

Table of Contents

1. Publishing overview	2
1.1 Choose your starting point	2
1.2 Choose your features	3
1.2.1 Authoring extensions	3
1.2.2 Lifecycle management tooling	4
1.2.3 Publishing outputs	5
1.3 Configure Prodockit features	5
1.3.1 What is <i>not</i> an extension	6
1.4 Follow the publishing path	6
1.5 Know which output you are checking	8
1.6 Build with revision dates	9
1.6.1 Use the command without adoption	9
1.6.2 Use the command without Git	10
1.6.3 Understand dates in a Git project	11
1.7 Use the detailed guides when needed	11

1. Publishing overview

Publishing turns the site you previewed locally into outputs other people can open: a website on GitHub Pages or GitLab Pages and, when the project needs one, a downloadable PDF.

This section is for a document author with a working local project. It covers template updates, continuous integration (CI), automated Pages deployment, and checks on the published result. For a new project, first choose [an installation path](#), including the manual installation route when every setup action must be performed directly. Maintainers changing the prodockit package itself should use [Maintain prodockit](#).

Start here after [building your first site](#). This section uses one repeatable publishing workflow: prepare the project, build both outputs, test what was built, push the source, and verify what the hosting service actually serves.

1.1 Choose your starting point

[Table 8.1](#) directs each project state to the appropriate publishing guide.

Table 1.1. Choose your starting point

Starting point	First guide
You want the ready-made <code>prodockit-template</code> report project	Build a template site explains what it provides and which files become yours
A new computer or an incomplete template-based checkout	Bootstrap checks and prepares Python, Git, the editor, and the project environment; <code>pdk pdf</code> owns PDF runtimes
An established documentation project that should keep its existing design and workflow	Adoption integrates selected prodockit components without replacing those choices
An existing template-derived project	Staying in step with the template brings shared workflows and publishing files up to date without replacing your writing
A working project that already previews with <code>zensical serve</code>	Continue with the publishing path below
A prodockit package release rather than a documentation project	Use the maintainer Build and release runbook instead

The template is a starting copy, not a live dependency. Your Markdown remains project-owned; later template fixes arrive only when you review and apply a template sync.

1.2 Choose your features

Prodockit separates features that change authored Markdown from tools that build, inspect, or maintain the complete project. Start with the group that matches the outcome you need.

1.2.1 Authoring extensions

These are standard Python-Markdown extensions configured in `zensical.toml`:

[Table 8.2](#) explains the practical benefit each extension brings to an author.

Table 1.2. Authoring extensions

Extension	Benefit to the author
prodockit.headings	Numbers the document hierarchy consistently in the website and PDF. Sections can move without the author manually renumbering every later heading.
prodockit.refs	Links prose to headings, figures and tables by identity rather than a typed number. The displayed number and title follow the target when the document is reorganised, preventing stale "see section..." references.
prodockit.citations	Provides a lightweight citation and reference-list approach written entirely in Markdown. A short document can present credible evidence without requiring a separate bibliography database or processing tool.
prodockit.glossary	Defines specialist terms and acronyms once, then presents them consistently throughout the document. Readers get expansions and a shared glossary while authors avoid repeating and synchronising definitions manually.
prodockit.tables	Adds widths, merged cells, grouped or rotated headings, shading and compact layout. Authors can give information the format it needs instead of struggling within basic Markdown table capabilities, with the result preserved across website and PDF.
prodockit.tree	Turns a simple indented description into a readable directory hierarchy. File relationships remain clear without maintaining fragile hand-drawn ASCII connectors.
prodockit.steps	Gives procedures and methods consistent numbering and visual progression. Steps can be inserted, removed or rearranged without manually repairing the sequence or layout.
prodockit.bibliography	Uses reusable BibTeX or BibLaTeX records and a CSL style to produce citations and a bibliography. It scales to larger evidence bases and formal publication styles while keeping references consistent.

Extension	Benefit to the author
prodockit.index	Generates a back-of-book index for the PDF from terms selected in the source. Readers can find related discussion across chapters without authors building and updating an index by hand.

Every extension is independent. Start with one; add another when the document needs it.

1.2.2 Lifecycle management tooling

Prodockit provides commands that keep an installed project supportable after its first successful build. They separate assessment, software alignment, template updates and verification so an author can make deliberate changes without rebuilding the project by hand.

[prodockit adopt](#)

Brings an established Zensical project onto the software combination supported by the installed Prodockit release, including upgrading or downgrading managed Python packages and Pandoc when required. The benefit is a repeatable route back to a tested toolchain without replacing the project's content, design, Git history or publishing workflow.

[prodockit template-sync](#)

Compares a project with the template release it came from and prepares updates to shared workflows, configuration and other template-managed files. It preserves author-owned content and isolates conflicts for review, so fixes and lifecycle improvements can be adopted without overwriting deliberate project customisation.

[prodockit pins](#)

Updates the version declarations spread across requirements, workflows and tool configuration as one reviewed change. This matters because independently upgraded or downgraded tools may still install successfully while producing different website or PDF output; Pins returns the project to a combination tested together.

[pdk diag](#)

Checks the active interpreter, installed distributions, project configuration, required renderers and version drift without changing the project. It tells the author what is wrong and what evidence supports that conclusion, reducing lifecycle maintenance from trial-and-error reinstalls to a targeted remediation.

[prodockit sync-repo](#)

Keeps repository links, badges, icons and related metadata consistent with the configured Git remote. It prevents a cloned, renamed or transferred project from continuing to publish stale ownership and repository information.

[prodockit update-dates](#)

Derives page modification dates from Git history and records them for publication.

Readers can judge how current the material is without requiring authors to maintain dates manually.

`prodockit.testing`

Checks the built website and PDF, including links, headings and required content. Lifecycle changes are useful only when the delivered artifacts still work, so these tests turn a successful command into evidence that the publication remains usable.

1.2.3 Publishing outputs

`prodockit.pdf` builds a standalone printable document from the same navigation and source as the website, while `prodockit.source-bundle` packages the underlying Markdown and configuration for disclosure or submission. The `prodockit.zensical_macros` integration adds reusable project, repository and document values to website templates without duplicating them throughout the source.

After choosing the features and outputs this project needs, continue with [Follow the publishing path](#). Use the [Authoring reference](#) for syntax and examples. Use the [command-line reference](#) for the exact behaviour of each command.

1.3 Configure Prodockit features

Each Prodockit extension is registered as a standard Python-Markdown extension under the `markdown.extensions` entry point group. Enable an extension by name, just as you would enable a built-in extension such as `toc` or a `pymdownx` extension:

```
import markdown

html = markdown.markdown(
    text,
    extensions=["prodockit.headings", "prodockit.refs", "prodockit.tables"],
)
```

In a [Zensical](#) project, enable extensions in `zensical.toml` alongside the built-in and `pymdownx` extensions. Unlike the `pymdownx` and Zensical namespaces, Zensical does not hoist a nested `prodockit.headings` table into that dotted extension name, so each Prodockit name must be quoted:

```
[project.markdown_extensions."prodockit.headings"]
[project.markdown_extensions."prodockit.refs"]
[project.markdown_extensions."prodockit.citations"]
[project.markdown_extensions."prodockit.glossary"]
[project.markdown_extensions."prodockit.tables"]
[project.markdown_extensions."prodockit.tree"]
[project.markdown_extensions."prodockit.steps"]
[project.markdown_extensions."prodockit.bibliography"]
[project.markdown_extensions."prodockit.index"]
```

Enable only the features the document uses. Each extension is independent and none requires another.

1.3.1 What is *not* an extension

Several parts of Prodockit have no `markdown.extensions` entry point and nothing to add to `zensical.toml`, because they are commands or integrations rather than Markdown syntax.

[Table 8.3](#) distinguishes these from the nine Markdown extensions.

Table 1.3. What is not an extension

prodockit_pdf	A separate PDF-generation build step
prodockit_source-bundle	Packages documentation source as a separate PDF
prodockit.zensical_macros	A <code>define_env()</code> module for Zensical's macros plugin, named under its <code>modules</code> config rather than as an extension
prodockit_testing	pytest fixtures and checks for an already-built site and PDF
prodockit_bootstrap	Sets up a machine and a project based on <code>prodockit-template</code>
prodockit_sync-repo	Keeps repository metadata and README badges matching the Git remote
prodockit_pins	Moves build-input version pins together
prodockit_template-sync	Brings a project back into step with its template

Contributors changing the package itself should use the editable installation and repository checks in [Development and code map](#).

1.4 Follow the publishing path

Build and inspect the complete outputs locally before asking the hosting service to publish the same commit.

1. Prepare the checkout

From the project root, confirm that Git sees only the work you intend to publish:

```
git status --short
```

For a project prepared through bootstrap, run its report-only setup check:

```
prodockit bootstrap
```

For an adopted or manually installed project, activate its environment and verify the dependencies described by its chosen route instead. A first build on a machine does not, by itself, mean that bootstrap should be run.

For a template-derived project, also preview upstream publishing changes:

```
prodockit template-sync
```

Neither ordinary command applies changes. Follow its detailed guide if it reports work to do.

2. Build the website strictly

```
zensical build --clean --strict
```

`--strict` turns validation warnings such as broken internal links into failures. The result is written to `site/` unless the project configures another `site_dir`.

3. Build the PDF from the completed website

```
prodockit pdf
```

The PDF uses the rendered website pages and the order in `zensical.toml`. The command does not invoke Zensical, so build the website first. Skip this step only when the project deliberately publishes no PDF.

See [Generate a PDF](#) for the three preparation routes, required system tools, and optional page, cover, diagram, maths, and index settings.

4. Add optional page dates

If the website should display page update dates, run:

```
prodockit update-dates
```

It adds each page's date to the completed HTML; the source Markdown and configuration remain unchanged. Omit it when dates are not required.

5. Test the files that were built

```
python -m pytest
```

Source tests and output tests answer different questions. Output tests can open the generated HTML and PDF, confirm expected pages and fonts, and detect raw Mermaid or TeX source left behind by a missing renderer. See [Test the built output](#).

6. Push through the publishing workflow

Commit only reviewed source files, then push the branch and use the repository's normal pull-request or merge-request gate:

```
git push -u origin HEAD
```

After the change reaches the default branch, the supplied workflow rebuilds the PDF and site and runs its output checks on a clean Linux runner before it deploys. [Publish automatically](#) explains the GitHub and GitLab recipes, which checks are gates, and every external tool they install.

7. Verify the public result

Do not stop at a green deployment job. Open the public website, follow its PDF download, and check a page changed by this publication.

GitHub Pages

Open the repository's **Actions** page, select the documentation workflow, and confirm both its deploy and live-verification jobs passed.

GitLab Pages

Open **Build > Pipelines**, inspect the `pages` job, then use **Deploy > Pages** to open the published address.

The workflow proves that the artifact was accepted; the final browser check proves that a reader can retrieve the intended version.

1.5 Know which output you are checking

Use [Table 8.4](#) to distinguish local intermediates from the website and PDF readers finally receive.

Table 1.4. Know which output you are checking

Output	Built by	Typical location	Final check
Local preview	<code>zensical serve</code>	Address printed in the terminal	Edit a page and see it refresh
Static website	<code>zensical build --clean --strict;</code> optionally <code>prodockit update-dates</code>	<code>site/</code>	Open pages, optional revision dates, navigation, links, search, and downloadable files
Complete PDF	<code>prodockit pdf</code>	<code>docs/</code> <code>site_documentation.pdf</code> by default	Inspect cover, contents, page breaks, diagrams, fonts, and index
Hosted website	GitHub Pages or GitLab Pages workflow	Project Pages URL	Confirm the public page contains the reviewed change

1.6 Build with revision dates

The optional `prodockit update-dates` command gives the final site an “Updated” fact without putting generated fields into tracked Markdown. Run it only when the published website should display page dates. Without it, the Zensical build and publication workflow remain complete and valid.

See [Page update dates](#) for where the date is inserted and how an author can override it for one page. The rest of this section covers building and publication.

1.6.1 Use the command without adoption

`prodockit update-dates` is a standalone command. It needs a website already built from an existing Zensical project, but it does **not** require that project to adopt Prodockit's extensions, stylesheets, macros, template, or publishing workflows.

1. Change to the directory containing `zensical.toml`:

```
cd /path/to/your-document
```

2. Activate the Python environment that normally builds the document:

macOS

```
source .venv/bin/activate
```

Windows PowerShell

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned  
.\.venv\Scripts\Activate.ps1
```

Ubuntu

```
source .venv/bin/activate
```

3. Install the latest Prodockit package into that environment:

macOS

```
pip3 install --upgrade prodockit
```

Windows

```
pip install --upgrade prodockit
```

Linux (Ubuntu)

```
pip install --upgrade prodockit
```

 If pip or pip3 does not work

If `pip` does not work, try `pip3`; if `pip3` does not work, try `pip`. Keep the intended virtual environment active and check that the alternative command belongs to it before installing packages.

4. Build the site with its usual command, then add the dates:

```
zensical build --clean --strict  
prodockit update-dates
```

The second command changes only the configured website output, normally `site/`. It does not call Zensical and does not edit the source Markdown or configuration. You do not need to run `prodockit adopt` before or after these steps.

1.6.2 Use the command without Git

Git is optional. When the project is not inside a Git repository, `prodockit update-dates` uses each Markdown file's modification timestamp as its update date. It converts the timestamp to a calendar date in UTC and reports that fallback while building. Saving a file changes its modification time, so the next build updates that page's date.

Run the same two-command sequence:

```
zensical build --clean --strict
prodockit update-dates
```

You do not need an option to enable this fallback. A manually supplied `revision_date`, as shown above, takes priority in both Git and non-Git projects.

1.6.3 Understand dates in a Git project

For a tracked page, the newest Git **author date** is used. A new or untracked page that has no Git history yet uses the source file's modification timestamp. Prodockit converts either automatic timestamp to UTC before taking its calendar date, so authors and CI runners in different time zones get the same result. The command names a modification-time fallback in its output. A manually written `revision_date` or `git_revision_date_localized` in page front matter always wins.

If the project is in Git but you deliberately want filesystem dates rather than Git author dates, select them explicitly:

```
prodockit update-dates --modification-dates
```

This applies modification dates to tracked and untracked Markdown files alike. Prodockit does not calculate or inject creation dates.

The command refuses a shallow repository because Git can return its oldest available boundary commit as a believable but incorrect page date. In GitHub Actions use `fetch-depth: 0`; in GitLab CI use `GIT_DEPTH: "0"`. A repository that exists but cannot be read also fails instead of silently substituting a different date.

Prodockit inserts the facts into the generated HTML only. It deliberately does not invoke a site builder, so this post-processing step composes with other tools that run before or after Zensical. Run it again after every clean site build because that build replaces the generated HTML.

A normal `zensical serve preview` rebuilds pages continuously and therefore does not retain automatically generated dates. Pages that declare a date in front matter can still show it in the preview; use the completed static build to inspect automatic Git or modification dates.

1.7 Use the detailed guides when needed

Open the detailed guide for the part of the workflow that needs attention:

- [Set up a machine](#) prepares a computer and checkout.
- [Build a template site](#) introduces the starter project, its two outputs, and the boundary between your work and shared publishing infrastructure.
- [Staying in step with the template](#) updates shared publishing infrastructure without taking ownership of project content.
- [Generate a PDF](#) covers local PDF requirements and configuration.
- [Publish automatically](#) provides GitHub Actions and GitLab CI workflows.
- [Test the built output](#) adds reusable pytest checks.
- [Website macros](#) is the advanced reference for values and layout helpers evaluated while the site and PDF are rendered.