

7 How to edit our documents **14**

7.1 From a published page . . . 14

7.2 Studio . . . 14

7.3 VS Code . . . 14

7.4 From the command line . . . 15

1 The editing cycle

Every published document, whatever tool made it, goes round the same loop. There is a **source**: the files a person writes, in a plain-text format such as Markdown, reStructuredText or LaTeX. A **build** turns the source into what readers see: HTML pages, a PDF, or both. A **publish** step copies the built files somewhere public. Editing after the first release means going round the loop again, and the differences between projects are mostly about who does each step, where, and how much of it is automatic.

1.1 Four questions that tell projects apart

1. **What is the source format?** Markdown is the most common: easy to read and write, limited for mathematics and cross-references. MyST Markdown and Quarto Markdown extend it with figures, equations, citations and cross-references. reStructuredText is the older native format of Sphinx. LaTeX is the most expressive for mathematics and print, and the hardest to turn into good web pages.

2. **Where is it edited?** A desktop editor (VS Code, RStudio, a LaTeX editor), a browser editor on GitHub, or a hosted service such as Overleaf.

3. **Where is it built?** On the author's computer, or by a continuous integration (CI) service such as GitHub Actions each time a change is pushed.

4. **How does a change reach readers?** By pushing built files to a branch that GitHub Pages serves, by letting CI build and deploy, or by a documentation host such as Read the Docs or Netlify building from the repository.

1.2 Why the answers matter

A local build is simple and works offline, but the published site is only as current as the last time someone remembered to build it, and it depends on one computer's setup. A CI build runs the same way every time from a written recipe, which is what makes it reproducible, and it can build a *preview* of every proposed change before it is accepted. The cost is that everything the build needs (for us: TeX, pandoc, Python packages, the data behind figures) must be installable on a clean machine.

The format decides how far a non-specialist can take part. A typo fix in Markdown can be made in the GitHub web editor by anyone; the same fix in a LaTeX chapter with custom macros is easy too, but checking that it still builds is not, unless CI builds it.

2 A teaching site built with Sphinx

Bob McNabb's site, iamdonovan.github.io, holds the practicals for several Ulster modules (EGM101, EGM310, EGM702, EGM703, EGM722) alongside research and writing pages. Its repository is public, github.com/iamdonovan/iamd so how it is made can be read directly. Everything below was taken from that repository in November 2026.

2.1 Source and tools

The source sits in a `docs` folder: 188 reStructuredText (`.rst`) files and one Python file, `conf.py`, which configures Sphinx. A practical is one page; the SAR practical for EGM703 Week 3, for example, is `docs/teaching/egm703/` with its 27 screenshots in `docs/img/egm703/week3/` and the data in `docs/files/EGM703_Week3.zip`.

The build environment is a conda file, `environment.yml`, naming Sphinx extensions: `sphinx-book-theme` for the look (left navigation, contents panel, search, light and dark mode), `sphinxcontrib-bibtex` for references from a `refs.bib`, `sphinx-design` for cards and tabs, and `sphinx-copybutton` for copying code. The site is licensed CC BY-SA 4.0.

2.2 Build and publish

There is no CI workflow. The `Makefile` builds into a folder beside the repository (`../../iamdonovan-sphinx-docs`), which is a checkout of the `gh-pages` branch; GitHub Pages serves that branch. Publishing is therefore: edit the `.rst`, run `make html`, commit and push the built pages on `gh-pages`, and commit the source on `main`. The same `Makefile` has a `latexpdf` target, so a PDF can come from the same source through Sphinx's LaTeX writer.

2.3 Editing after publication

The theme is configured with `use_repository_button`, `use_issues_button` and `use_edit_page_button`. Every page therefore carries three icons: one to the repository, one to open an issue (so a student can report a broken step), and one to edit that page's source on GitHub. The commit history shows the rhythm: small, frequent, plainly described changes (*update instructions for spss version 31*, *re-order teaching links*, *add carrivick LIA paper*).

2.4 What it does well

- One plain-text source per page, under version control, with a written environment.
- Edit and issue buttons on every page, which invite corrections from readers.
- A theme that gives navigation, search and a contents panel with no custom code.
- Course materials split out: each module's code and data have their own repository (for example

egm722), and the research packages (for example spymicmac) document themselves on Read the Docs from a `.readthedocs.yaml`, so their documentation is rebuilt by the host on every push.

3 How other projects do it

The projects below are open: their repositories show exactly how they are built. They were chosen because each does one part of the cycle unusually well. Repository contents were checked in November 2026.

3.1 Quarto books: R for Data Science and Geocomputation with Python

R for Data Science is written as one Quarto (`.qmd`) file per chapter, configured by `_quarto.yml`. A single GitHub Actions workflow, `build_book.yaml`, renders and publishes it. Results of code are stored in a committed `_freeze` folder, so CI does not have to rerun every computation for each edit; it reruns only chapters whose code changed. Geocomputation with Python goes further: separate workflows build the book on every push (`main.yaml`), build a preview for every pull request (`pr.yml`, `netlify.yml`), and the repository has a `.devcontainer.json` and `.gitpod.yml`, so a contributor can open a ready-made environment in the browser with everything installed. It also carries a `CITATION.bib`.

Quarto renders the same source to HTML and to PDF through LaTeX (Posit, 2026).

3.2 Jupyter Book and MyST: The Turing Way

The Turing Way, a community handbook on reproducible research, is a Jupyter Book written in MyST Markdown. Its CI is a set of small, single-purpose checks: `external_link_check.yml` (with a `lychee.toml`) finds broken links; `accessibility-alt-text-bot.yml` asks for alternative text on images; `validate_citation.yml` checks the `CITATION.cff`; `release-drafter.yml` and `.zenodo.json` turn releases into citable, archived versions with DOIs; `test-file-size.yml` stops oversized images. Netlify builds a preview of every pull request, at an address made from the pull request's number (`deploy-preview-<number>--the-turing-way.netlify`, removed once the pull request closes), so reviewers read the changed book, not a diff. Contributors are credited through `.all-contributorsrc`.

Jupyter Book 2 is built on the MyST engine, which exports PDFs through either LaTeX or Typst from the same source (Executable Books, 2026).

3.3 Manubot: papers written in the open

Manubot Rootstock is a template for writing a scholarly manuscript in Markdown on GitHub. Each push runs `manubot.yaml`, which rebuilds the manuscript as HTML and PDF and publishes it. Citations are written as identifiers, for example `@doi:10.1371/journal.pcbi.1007128`, and Manubot fetches the reference metadata itself (Himmelstein et al., 2019). Every version of the manuscript has its own permanent web address.

3.4 The Carpentries Workbench: lessons

The Carpentries lesson template keeps lessons as Markdown episodes with a `config.yaml`. Its workflows build the lesson in a container (`docker_build_deploy.yaml`), check pull requests before they are built (`pr-preflight.yaml`) and comment on each pull request with what changed in the rendered lesson (`pr-comment.yaml`). A `.devcontainer` gives a ready environment, and translations are managed with Crowdin.

3.5 LaTeX with a hosted editor

Many people who write in LaTeX use Overleaf, a browser editor with a live PDF beside the source and a rich-text view. Overleaf can sync a project with a GitHub repository on its paid plans; the source then lives in git and other tools can build from it. It produces a PDF, not web pages: a web version needs a separate converter.

3.6 Side by side

Project	Source	Built by	Preview of changes	After publication
Sphinx site (Ch. 2)	reStructuredText	author's computer	local build	edit button, issues button, rebuild and push
R for Data Science	Quarto	GitHub Actions	local render	push; CI rebuilds; frozen results
Geocomputation with Python	Quarto	GitHub Actions	Netlify per pull request	push or pull request; dev container
The Turing Way	MyST Markdown	CI and Netlify	Netlify per pull request	pull requests checked for links, alt text, file size; DOI releases
Manubot	Markdown	GitHub Actions	each commit is a version	push; citations by DOI
Carpentries	Markdown	GitHub Actions (container)	comment on pull request	pull request; preflight checks
Overleaf	LaTeX	Overleaf	live PDF	edit online; optional git sync (paid)

Table 3.1: How the projects in this chapter handle the editing cycle.

4 Whole workflows, step by step

The previous chapters compared tools. This one follows a working day: where a person opens the text, how they see the result while writing, what they type to publish, and what they do the next time something needs changing. Where a step is read from a public repository it says so; where it comes from a tool's own documentation, that is named instead.

4.1 A Sphinx teaching site (Chapter 2)

Read from the repository: the `.gitignore` was generated for PyCharm, so the source is most likely edited in a JetBrains IDE; the built site lives in a second checkout beside the source; and each commit on the published branch repeats the message of the source commit.

1. Open the `docs` folder in the editor and change a page, for example `teaching/egm703/practicals/week3.rst`. Screenshots go in `docs/img/egm703/week3/`.
2. Build: `make html`, which runs `sphinx-build` and writes the pages into the second checkout (`../../iamdonovan-` the `gh-pages` branch). Open the built page in a browser to check it.
3. Commit the source on `main` and push.
4. In the second checkout, commit the rebuilt pages with the same message and push. GitHub Pages serves them within a minute.
5. Next time: the same four steps. A reader who spots a problem can use the issue button on the page; the edit button takes the author straight to the source file on GitHub.

Two commits per change, both made by hand, is the cost of not having CI; the benefit is that nothing but Sphinx needs to be installed anywhere.

4.2 A Quarto book

From the Quarto documentation (Posit, 2026) and the R for Data Science repository.

1. Open the project in RStudio or VS Code (both have Quarto extensions). Each chapter is a `.qmd` file listed in `_quarto.yml`.
2. Run `quarto preview`. A local server opens the book in the browser and re-renders the chapter each time it is saved.
3. Commit and push to `main`. The workflow in `.github/workflows` renders the whole book on a clean machine and publishes it to GitHub Pages; nothing built is committed by hand. Code results already in `_freeze` are reused.
4. For a larger change: a branch and a pull request. In Geocomputation with Python, that pull request is built into a preview site before it is merged.

4.3 A Jupyter Book or MyST project

From the Jupyter Book and MyST documentation (Executable Books, 2026).

1. Write MyST Markdown (`.md`) or notebooks (`.ipynb`) in VS Code or JupyterLab; the table of contents is in `myst.yml`.
2. Run `jupyter book start` (or `myst start`): a live local server that updates as files are saved.
3. Push; CI builds the HTML (and, if configured, a PDF through LaTeX or Typst) and deploys. In The Turing Way, every pull request also runs link, alt-text and file-size checks and gets a Netlify preview.

4.4 A Manubot manuscript

From the Rootstock template (Himmelstein et al., 2019).

1. Edit the Markdown files in `content/`, in any editor or directly in GitHub's web editor. Cite by identifier, for example `@doi:10.1371/journal.pcbi.1007128`.
2. Commit and push (or open a pull request so co-authors can review the change line by line).
3. The workflow `manubot.yaml` builds HTML and PDF on GitHub and publishes them; each build keeps its own permanent address, so earlier versions can still be cited.

Writing happens entirely in git: review comments, suggested edits and authorship all come from GitHub.

4.5 A LaTeX paper or thesis

The most common setups among people who write LaTeX directly:

- **VS Code with LaTeX Workshop.** Saving the file runs `latexmk`, the PDF sits in a tab beside the source, and SyncTeX links the two: Ctrl+click in the PDF jumps to the source line, and a command jumps the other way. The project is a git repository; the author commits and pushes when a section is done. Changes for co-authors or reviewers are shown with `latexdiff old.tex new.tex > diff.tex`.
- **Overleaf.** The same edit-and-preview loop in the browser, with comments, tracked changes and a rich-text view for co-authors who do not write LaTeX. Git sync is on paid plans.
- **TeXstudio or TeXmaker.** Desktop editors with a built-in PDF viewer and SyncTeX.

In each case the output is a PDF. Web pages, when wanted, come from a separate tool, which is the gap our stack fills.

4.6 Our stack

1. Open the document: from any page's Admin bar (Studio, VS Code or browser), or `python -m research_helpers.stack studio`.

2. Write in the LaTeX or in the rendered preview; Studio re-renders as you type. For the exact PDF, Ctrl+Shift+B in VS Code.

3. Publish: *Save and publish* in Studio, or `python build_guides.py <guide>` then commit and push. The build makes the PDF and the web pages together; both, with the source, are committed to `main`, and the site's workflow deploys them.

4. Next time: the same, from the page you want to change.

4.7 Practices nearly everyone shares

- **A live preview while writing**: `quarto preview`, `jupyter book start`, `sphinx-autobuild`, LaTeX Workshop on save.
- **Built files are never edited.** Changes go into the source and the site is rebuilt.
- **Built files kept out of the source history**, either on a separate branch (the Sphinx site) or produced fresh by CI (Quarto, MyST, Manubot). We commit built pages beside the sources, which makes our history larger and our diffs noisier; building in CI would remove that.
- **Small commits with plain messages** for routine fixes, straight to `main`.
- **A branch and a pull request** for larger changes or other people's contributions, ideally with a preview build to read before merging.
- **A written environment** (`environment.yml`, `DESCRIPTION`, a dev container) so the build is the same on every machine.

5 Our stack, compared

Our guides, essays and the Atmosphere handbook are written in LaTeX and built by one shared pipeline, the `stack` in research-helpers. Each document has a `document.toml` naming its chapters. From the same chapter files the stack makes the PDF (through LaTeX) and the web pages (through pandoc and the Zensical site builder), with one shared look across the site. Notebooks written as plain Python files are run where their data is, and their last run is kept in the guide's `frozen` folder.

Since November 2026 GitHub builds the whole site from these sources on every push, as R for Data Science does: the repository holds sources only, and the built pages go straight to GitHub Pages.

5.1 Where we are ahead

- **One LaTeX source for print and web.** Most projects start from Markdown and make the PDF second. Starting from LaTeX keeps mathematics, cross-references and print layout exact, and the web version follows.
- **Results come from code.** Numbers, tables and figures in the text are generated by the notebooks, not typed, and frozen runs let a clean machine rebuild them without the data.
- **Editing tools.** Every page's Admin bar opens that chapter's `.tex`, in VS Code with a PDF preview, in the browser, or in Studio, which shows the whole document with each chapter beside its rendering and lets plain paragraphs be edited in the rendering itself.
- **Publishing guard.** Anything listed in `.stack-hold` is never built or published.

5.2 What the site does now

- **Built on GitHub from sources.** `.github/workflows/site.yml` installs pandoc, TeX Live, Poppler and the stack on a clean Ubuntu machine, builds every guide's PDF and pages and the site's own pages, and deploys the result. A build takes about three minutes. An edit made anywhere, including GitHub's web editor, goes live by itself.
- **The same PDFs everywhere.** The stack ships open-licence fonts (Caladea, metric-compatible with Cambria, and Inconsolata) and loads them by path, so a PDF does not depend on fonts installed on one computer.
- **Proposed changes are built first.** A pull request runs the same build and checks without publishing.
- **Checks on every build.** Internal links; images without alternative text, images over 800 KB and pages without a heading (`stack audit`); and every outside link (lychee), reported in the run summary.
- **Upkeep that announces itself.** A weekly scheduled build shows a broken toolchain or a dead link before an edit needs it, and Dependabot proposes updates to the workflow's actions each month.

- **Report a problem.** Every page links to a new GitHub issue on its document's repository, filled in with the page's title and address.
- **A written environment.** `requirements-site.txt`, `environment-stack.yml` and `python -m research_helpers.stack doctor` record exactly what a build needs.
- **Citable.** A `CITATION.cff` names how to cite the materials.

5.3 Still to do

- **Preview sites for proposed changes**, as The Turing Way has, need a host that serves one copy per pull request (Netlify or Cloudflare Pages, both free for this use) and an account there.
- **Archived releases with DOIs** need the repository connected to Zenodo once from a Zenodo account; tagged releases are then archived automatically.
- **The Atmosphere handbook** is built from a private source and committed as built, so it is the one part that still needs the laptop.

6 What to adopt

Ranked by value for the effort. Each item names where it comes from.

6.1 Small and worth doing now

1. **A "report a problem" link on every page** (Sphinx book theme's issues button). A link that opens a GitHub issue already filled in with the page's address and title. Readers, including students, can then flag a broken step without email.

2. **External link checking in CI** (The Turing Way's lychee workflow). A weekly and per-push job that lists dead links. It needs no TeX, so it is cheap.

3. **Alternative text and file-size checks** (The Turing Way). A build check that every figure has a caption or alt text and that no image is larger than a set size. This puts the accessibility rule we already follow into code.

4. **A written build environment** (the Sphinx site's `environment.yml`, Geocomputation's dev container). One file listing exactly what the build needs, so another machine, or CI, can build the site.

6.2 Larger, worth planning

5. **Build the guides in CI** (R for Data Science, Manubot, the Carpentries). A GitHub Actions job with TeX Live, pandoc and the stack. Browser edits would then publish themselves, and the site would no longer depend on one laptop. Heavy notebooks can keep committed outputs, as Quarto's `_freeze` does.

6. **Previews of proposed changes** (Geocomputation, The Turing Way). Once CI builds, a pull request can build into a separate preview address, so a change can be read as a page before it goes live.

7. **Versioned, citable releases** (The Turing Way with Zenodo). A tagged release of a guide gets a DOI and a `CITATION.cff`, matching how our data and code are released.

8. **Moving the remaining long documents onto the stack.** The Marxian pamphlet and the EGM305 handbook still have their own build scripts; with a `document.toml` each they would gain the same editing tools.

6.3 What not to copy

Moving from LaTeX to Markdown would make browser editing simpler, but would lose the exact mathematics and print layout that are the point of our sources, and would break the rule that long documents have LaTeX as their one source. MyST and Quarto can both read some LaTeX mathematics, but not our custom environments and macros. The better path is to keep LaTeX and add the checks and CI that these projects show are possible.

7 How to edit our documents

7.1 From a published page

Turn on the Admin bar once in a browser by adding `?admin=on` to any page address (`?admin=off` hides it). At the bottom of every page of a guide it then offers:

- **All chapters in Studio**: the whole document in the local editor (below), opened at the chapter you were reading. Studio must be running.
- **Edit ... in VS Code**: that chapter's `.tex` file on this computer.
- **Edit in browser**: the same file in GitHub's web editor. The change is committed to the repository, but the page changes only after a rebuild.

7.2 Studio

Start it with

```
python -m research_helpers.stack studio
```

and open `http://127.0.0.1:8770/`. It lists every document with a `document.toml`. A document opens as one long page: each chapter's LaTeX on the left, its rendering on the right.

- Type in the LaTeX, and the rendering follows after a pause.
- Click a plain paragraph or heading in the rendering and type there. When you leave it, that paragraph alone is written back into the LaTeX. Mathematics, citations and references show as grey chips that are carried over unchanged. Paragraphs with environments or equations are edited in the LaTeX: double-click them to jump there.
- Click a figure to replace its image or edit its caption. *Add figure*, or dropping an image on the LaTeX, saves the image beside the others (never over one) and inserts a figure at the cursor.
- Ctrl+S saves the changed chapters. It refuses if a file was changed elsewhere in the meantime.
- *Save and publish* rebuilds the document, commits what the rebuild changed, and pushes. Documents listed in `.stack-hold` are never published.

The rendering is a quick approximation. For the exact PDF, use the VS Code task below.

7.3 VS Code

Run once per repository:

```
python -m research_helpers.stack editor .
```

Then, in any chapter, Ctrl+Shift+B builds the document's PDF into `build/preview/`, and the task *Rebuild this document's web pages* rebuilds the site's pages for it. LaTeX lines wrap in the editor.

7.4 From the command line

```
python -m research_helpers.stack preview path/to/chapter.tex       # PDF
python -m research_helpers.stack preview path/to/chapter.tex web   # web pages
python build_guides.py <guide>                                     # the site's own build
```

References

Executable Books. Creating PDF documents, 2026. URL https://mystmd.org/guide/creating-pdf-documents. Accessed 13 November 2026.

Daniel S. Himmelstein, Vincent Rubinetti, David R. Slochower, Dongbo Hu, Venkat S. Malladi, Casey S. Greene, and Anthony Gitter. Open collaborative writing with Manubot. *PLOS Computational Biology*, 15(6):e1007128, 2019. doi:10.1371/journal.pcbi.1007128.

Posit. Quarto: publishing to GitHub Pages, 2026. URL https://quarto.org/docs/publishing/github-pages.html. Accessed 13 November 2026.