prodockit

A docs-as-code workflow for professional and academic documentation, built on Zensical and Markdown.

Release: prodockit-v0.41.0

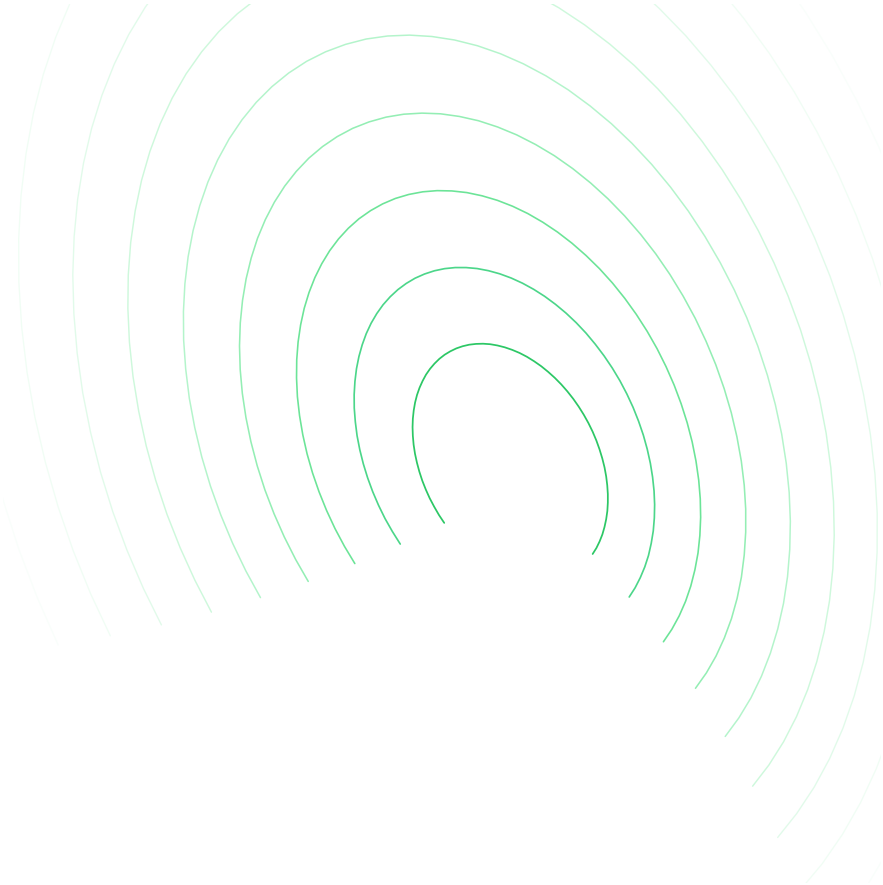


Table of Contents

- 37.16 0.32.0 (2026-08-16)
- 37.17 0.31.1 (2026-08-16)
- 37.18 0.31.0 (2026-08-13)
- 37.19 0.30.1 (2026-08-12)
- 37.20 0.30.0 (2026-08-12)
- 37.21 0.29.0 (2026-08-12)
- 37.22 0.28.1 (2026-08-12)
- 37.23 0.28.0 (2026-08-12)
- 37.24 0.27.0 (2026-08-12)
- 37.25 0.26.7 (2026-08-12)
- 37.26 0.26.6 (2026-08-12)
- 37.27 0.26.5 (2026-08-12)
- 37.28 0.26.4 (2026-08-11)
- 37.29 0.26.3 (2026-08-11)
- 37.30 0.26.2 (2026-08-11)
- 37.31 0.26.1 (2026-08-11)
- 37.32 0.26.0 (2026-08-11)
- 37.33 0.25.0 (2026-08-11)
- 37.34 0.24.1 (2026-08-10)
- 37.35 0.24.0 (2026-08-10)
- 37.36 0.23.0 (2026-08-10)
- 37.37 0.22.0 (2026-08-10)
- 37.38 0.21.0 (2026-08-10)
- 37.39 0.20.1 (2026-08-09)
- 37.40 0.20.0 (2026-08-09)
- 37.41 0.19.1 (2026-08-07)
- 37.42 0.19.0 (2026-08-07)
- 37.43 0.18.1 (2026-08-04)
- 37.44 0.18.0 (2026-08-02)
- 37.45 0.17.5 (2026-08-02)
- 37.46 0.17.4 (2026-08-02)
- 37.47 0.17.3 (2026-08-02)
- 37.48 0.17.2 (2026-07-30)
- 37.49 0.17.1 (2026-07-29)
- 37.50 0.17.0 (2026-07-28)
- 37.51 0.16.0 (2026-07-28)
- 37.52 0.15.2 (2026-07-28)
- 37.53 0.15.1 (2026-07-27)
- 37.54 0.15.0 (2026-07-27)
- 37.55 0.14.0 (2026-07-27)
- 37.56 0.13.0 (2026-07-27)
- 37.57 0.12.0 (2026-07-27)
- 37.58 0.11.1 (2026-07-27)
- 37.59 0.11.0 (2026-07-27)
- 37.60 0.10.9 (2026-07-26)
- 37.61 0.10.8 (2026-07-26)
- 37.62 0.10.7 (2026-07-25)
- 37.63 0.10.6 (2026-07-25)
- 37.64 0.10.5 (2026-07-25)

1. Overview

This section is for anyone new to prodockit. It explains what the package adds to Zensical, how to install it, and how to build a first local site before you choose authoring or publishing features.

prodockit adds the document features that professional and academic Zensical projects (Zensical website) commonly need: numbered sections, cross-references, citations, glossaries, richer tables, and a printable PDF. Install one Python package, then enable only the parts your project uses.

If you want to see it working before reading the reference pages, follow Build your first site. It starts with a new Zensical project and ends at a live local preview.

1.1 Choose what you need

1.1.1 Authoring extensions

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

Extension	Use it for
`prodockit.headings`	Numbered headings
`prodockit.refs`	Cross-references to headings, figures, and tables
`prodockit.citations`	A small, Markdown-defined reference list
`prodockit.glossary`	Acronyms and glossary terms
`prodockit.tables`	Widths, merged cells, dense tables, and richer headers
`prodockit.tree`	Readable directory trees
`prodockit.steps`	Procedures presented as numbered steps
`prodockit.bibliography`	BibTeX/BibLaTeX citations formatted with CSL
`prodockit.index`	A PDF-only back-of-book index

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

1.1.2 Publishing and project tools

prodockit also provides commands and integrations rather than Markdown syntax:

Feature	Use it for
`prodockit pdf`	Build a standalone PDF from the same navigation as the site
`prodockit source-bundle`	Package the underlying Markdown and configuration as a PDF
`prodockit.zensical_macros`	Word counts, repository data, and document-wide numbering in templates
`prodockit.testing`	Check the built website and PDF with pytest
Maintain prodockit	Maintain the package repository, build pins, automation, and releases

Go to the Authoring reference when you want to add document features. Go to Publish a document when you are ready to use the template, build a PDF, or publish with continuous integration. The command-line reference says which commands change files and which only report.

1.2 Project status

prodockit is early but functional. All nine extensions, the PDF and source bundle builders, website macros, testing support, and project-management commands are implemented and tested. See Support and compatibility for maturity, PyMdown Blocks and other required versions, platform coverage, and known constraints. Read the release notes before upgrading.

2. Installation

2.1 Requirements

Python 3.10 or later. Tested on 3.10, 3.11, 3.12 and 3.13; `pip` will refuse to install on anything older rather than failing later at import.

Everything below is pulled in automatically by `pip install prodockit`, except where noted:

Requirement	Needed for
`Markdown` (>= 3.10.3)	every extension
`zensical` (>= 0.0.55)	Zensical integration and `prodockit.zensical_macros`
PyMdown Extensions (>= 11.0.1)	`prodockit.steps` and `prodockit.tree` are built directly on the PyMdown Blocks API; `prodockit.pdf` also preserves the output of PyMdown features
`beautifulsoup4` (>= 4.12)	`prodockit.pdf`
`click` (>= 8.0)	the `prodockit` command-line tool
`pypdf` (>= 4.0)	`prodockit.pdf`
`tomli` (>= 2.0)	reading a template manifest on Python 3.10, where `tomllib` does not exist yet
`pymupdf` (>= 1.24)	only the back-of-book index - `pip install prodockit[index]`

The floors above are the ones declared in `pyproject.toml`, and a test keeps this table in step with them - the two had drifted apart, with `Markdown` recorded here as >= 3.4 long after the real floor moved to 3.10.3 (prodockit-extensions#372).

2.1.1 Not installed by pip

These are the ones `pip install prodockit` does **not** bring, and they differ in kind:

Requirement	Needed for
`weasyprint` (>= 69)	`prodockit.pdf`. A Python package, but not a dependency of prodockit - install it yourself. `prodockit.pdf` runs its command-line rather than importing it
`pandoc` (>= 3, builds pin 3.10.1)	`prodockit.pdf`, and `prodockit.bibliography` even without a PDF build. Genuinely not a Python package - there is nothing for `pip` to install
`mermaid-cli`, `mathjax-full` (Node >= 22)	only Mermaid diagrams and TeX maths in the PDF
Chrome or Chromium	only Mermaid diagrams - `mermaid-cli` renders them through a headless browser
A citation style (`.csl`)	only `prodockit.bibliography`. Fetched per build, not vendored - see below

The citation style is a download rather than an install. Pandoc resolves `harvard-cite-them-right.csl` from the directory it runs in, and every CI script here fetches it immediately before building:

```
curl -fsSL -o harvard-cite-them-right.csl "https://www.zotero.org/styles/harvard-cite-them-right"
```

It is deliberately **not** committed: it is third-party content with its own licence and its own release cadence, and a vendored copy would go stale silently while every build kept succeeding. `prodockit bootstrap` fetches it for you, and `.gitignore` keeps a local copy out of commits.

`weasyprint` is worth separating from `pandoc` rather than filing both as "external binaries": one is a `pip install` away and the other is not, and a reader who treats them alike goes looking for a package that does not exist, or misses one that does.

Pandoc is version-sensitive in a way that changes output rather than breaking the build: a major version below 3 renders code blocks as justified prose, and the builds pin an exact release because two 3.x versions have already disagreed about the same source. `prodockit bootstrap` installs the pinned version where a package manager allows it and tells you when your local pandoc differs - see Pinning build inputs.

See PDF generation for how `prodockit.pdf` locates these, and Known limitations for why the Node ones are needed at all. A build with neither Mermaid diagrams nor maths needs neither of them, and no browser.

2.2 From PyPI

```
pip install prodockit
```

For a minimal project that needs no PDF toolchain, continue with Build your first site. The external tools above can be added later when the document needs their features.

2.3 Enabling an extension

Each prodockit extension is registered as a standard Python-Markdown extension under the `markdown.extensions` entry point group, so it can be enabled by name, the same way you'd enable a built-in extension like `toc` or a `pymdownx` one:

```
import markdown

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

Or, for a Zensical project, in `zensical.toml` alongside the built-in and `pymdownx` extensions. Unlike `pymdownx`'s and Zensical's own namespaces, Zensical doesn't hoist a nested `prodockit.headings` table into that dotted extension name, so each one needs a quoted key instead:

```
[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 ones you use - each is independent, and none of them requires another.

2.3.1 The nine extensions

See each extension's own page for its syntax, examples, and configuration:

Extension	What it adds
`prodockit.headings`	Numbered headings, and a number a cross-reference can point at
`prodockit.refs`	Cross-references that resolve to a number *and* a name
`prodockit.citations`	Citation handling
`prodockit.glossary`	Acronyms and a glossary
`prodockit.tables`	Column widths, dense tables, multi-row headers, merged cells, rotated headings
`prodockit.tree`	A directory listing that looks like one
`prodockit.steps`	Numbered steps a reader works through in order
`prodockit.bibliography`	A bibliography built from your `.bib` files
`prodockit.index`	A back-of-book index (PDF only)

2.3.2 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 not Markdown syntax:

`prodockit pdf`	A separate PDF-generation build step
`prodockit source-bundle`	Packages documentation source as a separate PDF
`prodockit.zensical_macros`	A `define_env()` module for Zensical's macros plugin, named under its `modules` 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 to build the docs
`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 the template it came from
`prodockit init-tools` / `init-mathjax`	Sets up optional Mermaid and maths rendering tools

Contributors changing the package itself should use the editable installation and repository checks in Development and code map.

3. Build your first site

This walkthrough starts with an empty directory and ends with a local Zensical site using numbered headings and a cross-reference. It also demonstrates `prodockit.steps`: the procedure you are reading is rendered by that extension.

1. Install Python

   prodockit requires Python 3.10 or later. Install Python for your operating system, then close and reopen the terminal so the new command is on `PATH`.

   macOS

   Install Homebrew first if you do not already have it, then run:

   ```
   brew update
   brew install python
   python3 --version
   ```

   Windows

   Open PowerShell and run:

   ```
   winget install --exact --id Python.Python.3.13
   python --version
   ```

   If `python` opens the Microsoft Store, search Windows for **Manage app execution aliases** and turn off the App Installer aliases for `python.exe` and `python3.exe`.

Linux (Ubuntu)

Open a terminal and run:

```
sudo apt update
sudo apt install python3 python3-venv python3-pip
python3 --version
```

2. Create and activate a virtual environment

   Create a directory for the site, then create the virtual environment inside it:

   macOS

   ```
   mkdir my-docs
   cd my-docs
   "$(brew --prefix)/bin/python3" -m venv .venv
   source .venv/bin/activate
   ```

   Windows

   In PowerShell:

   ```
   mkdir my-docs
   cd my-docs
   python -m venv .venv
   .\.venv\Scripts\Activate.ps1
   ```

   Linux (Ubuntu)

   ```
   mkdir my-docs
   cd my-docs
   python3 -m venv .venv
   source .venv/bin/activate
   ```

   The prompt normally starts with `(.venv)` after activation. The rest of the

walkthrough uses `python`, which now means the interpreter inside that virtual environment on all three platforms.

3. Install prodockit

```
python -m pip install prodockit
```

Zensical is a core dependency, so this installs the `zensical` command too. Confirm both commands are available:

```
prodockit --version
zensical --version
```

4. Create the Zensical project

```
zensical new .
```

This creates `zensical.toml` and a starter `docs/` directory without overwriting unrelated files.

5. Enable the two extensions

Add these tables at the end of `zensical.toml`:

```
[project.markdown_extensions."prodockit.headings"]
[project.markdown_extensions."prodockit.refs"]
```

The quoted table names matter: each dotted extension name must remain one TOML key. Extensions are independent, so a project can enable only these two.

6. Add content that uses them

Replace `docs/index.md` with:

```
# My first document

The detail is in \ref{results}.

## Method

Describe what you did here.

## Results {: #results }

Describe what you found here.
```

`prodockit.headings` numbers the sections. `prodockit.refs` turns `\ref{results}` into a link containing the current number and title, so it stays correct if the sections move.

7. Preview the site

```
zensical serve
```

Open the local address printed in the terminal. Zensical rebuilds the preview when a source file changes; stop it with `Ctrl+C`.

3.1 Where to go next

- Browse the authoring reference when you need another document feature.
- Read Generate a PDF when the website is ready to print or submit.
- Follow the project maintenance cycle when the first site becomes a maintained project, then use the command-line map to choose a command safely.

> **Previewing these documentation changes**
>
> From this repository's root, run `zensical serve` and open the address it prints. This page already has `prodockit.steps` enabled and styled.

4. Authoring reference

This section is for a document author who has built a first Zensical site and wants to add structure or specialist content to its Markdown pages. You do not need to read every page: choose the feature the document needs, enable that extension, and copy its smallest complete example.

4.1 Build on PyMdown Blocks

Two prodockit extensions are built directly on PyMdown Blocks (see the upstream Blocks guide): `prodockit.steps` for procedures and `prodockit.tree` for directory listings. They use PyMdown's slash-fenced container syntax, nesting rules, and block options rather than introducing a separate container language.

If you already use PyMdown Blocks, the structure will be familiar. If you do not, start with Numbered steps: its first example shows the complete opening and closing fences before explaining nested blocks.

4.2 Follow the same three stages

Every extension guide starts with the same learning path:

1. Enable the extension

   Add the extension's table to `zensical.toml`. Each prodockit extension is independent, so enabling headings does not silently enable citations, tables, or another feature.

2. Write the Markdown

   Start with the complete copyable example. The guide shows both its Markdown source and rendered result before introducing variations.

3. Configure only when needed

   Keep the defaults for a first use. The configuration section explains the available `zensical.toml` settings and shows a rendered example when an option changes visible output.

4.3 Choose a feature

Document need	Reference
Number sections and give headings stable links	Headings

Document need	Reference
Refer to a heading, figure, or table without typing its changing number	Cross-references
Define short references directly in Markdown	Hand-written citations and references
Expand abbreviations and define specialist terms	Acronyms and glossary
Control column widths, merged cells, and dense layouts	Tables
Show a folder and file hierarchy	Directory trees
Present a procedure with connected numbered stages	Numbered steps
Cite a BibTeX library in a selected CSL style	Bibliography
Mark terms for a PDF back-of-book index	Index

`prodockit.citations` and `prodockit.bibliography` are two approaches to the same broad task. A small document can define sources directly in Markdown; a report with an existing `.bib` library normally uses the bibliography extension. You do not need to enable both.

4.4 Keep publishing concerns separate

The authoring pages explain what to write. When the content is ready to become a complete PDF or hosted website, continue to Publish a document. Machine setup, template updates, CI, Pages deployment, and output tests live there so they do not interrupt the Markdown reference.

5. Headings

`prodockit.headings` numbers the headings in your document as sections, such as `1`, `1.1`, and `1.2`. The numbers update automatically when you add, remove, or move a heading.

Use it when your document needs numbered sections. Add Cross-references when you also want to link readers to those sections by number and name.

5.1 Enable the extension

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.headings"]
numbering = "continuous"
```

`continuous` carries the numbering across the pages in your Zensical navigation. Use `per-document` instead when every page should start again at section 1.

5.2 Number headings

Each `#` heading starts a main section (`1`, `2`, and so on). A `##` heading starts a section inside it (`1.1`, `1.2`, and so on):

Markdown

```
# Introduction

## Background

## Scope

# Method
```

Result

Heading	id	number
Introduction	`introduction`	`1`
Background	`background`	`1.1`
Scope	`scope`	`1.2`
Method	`method`	`2`

The extension calculates the section numbers, but it does not add them to the visible heading text on the website. Cross-references uses the calculated numbers in links such as “1.1 Background”.

> **Why the heading itself does not visibly change**
>
> The section numbers appear in Cross-references, rather than next to the website's heading text.

5.3 Configure headings

5.3.1 Choose how numbering continues

The `numbering` setting accepts two values:

Value	Result
`"continuous"`	Continue the main section numbers across pages in Zensical navigation order.
`"per-document"`	Start each page's main section numbering at 1. This is the extension's default.

For a multi-page documentation site, set the option in `zensical.toml`:

```
[project.markdown_extensions."prodockit.headings"]
numbering = "continuous"
```

For example, if one page ends at section 3, the next page starts at section 4. This also lets Cross-references show one consistent set of section numbers across the site.

5.3.2 Number appendices

Add `is_appendix: true` to the settings at the top of the page (its front matter) to use letter-based numbering instead of the normal numeric sequence - `"A"`, `"A.1"`, `"A.1.1"` - once you've enabled continuous numbering (see Reference below). An appendix page doesn't consume a number from the numeric sequence at all, so pages after it aren't left with a gap. Letters are assigned sequentially in nav order - the first `is_appendix` page becomes `"A"`, the second `"B"`, and so on, independent of how many numbered pages come before them.

For example:

Markdown

```
---
is_appendix: true
---

# Glossary

## Terms {: #terms }
```

Result

Heading	Number
Glossary	A
Terms	A.1

a prodockit.refs reference to `Terms` from another page:

```
<!-- references.md -->
# References

See \ref{terms} for defined terms.
```

renders to (a link to `glossary.md#terms`, shown here as a code block since `glossary.md` isn't a real page on *this* site):

```
<p>See <a class="prodockit-ref" href="glossary.md#terms">A.1</a> for
defined terms.</p>
```

`Glossary`'s own `h1` becomes `"A"` (the first appendix page in nav) and its `Terms` subheading becomes `"A.1"` - and `References`, the page after it, still gets the next plain number in the numeric sequence (`"2"`, not `"3"`), exactly as if the appendix page had never consumed one. Only meaningful under Zensical; ignored otherwise.

5.3.3 Leave a heading unnumbered

A heading with an `unnumbered` class - e.g. a cover page or title slide - still gets an id, but is skipped when computing section numbers, so it doesn't consume a counter position:

Markdown

```
# Cover Page {: .unnumbered }

# Introduction
```

Result

Heading	Number
Cover Page	none
Introduction	1

`Introduction` above is still numbered `1`, as if `Cover Page` weren't there at all.

5.3.4 Hide a heading from PDF navigation

A PDF built by prodockit.pdf has *two* tables of contents, and `unnumbered` alone only reaches one of them:

- The generated **Table of Contents page**, built from every heading Pandoc sees. Add `unlisted` to also keep a heading off this page - it still keeps its `id` and number (if any), the same as `unnumbered` above. Pandoc itself defines this class and honours it via

`pandoc.structure.table_of_contents()`; prodockit doesn't add or change its meaning.
- The **bookmark outline** - the navigation pane a PDF reader shows down the side. This is built separately by WeasyPrint from every `h1`-`h6` in the document, and `unlisted` has no effect on it at all. Add `unbookmarked` to also remove a heading from the outline.

```
# Cover Page {: .unnumbered .unlisted .unbookmarked }
```

This distinction matters because outline nesting follows heading level: an `unlisted` (but not `unbookmarked`) `h1` still becomes a *top-level* outline node, and every following heading of lower level nests underneath it instead of under its real chapter - not just one stray entry, but a misnested chunk of the outline. See Table of contents and bookmark outline for the underlying stylesheet rule and worked example.

5.4 Reference

5.4.1 Ids

An id comes from one of, in order of precedence:

1. An explicit id set via `attr_list`, e.g. `# Introduction {: #custom-id }`.
2. Python-Markdown's own `toc` extension, which `prodockit.headings` enables automatically (with its defaults) if you haven't already enabled it yourself - so if you *have* configured `toc` (e.g. with `permalink: true`), that configuration is left untouched and reused.
3. A minimal built-in slugify fallback, used only if `toc` is somehow not registered at all (this should not normally happen, since `prodockit.headings` enables it).

5.4.2 Zensical settings

Setting	Default	What it controls
`numbering`	`"per-document"`	Use `"continuous"` to carry main section numbers across pages in Zensical navigation order.
`appendix_attr`	`"is_appendix"`	Name of the front matter setting that marks an appendix page. Change this only if your project uses another name.
`source`	`""` (detected automatically)	Advanced: identifies the current page when using the extension outside Zensical. Leave it unset in `zensical.toml`.

5.4.3 Cross-page numbering in Zensical

Zensical shares heading information across the complete navigation automatically. With `numbering = "continuous"`, adding or reordering a page updates later section numbers without a manual starting value.

Give repeated headings explicit ids. If several pages each contain `## Quick start`, their automatically generated ids collide and the build warns; which page renders first is not a stable way to choose a link target:

```
## Quick start {: #refs-quick-start }
```

For integration with another Markdown renderer, see Extension integration.

5.5 Customise with a CSS style sheet

`prodockit.headings` doesn't add any class of its own to a heading - only an `id` (see above), the class(es) already on the heading (e.g. `unnumbered`), and whatever numbers are consumed by prodockit.refs. There is no `prodockit-heading`-style class to hook a stylesheet onto directly.

6. Cross-references

`prodockit.refs` creates links to headings elsewhere in your documentation. Each link shows the heading's current number and name, such as “1.1 Configuration”, so you do not have to update it when sections move.

Use `\ref` for a link that works on the website and in the PDF. Use `\autoref` when the PDF should also show the target's page number.

6.1 Enable the extension

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.refs"]
```

6.2 Reference a heading

Reference any heading's id with `\ref{id}`:

Markdown

```
# Introduction {: #intro }

See \ref{intro} for background.

## Background
```

Result

Introduction

See 1 Introduction for background.

Background

The link takes the reader to the `Introduction` heading. Its number and name update automatically if the document changes.

6.3 Configure cross-references

6.3.1 Choose the missing-reference text

An unresolved reference displays `??` by default. Set `unresolved` if your project uses a different marker:

```
[project.markdown_extensions."prodockit.refs"]
unresolved = "MISSING"
```

`source` is the only other TOML setting. It identifies the current page, but Zensical detects it automatically; leave it unset in `zensical.toml`.

6.3.2 Reference a heading before it appears

A reference to a heading defined *later* in the same document resolves correctly:

```
See \ref{background} below.

## Background {: #background }
```

6.3.3 Fix a missing reference

If the heading id is missing or mistyped, `\ref{id}` displays `??` instead of a link. Check that the text inside the braces exactly matches the id on the heading.

A heading marked `unnumbered` still works as a link. Because it has no section number, the link displays only the heading name.

```
# Cover Page {: .unnumbered #cover-page }

See \ref{cover-page}.
```

renders `\ref{cover-page}` as `Cover Page`, linked to `#cover-page`.

6.3.4 Include a page number in the PDF

`\ref{id}` and `\autoref{id}` render exactly the same text - the target's number and name. The difference is that `\autoref{id}` also carries the target's **page number** in the PDF, which is what a reader holding a printout needs and what a website reader has no use for:

Markdown

```
Configuration is covered in \autoref{configuration}.
```

Website

Configuration is covered in 1.1 Configuration.

PDF

Configuration is covered in 1.1 Configuration on page 12.

The " on page N" suffix comes from prodockit.pdf's own stylesheet, so it appears only in the PDF - a page number on a scrolling website would be meaningless. Nothing to enable: build the PDF and it is there.

Which to use is a per-reference decision rather than a project-wide setting: use `\autoref{id}` where a printed reader needs to turn to something, and `\ref{id}` where the extra "on page N" would just be noise.

An appendix needs nothing special - its letter is already the first segment of its number, so `\ref{terms}` renders "A.1 Terms".

6.3.5 Reference a figure or table

`\ref{id}` also resolves a captioned figure or table, rendering its label:

```
![Component Model](assets/images/component-model.png)
{ width="100%" }
/// figure-caption
    attrs: {id: fig-component-model}

Component Model
///

The components inside the System Context boundary are shown in
\ref{fig-component-model}.
```

which renders as a link reading **Figure 3.1**.

Figures and tables are counted separately, and both restart per page and carry the page's chapter number - the same numbering the caption itself shows, so a reference and the thing it points at always agree.

> ⚠ **The id goes in an `attrs:` option, not `{: #id }`**
>
> Caption blocks take attributes the Blocks API way - an indented `attrs:` line, then a blank line, then the caption text. The `{: #id }` form used on headings, images and table cells **does not work here**: it produces no figure at all, silently, and the `///` lines appear as literal text.
>
> An id containing a colon (`fig:component-model`) also produces no figure, quoted or not. Use a hyphen.

Unlike a heading, a caption reference is **its label alone** - "Figure 3.1", not "Figure 3.1 Component Model". A caption is referred to mid-sentence, where repeating its own words reads as a stutter; a heading's number alone would say nothing about where the reader is being sent, so that keeps its name.

6.4 Reference

6.4.1 Syntax

Syntax	Result
`\ref{<id>}`	The target's current number and name
`\autoref{<id>}`	The same link, plus “on page N” in the PDF

`<id>` is the target heading's id - either one you set explicitly via `attr_list` (`# Introduction {: #intro }`), or the one `toc` derived automatically from the heading text (see prodockit.headings for the exact precedence).

`\ref{...}` is recognised the same way Python-Markdown's own inline syntax is - meaning it's protected inside inline code spans and fenced code blocks, so it's safe to show as a literal example:

````
Type `\ref{intro}` to reference a section.

```
\ref{intro}
```
````

Neither of the two shown above is resolved; both render the literal text.

6.4.2 Zensical settings

Setting	Default	What it controls
`unresolved`	`"??"`	Text shown when an id cannot be found.
`source`	`""` (detected automatically)	Advanced: identifies the current page when using the extension outside Zensical. Leave it unset in `zensical.toml`.

6.4.3 Cross-page references

Under Zensical, cross-page references work automatically when headings and references are enabled. Prodockit reads heading ids and numbers across the navigation before individual pages finish rendering, so a reference can point to a page built later.

Two pages with the same generated heading id produce a warning. Give the headings distinct explicit ids so every destination is stable.

For integration with another Markdown renderer, see Extension integration.

6.5 Customise with a CSS style sheet

`prodockit.refs` always sets a class on the `\ref{id}` link it renders - resolved or not - so a stylesheet has a stable hook either way:

Syntax	State	Class
`\ref{id}`	Resolved	`prodockit-ref`
`\ref{id}`	Unresolved	`prodockit-ref prodockit-ref-unresolved`
`\autoref{id}`	Resolved	`prodockit-autoref`
`\autoref{id}`	Unresolved	`prodockit-autoref prodockit-autoref-unresolved`

An unresolved reference (see Unresolved references above) still gets a `class` either way; style `prodockit-ref-unresolved` distinctly (e.g. a warning colour) to make a broken cross-reference visually obvious. Unresolved references have no destination, so an unresolved `\autoref` does not print a stray page-number suffix in the PDF.

7. Hand-written citations and references

`prodockit.citations` lets you write a reference once and cite it from any page. A citation such as `[Skoulikari, 2023]` links readers to the full reference.

> **Looking for an auto-generated bibliography instead?**
>
> This page is for a short reference list that you write yourself. To create a reference list automatically from a `.bib` file, use Bibliography.

Use this extension when you want to write and format a short reference list by hand, but define each citation's link text only once.

7.1 Enable the extension

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.citations"]
```

7.2 Add and cite a reference

Write the full reference, then add its id and the shorter text you want to show in citations on the line below. Cite it with `\citeref{id}`:

Markdown

```
Git is a tool used to manage version control.\citeref{skou-
example}

Skoulikari, A. (2023) *Learning Git: A Hands-On and Visual Guide
to the
Basics of Git*. Sebastopol, CA: O'Reilly Media.
{: #skou-example .reference data-cite-text="Skoulikari, 2023" }
```

Result

Git is a tool used to manage version control.[Skoulikari, 2023]

Skoulikari, A. (2023) *Learning Git: A Hands-On and Visual Guide to the Basics of Git*. Sebastopol, CA: O'Reilly Media.

The extension replaces `\citeref{skou-example}` with the linked citation `[Skoulikari, 2023]`. Select it to jump to the full reference.

7.3 Configure citations

7.3.1 Choose the missing-citation text

An unresolved citation displays `?` by default. Set `unresolved` if your project uses a different marker:

```
[project.markdown_extensions."prodockit.citations"]
unresolved = "MISSING"
```

`source` is the only other TOML setting. It identifies the current page, but Zensical detects it automatically; leave it unset in `zensical.toml`.

Multiple comma-separated keys join into one bracket: `\citeref{skou2023,chacon2014}` → `[Skoulikari, 2023; Chacon and Straub, 2014]`.

7.3.2 Cite a source before its full reference

A citation to a source defined *later* in the same document resolves correctly:

Markdown

```
See \citeref{forward-citation-example} for an introduction to
Git.

Skoulikari, A. (2023) *Learning Git*.
{: #forward-citation-example data-cite-text="Skoulikari, 2023" }
```

Result

See [Skoulikari, 2023] for an introduction to Git.

Skoulikari, A. (2023) *Learning Git*.

7.3.3 Fix a missing citation

If a citation id is missing or mistyped, the extension displays `?` for that entry. Other valid entries in the same citation still work:

Markdown

```
\citeref{forward-citation-example,does-not-exist}
```

Result

[Skoulikari, 2023; ?]

Check the id that produced `?` against the id on the full reference.

7.4 Reference

7.4.1 Syntax

Defining and citing are bundled into one extension, unlike prodockit.headings/ prodockit.refs: a definition is useless without somewhere to cite it, so there's no independently useful "just defining" half to split out.

Defining a source

Any block element - typically a paragraph - with both an `id` and a `data-cite-text` attribute becomes a citable source:

```
Chacon, S. and Straub, B. (2014) *Pro Git*. 2nd edn. New York:
Apress.
{: #chacon2014 .reference data-cite-text="Chacon and Straub, 2014" }
```

`data-cite-text` is the short text rendered at each citation site - it's stripped from

the rendered output (it's internal bookkeeping, not meant to be visible), while `id` stays, since citations link straight to it.

Citing a source

Syntax	Purpose
`\citeref{<id>}`	Cite one defined source
`\citeref{<id1>,<id2>,...}`	Cite several sources in one bracket

Like prodockit.refs, `\citeref{...}` is recognised the same way Python-Markdown's own inline syntax is, so it's protected inside inline code spans and fenced code blocks:

````
Type `\citeref{skou2023}` to cite a source.

```
\citeref{skou2023}
```
````

Neither of the two shown above is resolved; both render the literal text.

7.4.2 Zensical settings

Setting	Default	What it controls
`unresolved`	`"?"`	Text shown for a citation id that cannot be found.
`source`	`""` (detected automatically)	Advanced: identifies the current page when using the extension outside Zensical. Leave it unset in `zensical.toml`.

7.4.3 Cross-page citations

Under Zensical, a citation can refer to a source defined on another page, including a references page later in navigation. Prodockit reads definitions across the navigation before page conversion, so no author configuration is required.

Two definitions using the same key produce a warning and the first definition is retained. Use a unique key for every source.

For integration with another Markdown renderer, see Extension integration.

7.4.4 What this doesn't do (yet)

`prodockit.citations` covers citation-key management - the "define once, cite

anywhere" part. It doesn't auto-generate the references page's listing itself from structured bibliographic data (author/year/title/publisher/URL fields) the way a full BibTeX-style tool would - the reference entry's own text (as shown in the examples above) is still hand-authored prose, just like today. See prodockit.bibliography for the alternative that does exactly this, from a `.bib` file, in any citation style - and for the tradeoffs between the two approaches.

7.5 Customise with a CSS style sheet

`prodockit.citations` emits three hooks - one on the outer wrapper, one on each individual key's own link:

Element	State	Class
Outer `<span>` wrapping the whole `\citeref{...}` citation	always	`prodockit-cite`
Each key's own `<a>`	resolved	`prodockit-cite-resolved`
Each key's own `<a>`	unresolved	`prodockit-cite-unresolved`

An unresolved key's `<a>` has no `href` (see Unresolved citations above) - style `prodockit-cite-unresolved` distinctly (e.g. a muted colour, no underline) to make a missing citation visually obvious.

8. Acronyms and glossary

`prodockit.glossary` lets you define an acronym or term once and reuse it throughout your documentation. Each use links readers to the definition.

Use it for terms that readers may want to look up, such as an acronym or a specialist word.

8.1 Enable the extension

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.glossary"]
```

8.2 Define and use a term

Write the definition, then add its id and the text that should appear in your sentences on the line below. Insert the linked term with `\gls{id}`:

Markdown

```
This site uses a \gls{css-example} to control appearance.

**CSS style sheet** - A file containing Cascading Style Sheets
rules.
{: #css-example .glossary data-term="CSS style sheet" }
```

Result

This site uses a CSS style sheet to control appearance.

CSS style sheet - A file containing Cascading Style Sheets rules.

The extension replaces `\gls{css-example}` with **CSS style sheet** and links it to the definition. Select the link to jump to the definition.

8.3 Configure glossary terms

8.3.1 Choose the missing-term text

An unresolved term displays `?` by default. Set `unresolved` if your project uses a different marker:

```
[project.markdown_extensions."prodockit.glossary"]
unresolved = "MISSING"
```

`source` is the only other TOML setting. It identifies the current page, but Zensical detects it automatically; leave it unset in `zensical.toml`.

8.3.2 Use a term before its definition

You can use a term before its definition appears on the page:

Markdown

```
This example uses a \gls{css-forward-example} for its layout.

**CSS style sheet** - A file containing Cascading Style Sheets
rules.
{: #css-forward-example data-term="CSS style sheet" }
```

Result

This example uses a CSS style sheet for its layout.

CSS style sheet - A file containing Cascading Style Sheets rules.

8.3.3 Fix a missing term

If a term id is missing or mistyped, the extension displays `?` instead of a link:

Markdown

```
\gls{does-not-exist}
```

Result

?

Check that the text inside the braces exactly matches the id on the definition.

8.3.4 Keep acronyms and glossary terms on separate pages

You can keep acronym expansions on one page and longer glossary definitions on another. `\gls{id}` works with a definition on either page:

Markdown

```
←!── acronyms.md ──→
**CSS** - Cascading Style Sheets.
{: #css .acronym data-term="CSS" }

←!── glossary.md ──→
**Cascading Style Sheets** - The language used to control
appearance.
{: #css-def .glossary data-term="Cascading Style Sheets" }
```

Result

CSS - Cascading Style Sheets.

Cascading Style Sheets - The language used to control appearance.

Link the two entries

Use an ordinary Markdown link when the link text needs to say “glossary” or “acronyms”. The text inside square brackets is what the reader sees:

```
←!── acronyms.md ──→
**CSS** - Cascading Style Sheets. See the [glossary]
(glossary.md#css-def) for what this means in practice.
{: #css .acronym data-term="CSS" }
```

```
<!-- glossary.md -->
**Cascading Style Sheets** - The language used to control
appearance. See the [Acronyms](acronyms.md#css) entry for the
expansion.
{: #css-def .glossary data-term="Cascading Style Sheets" }
```

Use `\gls{id}` to insert the term itself. Use `[link text](page.md#id)` when you want to choose different words for the link.

8.4 Reference

8.4.1 Syntax

Like prodockit.citations, defining and inserting are bundled into one extension: a definition is useless without somewhere to use it.

Defining a term

Any block element - typically a paragraph - with both an `id` and a `data-term` attribute becomes usable with `\gls{id}`:

```
**GUI** - Graphical User Interface.
{: #gui .acronym data-term="GUI" }
```

`data-term` is the text inserted at each `\gls{id}` site - it's stripped from the rendered output (it's internal bookkeeping, not meant to be visible), while `id` stays, since references link straight to it.

Using a term

Syntax	Purpose
`\gls{<id>}`	Insert one term's registered display text and link it to its definition

Unlike `\citeref{...}`, `\gls{...}` only ever takes a single id - there's no multi-term/bracketed form, since inserting a term's own text doesn't compose the way a citation list does.

Like prodockit.refs/prodockit.citations, `\gls{...}` is recognised the same way Python-Markdown's own inline syntax is, so it's protected inside inline code spans and fenced code blocks:

```
Type `\gls{css}` to insert a term.

```
```
```

```
\gls{css}
```

Neither of the two shown above is resolved; both render the literal text.

8.4.2 Zensical settings

Setting	Default	What it controls
`unresolved`	`"?"`	Text shown for a term id that cannot be found.
`source`	`""` (detected automatically)	Advanced: identifies the current page when using the extension outside Zensical. Leave it unset in `zensical.toml`.

8.4.3 Cross-page terms

Under Zensical, a term can be used on a different page from its definition, including an Acronyms or Glossary appendix later in navigation. Prodockit reads definitions across the navigation before page conversion, so no author configuration is required.

Two definitions using the same id produce a warning and the first definition is retained. Use a unique id for every term.

For integration with another Markdown renderer, see Extension integration.

8.5 Customise with a CSS style sheet

`prodockit.glossary` always sets a class on the `\gls{id}` link it renders - resolved or not - so a stylesheet has a stable hook either way:

State	Class
Resolved	`prodockit-gls`
Unresolved	`prodockit-gls prodockit-gls-unresolved`

An unresolved id's `<a>` has no `href` (see Unresolved references above) - style `prodockit-gls-unresolved` distinctly (e.g. a warning colour) to make a missing term visually obvious.

9. Tables

`prodockit.tables` adds layout controls to an ordinary Markdown table. You can change column widths, reduce spacing, use more than one header row, merge cells, and rotate long headings.

9.1 Enable the extension

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.tables"]
```

Choose the feature that solves the table's problem:

Need	Attribute
Set a column width	`width="30%"` or a fixed width such as `8rem`
Fit many short columns	`.compact`
Repeat more than one header row	`.header`
Merge cells	`colspan=2` or `rowspan=2`
Turn a long heading vertically	`rotate=90` or `rotate=270`, with `width`

The next examples show each feature in isolation before combining them.

9.2 Set column widths

9.2.1 Percentages that add up to 100%

Give every column an explicit percentage and they're used exactly as written:

Markdown

```
| Name {: width="25%" } | Description {: width="50%" } | Due {:
width="25%" } |
|---|---|---|
```

```
| Headings | Heading ids and section numbers | Q1 |
| Refs | Cross-references, resolved by number | Q2 |
```

Result

Name	Description	Due
Headings	Heading ids and section numbers	Q1
Refs	Cross-references, resolved by number	Q2

9.2.2 Percentages that don't add up to 100%

Leave a column without a width and it uses the remaining space. If several columns have no width, they share that space evenly:

Markdown

```
| Name {: width="20%" } | Description | Due {: width="15%" } |
|---|---|---|
| Headings | Heading ids and section numbers | Q1 |
| Refs | Cross-references, resolved by number | Q2 |
```

Result

Name	Description	Due
Headings	Heading ids and section numbers	Q1
Refs	Cross-references, resolved by number	Q2

`Name` and `Due` get the widths given; `Description`, left unannotated, takes the remaining 65%. A column left unannotated in a table with no `width` anywhere at all is completely untouched, though - only a table with at least one `width` gets a `<colgroup>`.

9.2.3 Fixed widths for every column

A fixed width is useful when a column should stay the same size even when the page becomes wider or narrower. You can give every column a fixed width:

Markdown

```
| Icon {: width="60px" } | Description {: width="200px" } |
Format {: width="100px" } |
|---|---|---|
| :material-file-pdf-box: | A downloadable PDF | PDF |
| :material-file-document: | A Markdown source file | Markdown |
```

Result

Icon	Description	Format
	A downloadable PDF	PDF
	A Markdown source file	Markdown

9.2.4 Mixing percentages and fixed widths

Percentage and fixed-width columns can appear in the same table. A column can still be left without a width and use the remaining space:

Markdown

```
| # {: width="40px" } | Name {: width="50%" } | Description |
|---|---|---|
| 1 | prodockit.headings | Heading ids and section numbers |
| 2 | prodockit.tables | Column widths on a table |
```

Result

#	Name	Description
1	prodockit.headings	Heading ids and section numbers
2	prodockit.tables	Column widths on a table

9.2.5 Left-aligning a header

By default a header cell is centred and a body cell is left-aligned - the browser's own default styling for `<th>` / `<td>`, unrelated to `prodockit.tables`. To left-align a header too, use Python-Markdown's own column-alignment syntax - a `:` on the left side of that column's own dashes in the separator row - which applies to the header *and* every body cell in that column alike, and combines with `width` on the same header cell with no conflict:

Markdown

```
| Name {: width="30%" } | Description |
|:---|---|
| Headings | Heading ids and section numbers |
| Refs | Cross-references, resolved by number |
```

Result

Name	Description
Headings	Heading ids and section numbers
Refs	Cross-references, resolved by number

`:---:` / `---:` center- or right-align a column the same way - see Python-Markdown's own `tables` docs for the full syntax. This isn't a `prodockit.tables` feature; it's documented here because it's the natural companion to `width` when sizing a column, not something `prodockit.tables` needs to reimplement.

9.3 Configure table layouts

There are no additional `prodockit.tables` settings in `zensical.toml`. Configure each table in its Markdown by adding the attributes shown in the examples below.

9.3.1 Use a compact layout

A table with many short columns can become wider than the page. Add `.compact` to reduce the minimum column width and cell spacing.

Mark it `{: .compact }` on any header cell:

Markdown

```
| Threat {: .compact } | Likelihood | Impact | Risk |
|---|---|---|---|
| Credential theft | H | H | H |
```

Result

Threat	Likelihood	Impact	Risk
Credential theft	H	H	H

Use it only when the normal table is too wide. Put the marker on any header cell. It affects the whole table and can be combined with column widths.

9.3.2 Use more than one header row

A Markdown table normally has one header row. Mark the first additional row with `.header` when a grouped heading needs a second row.

Mark it `{: .header }`:

Markdown

```
| Target {: rowspan=2 } | Measured {: colspan=2 } | | Note {:
rowspan=2 } |
|---|---|---|---|
```

```
| | Before {: .header } | After | |
| Widget | 1 | 2 | ok |
```

Result

Target	Measured		Note
	Before	After	
Widget	1	2	ok

Both header rows then repeat when a long table continues onto another PDF page.

The marker has to go on a cell that **has text** - `attr_list` has nothing to attach to in an empty one. Any cell in the row will do. Only the leading run of marked rows is promoted: a header is the top of a table, and a marked row further down stays where it is rather than the table being quietly re-ordered around it.

9.3.3 Merge cells

Use `colspan` to join cells across columns and `rowspan` to join cells down rows. Keep an empty placeholder for every cell covered by the span, as shown below.

Markdown

```
| Target {: rowspan=2 } | Measured {: colspan=2 } | | Note {:
rowspan=2 } |
|---|---|---|---|
| | Before {: .header } | After | |
| Widget | 1 | 2 | ok |
```

Result

Target	Measured		Note
	Before	After	
Widget	1	2	ok

The empty cell after `Measured` and the empty cells beneath the two `rowspan=2` headings are structural placeholders. The extension removes those placeholders after applying the spans.

9.3.4 Rotate headings

A wide table is often wide because of its headings, not its data. Turn them on their side:

Markdown

```
| Control | Availability requirement {: rotate=270 width="1.8em"
height="105pt" } |
|---|---|
| Backups | H |
```

Result

Control	Availability requirement
Backups	H

`270` reads bottom-to-top, `90` top-to-bottom, and nothing else is allowed: another angle gives a heading nobody can read and a row height nobody can predict.

Set `width` with `rotate`; a missing width is rejected because rotating text alone does not make the column narrower. Set `height` when a long heading needs more room.

9.4 Reference

9.4.1 Syntax

Builds on Python-Markdown's own `tables` extension (auto-enabled if not already present, the same way prodockit.refs auto-enables prodockit.headings).

Attach table attributes to header cells. A column's width is a property of the whole column, so declare it once on the heading rather than on a body cell.

Attribute	Where to put it	Effect
`width="<css-length>"`	Header cell	Set that column's width
`.compact`	Any header cell	Apply the compact layout to the whole table
`.header`	A non-empty cell in a leading body row	Move that row into `<thead>`
`colspan=<n>`	Cell being widened	Merge it with the following placeholder cells
`rowspan=<n>`	Cell being deepened	Merge it with placeholder cells below
`rotate`=90 or `rotate=270`	Header cell that also has `width`	Rotate the heading text
`height="<css-length>"`	Rotated header cell	Reserve height for the rotated text

The minimal width form is:

```
| Column {: width="<css-length>" } | ... |
|---|---|
```

`<css-length>` is any valid CSS width value - a percentage (`"30%"`) or a fixed length (`"120px"`, `"4cm"`, `"3em"`, ...) - passed through to the generated `<colgroup>` as-is, with no validation of its own; an invalid value behaves exactly as it would in any other hand-written CSS, since `prodockit.tables` doesn't parse or interpret it beyond that.

9.5 Customise with a CSS style sheet

The extension adds stable classes that a website CSS style sheet can target:

Element	Condition	Hook
`<table>`	at least one header cell has `width`	`class="prodockit-table-sized"`
`<table>`	any header cell has `.compact`	`class="prodockit-table-compact"`
`<th>`	heading has `rotate=90` or `rotate=270`	`class="prodockit-rotate"` plus an inline transform
`<col>`	that column has `width`	`style="width: <value>;"`

Add at least this rule for sized and compact website tables:

```
.md-typeset table.prodockit-table-sized,
.md-typeset table.prodockit-table-compact {
  table-layout: fixed;
  width: 100%;
  background-color: var(--md-default-bg-color);
  border: 0.05rem solid var(--md-typeset-table-color);
  border-radius: 0.1rem;
  font-size: 0.64rem;
}
```

The complete light- and dark-mode rules used by this site are in `docs/stylesheets/extra.css`. PDF builds already include equivalent layout rules. Contributors changing the generated table structure should read Extension integration.

10. Directory trees

`prodockit.tree` turns an indented list of folders and files into a directory tree. Use it to show readers where files belong in a project.

10.1 Enable the extension

```
[project.markdown_extensions."prodockit.tree"]
```

10.2 Write a tree

Markdown

```
/// tree
docs/ - the documentation source tree
  index.md - the home page
  stylesheets/ - style sheets for the website and PDF
    extra.css - website style sheet
    print.css - PDF style sheet
zensical.toml - project configuration
///
```

Result

- docs — the documentation source tree
  - index.md — the home page
  - stylesheets — style sheets for the website and PDF
    - extra.css — website style sheet
    - print.css — PDF style sheet
- zensical.toml — project configuration

The complete block starts and ends with a three-slash fence. Its body follows three rules:

- **Indentation is the structure.** Two spaces per level by default.
- **A trailing `/` means a directory.** Anything else is a file. Nothing else marks one, so nothing else can disagree with it.
- **`-` starts a description**, and it is optional. The spaces are required, which is what keeps `harvard-cite-them-right.csl` in one piece.

10.3 Configure indentation and icons

There are no additional `prodockit.tree` settings in `zensical.toml`. Configure each tree inside its Markdown block. Put the options directly below `/// tree`, indent them by at least four spaces, and leave a blank line before the listing:

Option	Default	What it does
`indent`	`2`	How many spaces one level costs. Set `4` for a listing written that way.
`directory_icon`	`':lucide-folder:'`	Icon shortcode placed before every directory.
`file_icon`	`':lucide-file:'`	Icon shortcode placed before every file.

This example changes both the indentation and the icons:

Markdown

```
/// tree
    indent: 4
    directory_icon: ':octicons-file-directory-16:'
    file_icon: ':octicons-file-16:'

docs/
    index.md
///
```

Result

docs
└ index.md

10.3.1 Fix indentation errors

A listing is read for its shape, so an entry attached to the wrong parent is a diagram that is wrong and looks right. Rather than guess, the build stops:

```
TreeError: indent of 3 is not a multiple of 2: 'index.md'
TreeError: indented 2 levels at once: 'index.md'
```

10.3.2 A report project

Use the block for a structure readers need to understand. This example shows the main files supplied by `prodockit-template`:

```
docs
├── index.md — cover page
├── section1.md — first report section
├── acronyms.md — acronym definitions
├── glossary.md — glossary definitions
├── references.md — generated reference list
└── stylesheets
    ├── extra.css — website style sheet
    └── print.css — PDF style sheet
tools
├── mermaid — diagram renderer
└── mathjax — maths renderer
zensical.toml — navigation and extension configuration
references.bib — bibliography source
```

The package's own source-code map now lives under Contributor internals.

10.4 Reference

Syntax or option	Purpose
`/// tree`	Open or close a directory tree
A trailing `/`	Mark an entry as a directory
`- description`	Add an optional description
`indent: 4`	Use four spaces for each level instead of two
`directory_icon: '…'`	Choose the directory icon
`file_icon: '…'`	Choose the file icon
`attrs: {...}`	Add an id, class, or other attribute to the tree

The `tree` block follows the same fence, option, and nesting rules as PyMdown Blocks.

10.5 Customise the appearance

The extension adds stable class names that you can target in your project's CSS style sheet. This repository's `docs/stylesheets/extra.css` contains the styles used by the examples on this page.

10.5.1 Generated HTML

The extension provides structure; your project supplies the appearance:

```
<div class="prodockit-tree">
  <ul>
    <li class="tree-directory">
      <span class="tree-icon">…</span>
      <span class="tree-name">docs</span>
      <span class="tree-note">the documentation source tree</span>
      <ul>
        <li class="tree-file">
          <span class="tree-icon">…</span>
          <span class="tree-name">index.md</span>
        </li>
      </ul>
    </li>
  </ul>
</div>
```

Stable class names include `.prodockit-tree`, `.tree-directory`, `.tree-file`, `.tree-icon`, `.tree-name`, and `.tree-note`.

`docs/stylesheets/extra.css` in this repository carries a CSS style sheet to start from. Keep the rail and stub positioned from one shared measurement, so changing indentation cannot pull them apart, and stop the last child's rail at its own stub rather than continuing past the final entry.

11. Numbered steps

`prodockit.steps` presents a procedure as numbered steps that a reader works through in order. Each step has room for a title, instructions, commands, and supporting content. A joining line makes the sequence clear on both the website and in the PDF.

Use an ordinary numbered list for a list of facts. Use `prodockit.steps` when the reader needs to complete one action before moving to the next.

11.1 Enable the extension

Add the extension to `zensical.toml`:

```
[project.markdown_extensions."prodockit.steps"]
```

11.2 Write a procedure

Copy this complete example:

````
/// steps

//// step | Load the key into the agent

```bash
ssh-add --apple-use-keychain ~/.ssh/id_ed25519_gitlab
```

////

//// step | Upload the public key

Paste it into your host's SSH keys page.

////

///
````

It renders as:

1. Load the key into the agent

   ```
   ssh-add --apple-use-keychain ~/.ssh/id_ed25519_gitlab
   ```

2. Upload the public key

   Paste it into your host's SSH keys page.

11.2.1 Opening and closing the blocks

This extension uses the fenced-container syntax from PyMdown Blocks. An opening fence names the block; its closing fence uses the same number of slashes without the name. Nested blocks must use a different fence length. In a numbered procedure:

- `/// steps` opens the complete ordered procedure; `///` closes it.
- `//// step` opens one step; `////` closes it.
- Add a title after `|`, as in `//// step | Upload the public key`.
- Leave off `|` and the title when a step needs only body content.

The blank line after a step header is not required for a simple paragraph, but some block content needs it. Keeping the blank lines shown in the example is the safe, consistent form for paragraphs, fenced code, admonitions, and content tabs.

> ⚠ **Fence lengths must differ when blocks are nested**
>
> The `step` fence uses four slashes because it is nested inside the three-slash `steps` fence. If a step contains another Blocks-based container, give that nested block another unused fence length.

> 📢 **Showing Markdown that contains a code fence**
>
> The copyable example uses four backticks around the complete Markdown sample because the sample itself contains a three-backtick `bash` fence. The outer fence must be longer than a fence inside it.

11.2.2 Content inside a step

A step body is ordinary block Markdown. It can contain multiple paragraphs, code fences, admonitions, or content tabs. For example, the Build your first site walkthrough places macOS, Windows, and Linux commands in tabs inside its first two steps.

11.3 Configure numbered steps

There are no additional `prodockit.steps` settings in `zensical.toml`. Configure each procedure inside its `/// steps` Markdown block.

11.3.1 Continue numbering after a break

A long procedure can be split across sections or pages. Set `start` on the later block so its first step continues at the required number:

Markdown

```
/// steps
    start: 9

//// step | Point the clone at your own project

Update the remote URL before pushing.

////

///
```

Result

1. Point the clone at your own project

   Update the remote URL before pushing.

Options use YAML syntax. Put them directly below `/// steps`, with no blank line between the header and its options, and indent each option by at least four spaces. The blank line after the final option separates the options from the procedure's content.

11.3.2 Add an id or other HTML attributes

Use the Blocks API's `attrs` option to add attributes to the generated ordered list. This complete example combines an id with continued numbering:

Markdown

```
/// steps
    start: 9
    attrs: {id: 'setup-continued'}
```

```
//// step | Verify the remote

Run `git remote -v` before pushing.

////

///
```

Result

1. Verify the remote

   Run `git remote -v` before pushing.

`attrs` is inherited from the underlying Blocks API rather than defined by `prodockit.steps`; `start` is the extension's only steps-specific option.

11.4 Reference

Syntax	Purpose
`/// steps`	Open or close the ordered procedure
`//// step`	Open or close one step without a title
`//// step | Title`	Open one step with a title
`start`: 9	Start this procedure at step 9
`attrs: {...}`	Add HTML attributes to the generated `<ol>`

Each title becomes a separate paragraph before the step body. A step may have no title, but every `step` must be inside a `steps` block for the resulting HTML to be a valid ordered list.

11.5 Customise the appearance

Add the numbered-step styles to your project's CSS style sheet. This repository's `docs/stylesheets/extra.css` contains the styles used by the examples on this page.

11.5.1 Generated HTML

The first example produces this structure:

```
<ol class="prodockit-steps">
  <li>
    <p class="prodockit-step-title">Load the key into the agent</p>
    <pre><code>ssh-add --apple-use-keychain ~/.ssh/
id_ed25519_gitlab</code></pre>
  </li>
  <li>
    <p class="prodockit-step-title">Upload the public key</p>
    <p>Paste it into your host's SSH keys page.</p>
  </li>
</ol>
```

The two stable class names are:

Selector	Element
`ol.prodockit-steps`	The complete procedure
`.prodockit-step-title`	A step's optional title paragraph

The title is an element rather than bold text, so it can be styled separately from emphasis used in the step body.

If you adapt the CSS style-sheet rules in `docs/stylesheets/extra.css`, retain these details:

- the joining line is positioned from the number's own size, so changing `--step-size` does not move the line away from the number;
- the line's `::after` pseudo-element uses the same `font-size` as the number's `::before` pseudo-element, keeping `em`-based measurements aligned;
- the final step has no trailing joining line.

The bootstrap quick start and Build your first site are larger working examples.

Contributors changing the generated representation should read Extension integration.

12. BibTeX bibliography

`prodockit.bibliography` creates citations and a formatted reference list from a `.bib` file. Use it when you have many sources or need a particular citation style, such as Harvard, APA, IEEE, or Vancouver.

For a short reference list that you prefer to write yourself, use Hand-written citations and references.

12.1 Before you start

Pandoc must be installed before you use this extension:

macOS

```
brew install pandoc
```

Windows

```
winget install --source winget --exact --id JohnMacFarlane.Pandoc
```

Linux (Ubuntu)

```
sudo apt update
sudo apt install pandoc
```

Check the installation:

```
pandoc --version
```

See Pandoc's installation guide for its installers and other operating systems.

12.2 Enable the extension

Enable it in `zensical.toml` and set `bib_file` to the name of your bibliography file:

```
[project.markdown_extensions."prodockit.bibliography"]
bib_file = "references.bib"
```

12.3 Add and cite a source

Create `references.bib` in the project root:

references.bib

```
@book{chacon2014,
  author    = {Chacon, Scott and Straub, Ben},
  title     = {Pro Git},
  edition   = {2},
  year      = {2014},
  publisher = {Apress}
}
```

Create a references page and put the bibliography marker alone on its own line:

docs/references.md

```
# References

\bibliography
```

Now cite the entry from any page with `\cite{id}`:

Markdown

```
Git is a distributed version control system \cite{chacon2014}.
```

Result (default style)

Git is a distributed version control system (Chacon and Straub 2014).

The exact punctuation and ordering come from the selected citation style; see Choosing a citation style.

The citation links to the matching entry on the References page. Keep the `\bibliography` marker on that page so the reference list is created.

12.4 Configure the reference list

12.4.1 Choose the bibliography settings

These are the usual project settings:

```
[project.markdown_extensions."prodockit.bibliography"]
bib_file = "references.bib"
csl_style = "harvard-cite-them-right.csl"
unresolved = "?"
```

Setting	Default	What it controls
`bib_file`	`"references.bib"`	Path to the `.bib` file, relative to the directory where you run `zensical build` or `zensical serve`.
`csl_style`	`""` (Pandoc's default)	Path to the `.csl` file that controls citation and reference-list formatting. Leave it out to use Pandoc's default style.
`unresolved`	`"?"`	Text shown for a citation key that cannot be found in the `.bib` file.
`source`	`""` (detected automatically)	Advanced: identifies the current page when using the extension outside Zensical. Leave it unset in `zensical.toml`.

Only `bib_file` is normally required. Add `csl_style` when you need a particular citation style, and change `unresolved` only when you want another missing-citation marker.

Put a bare `\bibliography` marker, alone on its own paragraph, wherever you want the complete, formatted reference list to appear - typically a dedicated References page, kept at the end of `nav` as an appendix, the same convention `prodockit.citations`' own hand-authored references pages already use:

The quick start above uses the default form, `\bibliography`.

Every entry in `bib_file` appears, in the order your chosen CSL style sorts them (alphabetically by default) - not just the ones actually cited, the same way LaTeX's `\nocite{*}` includes every `.bib` entry regardless of whether it's cited in the document. A `\cite{id}` written on any page links directly to its own entry here, adjusted for that page's own directory depth - a real Zensical clean-URL link like `prodockit.refs` / `prodockit.citations` already build, not a bare `#id` fragment that would 404 from a different page.

`\bibliography` takes two optional parameters narrowing this down to just what's actually cited, and/or a different `.bib` file than the configured default - see Multiple sections: References and Bibliography below.

12.4.2 Multiple sections: References and Bibliography

A style guide often distinguishes two different lists:

- **References** (or "Works Cited") - a strict list of only the sources actually cited in the text.
- **Bibliography** - a broader list: everything cited, *plus* background reading the author wants to list even though it's never individually cited inline.

`\bibliography` takes two optional, positional parameters for exactly this:

```
\bibliography{<file>}{<true|false>}
```

- `<file>` - which `.bib` file *this* marker draws from. Leave it empty (`{}`) or omit it entirely to use the configured `bib_file`.
- `<true|false>` - `true` restricts this marker's list to only entries actually `\cite{}`-cited somewhere in the build; `false` (the default, and the only behaviour before these parameters existed) keeps every entry, cited or not.

Write two markers to get both sections from the same `.bib` file:

```
<!-- references.md -->
# References

\bibliography{}{true}
```

```
<!-- bibliography.md -->
# Bibliography

\bibliography{}{false}
```

or from two different files, if background/further-reading sources live separately from the ones actually cited:

```
<!-- references.md -->
# References

\bibliography{references.bib}{true}
```

```
<!-- bibliography.md -->
# Bibliography

\bibliography{background.bib}
```

A `\cite{id}` links to whichever marker's page actually lists that entry, based on which `.bib` file defines it - in the common single-file case (one bare `\bibliography`, as in Quick start above), that's still just the one page, exactly as before. `csl_style` stays a single, extension-wide setting - only the file and cited-only flag are per-marker.

12.4.3 Choosing a citation style

Point `csl_style` at any `.csl` file - your institution's own house style, or one of the thousands available from the Citation Style Language project (the same repository Zotero/Mendeley pull styles from). This project's own docs, and `prodockit-template` / `prodockit-userguide` 's hand-authored references, already follow Cite Them Right Harvard - `harvard-cite-them-right.csl` reproduces that exact style automatically:

```
[project.markdown_extensions."prodockit.bibliography"]
bib_file = "references.bib"
csl_style = "harvard-cite-them-right.csl"
```

renders (confirmed directly against the same `.bib` file used above):

```
<p>Git is a distributed version control system <span
class="citation">(Chacon and Straub, 2014)</span>.</p>
...
<div id="ref-chacon2014" class="csl-entry">
Chacon, S. and Straub, B. (2014) <em>Pro git</em>. 2nd edn. New
York: Apress. Available at: <a href="https://git-scm.com/
book">https://git-scm.com/book</a>.
</div>
```

- exactly the format already hand-typed throughout this project's own reference lists, just generated instead. Leaving `csl_style` unset uses Pandoc's own default (a Chicago author-date style) instead. Confirmed directly: the exact same `.bib` file, with only `csl_style` changed, produces correctly (and very differently) formatted output - author-date parenthetical

citations and a hanging-indent reference list for APA, numbered `[1]` citations and a numbered list for IEEE, and so on - with no other configuration.

12.4.4 Unresolved citations

A key that doesn't resolve to a `.bib` entry renders the `unresolved` marker (`?` by default), unlinked:

```
\cite{does-not-exist}
```

renders `?`, with no link.

12.5 Reference

12.5.1 Syntax

Syntax	Purpose
`\cite{<id>}`	Cite one entry from a configured `.bib` file
`\bibliography`	Generate every entry from the configured `bib_file`
`\bibliography{<file>}`	Generate every entry from another `.bib` file
`\bibliography{<file>}{true}`	Generate only entries cited in the build
`\bibliography{<file>}{false}`	Generate every entry, cited or not

Only a single key is supported - unlike `prodockit.citations`' `\citeref{id1,id2,...}`, a multi-key citation isn't matched by this extension's own syntax at all (falls through as literal text, a visible, honest "not supported" rather than a silently wrong result) - see Comparing the two approaches for why.

Like prodockit.citations, `\cite{...}` is recognised the same way Python-Markdown's own inline syntax is, so it's protected inside inline code spans and fenced code blocks.

`<file>` and `<true|false>` are both optional - see Multiple sections: References and Bibliography above for what they do and when to use them. Put the marker alone on its own paragraph/line.

The extension delegates CSL formatting to Pandoc. Contributors changing that integration should read Extension integration.

12.6 Comparing the two approaches

Both extensions solve the same problem - cite a source by key, get a formatted reference list - but make a fundamentally different tradeoff about where the formatted text comes from. They can be enabled together in the same build without conflict (this project's own docs do, to demonstrate both side by side), though a typical single project only needs one.

	prodockit.citations	prodockit.bibliography
Source of truth	A hand-typed paragraph, once, tagged `data-cite-text`	A `.bib` file entry
Reference list	You write it, by hand, in full	Generated automatically
Citation style	Whatever you typed - one style, fixed	Any CSL style, swappable via one setting
Multi-key citations (`\citeref{a,b}`)	Yes - each key individually linked	Not supported (falls through as literal text)
External dependencies	None	`pandoc` on `PATH`, even without a PDF build
Editing a reference	Edit the prose by hand, on the references page	Edit the `.bib` entry once, everywhere it's cited updates
Separate References/ Bibliography sections	Not built in - would need two hand-authored lists kept in sync manually	Built in - `\bibliography{<file>}{<true|false>}` generates a strict cited-only list and/or a broader everything-included list, see Multiple sections

Where `prodockit.citations` fits best: a short reference list, a house style unlikely to ever change, or a project that doesn't want a `pandoc` dependency for its website build at all (only for its optional PDF, via prodockit.pdf, which already needs `pandoc` anyway).

Where `prodockit.bibliography` fits best: a longer, actively-maintained reference list; needing to match a specific institution's CSL style (or switch between several, e.g. a thesis needing IEEE for one chapter's publications list and Harvard for the rest of the document - one `.bib` file, several instances with different `csl_style` values); or wanting a citation added anywhere in the document to also add it to the reference list, correctly formatted, with no hand-typing at all.

12.6.1 What this project's own template and user guide currently do

prodockit-template has adopted `prodockit.bibliography`, and is a worked example of Multiple sections in a real project: a `references.md` page (`\bibliography{}{true}`) listing only the sources actually `\cite{}`-cited in the document, from a `references.bib` file, and a separate `bibliography.md` page (`\bibliography{bibliography.bib}`) listing everything in a distinct `bibliography.bib` - further reading the author wants to list even though it's never individually cited inline. Both pages share one `csl_style` (Cite Them Right Harvard), configured once on the extension.

prodockit-userguide still uses `prodockit.citations`' hand-authored approach for its own reference list - a short, relatively static one (around ten entries) - and enables `prodockit.bibliography` alongside it only to demonstrate the two coexisting without conflict, the same way this project's own docs do, not as its real reference mechanism. A project outgrowing a short, hand-typed list - a longer, frequently-updated bibliography, or needing to match a specific CSL style - is exactly the case `prodockit.bibliography` is built for, as prodockit-template's own adoption shows.

> **A newer authoring option**
>
> `prodockit.bibliography` is newer and less battle-tested than `prodockit.citations`. The project-wide maturity and compatibility boundary is documented under Support and compatibility.

12.7 Customise with a CSS style sheet

Element	Condition	Hook
`<span>` wrapping a resolved `\cite{id}`	always	`class="prodockit-bib-cite"`
`<span>` wrapping an unresolved `\cite{id}`	always	`class="prodockit-bib-cite prodockit-bib-cite-unresolved"`
Each generated reference-list entry	always	`class="csl-entry reference"`

Every generated reference-list entry also gets `class="reference"` (in addition to Pandoc's own `csl-entry`) - matching the class `prodockit.citations`' own hand-authored entries already use, so `prodockit.zensical_macros`' `reference_style()` / prodockit.pdf's own `reference_style` setting apply uniformly, whether an entry was hand-typed or generated.

13. Index (PDF only)

`prodockit.index` marks important terms where they are discussed and uses those markers to build an alphabetised, PDF-only back-of-book index with page numbers. The marked words remain ordinary inline text on the website, where readers can use Zensical's site search instead.

Use this extension for concepts a printed reader may need to find again. Use Acronyms and glossary when a term needs a separate definition that readers can follow as a link. The actual index page is generated by prodockit.pdf; marking terms alone does not add that page.

13.1 Before you start

Install the optional index support before generating a PDF index:

```
python -m pip install 'prodockit[index]'
```

13.2 Enable the extension

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.index"]
include = true
```

13.3 Mark a term

Mark a term with `\index{Term}`:

Markdown

```
A \index{widget} is the basic unit of work.

Later, this \index{widget} gets combined with a \index{gadget}.
```

Result

A widget is the basic unit of work.

Later, this widget gets combined with a gadget.

Every marked term renders inline exactly as written - `\index{widget}` becomes plain "widget" text, nothing more, on the live website. The marker only has an effect on the *PDF*, and only once `include = true` is set - see Generating the index below for the generated index page itself.

Marking the same term more than once creates one index entry with all of its page numbers.

13.4 Configure the generated index

Set `include = true` in the extension's `zensical.toml` table for a traditional, two-column back-of-book index - terms grouped under a bold letter heading (A, B, C, ...), each followed by the page number(s) it appears on - appended as its own page(s) at the very end of the document:

```
[project.markdown_extensions."prodockit.index"]
include = true
title = "Index"
```

Setting	Default	What it controls
`include`	`false`	Whether `prodockit pdf` generates the index pages.
`title`	`"Index"`	Heading shown on the first generated index page.

You can leave out `title` when “Index” is the heading you want.

The `\index{widget}` / `\index{gadget}` example above renders to an index page like:

```
Index

G
Gadget, 3

W
Widget, 1, 3
```

with its own page list deduplicated and sorted; consecutive pages collapse into an en-dash range (`67–70`) rather than listing every page individually, and non-consecutive pages/ranges are comma-separated (`64, 175`) - standard back-of-book index convention.

A term is alphabetised (and letter-grouped) ignoring any leading punctuation - `--set-upstream option (git push)` and `-u option (git branch)` are filed under **S** and **U** respectively (matching where "set-upstream"/"u" itself would sort), not lumped into a separate "symbols" section, the same way a technical book's own index treats command-line options.

13.4.1 Add sub-entries

`\index{Parent!Child!Grandchild}` - `!` separates up to three levels in practice, matching LaTeX `makeidx`'s own long-established `\index{primary!secondary!tertiary}` convention - nests a term under another, the same way a printed book's own index groups related entries together (e.g. "staging area" with "adding"/"modified" indented beneath it) rather than listing every term as one flat alphabetical run:

Markdown

```
Now generate the \index{Git!ssh keys} to use for authentication.
```

Result

Now generate the ssh keys to use for authentication.

Only the *last* segment displays inline (`ssh keys` above) - wherever the term is actually mentioned in your prose - the earlier segments (`Git`) are only ever used to build the generated index's own nesting, and never appear inline themselves. Renders a nested entry like:

```
G

Git
    ssh keys, 13, 89
```

A parent with no marker of its own anywhere (like `Git` above) still gets its own line - a bare category label with no trailing page list - so its children have somewhere to nest under.

13.4.2 Add code-styled terms

Backticks around the *last* segment mark a command or other code term - it displays inline in a real `<code>` element instead of plain text, and the generated index entry renders the same way:

Markdown

```
Run \index{`git commit`} to save your changes.
```

Result

Run `git commit` to save your changes.

Combine this with sub-entries by putting the backticks around just the last segment - nesting a code-styled `git commit` entry under a plain `Git` one:

```
\index{Git!`git commit`}
```

Both render an index entry in your document's own monospace font, matching how the term already displays inline:

```
G

Git
    git commit, 13, 89
```

> **Showing this syntax as literal example text**
>
> Unlike a plain `\index{Term}` (protected by a real code span, the same way this page's own examples above are written), the code-styled pattern has to run *before* Python-Markdown's own backtick handling, so it can recognise its own inner backticks - a side effect is that wrapping the whole call in inline backticks doesn't protect it the way it does for the plain syntax. A fenced code block (as every example on this page already uses) still works, since it's stashed before any inline pattern - this one included - ever runs.

13.4.3 Add linked terms

A term can be a Markdown link. The link remains clickable in the text while the index files it under the visible link text:

Markdown

```
\index{[Git](https://git-scm.com/)} is a version control system.
```

Result

Git is a version control system.

> ⚠️ **`attr_list` (`{target="_blank"}`) doesn't combine cleanly**
>
> Do not put `{target="_blank"}` after a linked index marker; it attaches to the marker rather than the link. If the link must open a new tab, use a raw inline `<a>` tag:
>
> ```
> \index{<a href="https://git-scm.com/" target="_blank">Git</a>}
> is a version control system.
> ```

13.5 Reference

Syntax	Inline result	Index structure
`\index{Term}`	`Term`	One flat entry
`\index{Parent!Child}`	`Child`	`Child` nested under `Parent`
`\index{Parent!Child!Grandchild}`	`Grandchild`	Up to three levels
`\index{`command`}`	Code-styled `command`	Code-styled flat entry
`\index{Parent!`command`}`	Code-styled `command`	Code-styled child entry
`\index{[Text](url)}`	Linked `Text`	Entry filed as `Text`

Important constraints:

- `!` separates hierarchy levels; only the final segment appears in the prose.
- Backticks for code styling belong around the final segment.
- Use a fenced code block, not an inline code span, when documenting the code-styled marker literally.
- `attr_list` attributes do not attach cleanly to a Markdown link inside an index marker; use a raw `<a>` element when a linked term needs attributes.

13.5.1 How index generation works

Index generation uses two PDF passes. The first pass determines the page of each marked term; the second adds the completed, deduplicated index. Document authors do not need to configure those passes. Contributors changing the pipeline should read Extension integration.

13.6 Customise with a CSS style sheet

Every marked term has the class `index`. Target `.index` if you want marked terms to look different on the website. A code-styled term also contains a normal `<code>` element, so it keeps the project's existing code style.

The attributes used to construct the generated PDF index are an internal pipeline contract. Contributors changing them should read Extension integration.

14. 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. It covers the template, machine setup, local deliverables, continuous integration (CI), automated Pages deployment, and checks on the published result. 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.

14.1 Choose your starting point

Starting point	First guide
You want a ready-made report project	Start with prodockit-template explains what it provides and which files become yours
A new computer or an incomplete checkout	Set up a machine checks and prepares Python, Git, Node, Pandoc, fonts, and the project environment
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 `zensical serve`	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.

14.2 Follow the publishing path

1. Prepare the checkout

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

```
git status --short
```

If this machine has not built the project before, run the report-only setup check:

```
prodockit bootstrap
```

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 PDF first

```
prodockit pdf
```

The PDF uses the pages and order in `zensical.toml`. Build it before the website because Zensical copies the finished PDF into the site output. Skip this step only when the project deliberately publishes no PDF.

See Generate a PDF for the required system tools and optional page, cover, diagram, maths, and index settings.

3. Build the website strictly

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

`--clean` removes output from an earlier run. `--strict` turns validation warnings such as broken internal links into failures. The result is written to `site/` unless the project configures another `site_dir`.

4. 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.

5. 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.

6. 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.

14.3 Know which output you are checking

Output	Built by	Typical location	Final check
Local preview	`zensical serve`	Address printed in the terminal	Edit a page and see it refresh
Static website	`zensical build --clean --strict`	`site/`	Open pages, navigation, links, search, and downloadable files
Complete PDF	`prodockit pdf`	`docs/site_documentation.pdf` by default	Inspect cover, contents, page breaks, diagrams, fonts, and index

Output	Built by	Typical location	Final check
Hosted website	GitHub Pages or GitLab Pages workflow	Project Pages URL	Confirm the public page contains the reviewed change

14.4 Use the detailed guides when needed

- Set up a machine prepares a computer and checkout.
- Start with prodockit-template 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.

15. Command-line tools

This page inventories the public command-line interface (CLI) for document authors and project maintainers. It keeps command names, aliases, safe defaults, write behaviour, and automation semantics visible in one place.

Start with Publish a document to see each command in the task that needs it. Run a command from the project root—the directory containing `zensical.toml`—unless an option explicitly names another location.

Use the shorter `pdk` executable when you prefer it; it is an exact alias. `bootstrap` also answers to `boot`, and `source-bundle` to `source`.

15.1 Check the installation

```
prodockit --version
prodockit --help
```

The first prints the installed release. The second lists the commands supplied by that release. Ask an individual command for its current options:

```
prodockit pins --help
```

If a guide and local help disagree, the local help describes the code you are actually running. Check `prodockit --version`, then compare it with the version pinned by the project before assuming an option is unavailable.

15.2 Choose a command

Command	Use it when	Safe first run	Writes
`prodockit bootstrap`	A machine or checkout is not ready to build and publish	`prodockit bootstrap`	Only with `--apply`; configuration questions use `--configure`
`prodockit init-tools`	The project needs local Mermaid or MathJax rendering tools	`prodockit init-tools`	Tool manifests, scripts, and ignore entries; existing files require `--force`
`prodockit init-mathjax`	The website needs the installed MathJax bundle copied into its assets	`prodockit init-mathjax`	Website JavaScript assets and, unless disabled, `.gitignore`

Command	Use it when	Safe first run	Writes
`prodockit pdf`	You need one PDF containing the pages in `nav`	`prodockit pdf`	The configured PDF output
`prodockit source-bundle`	A submission needs the Markdown and configuration as a separate PDF	`prodockit source-bundle`	The configured source-bundle output
`prodockit sync-repo`	Repository links or badges must match the current remote	`prodockit sync-repo --check`	`zensical.toml` and the managed README badge block without `--check`
`prodockit pins`	Build-input versions disagree or need a reviewed upgrade	`prodockit pins --check --offline`	Matching version declarations when a version is selected
`prodockit template-sync`	A generated project needs later template fixes	`prodockit template-sync`	With `--apply`, template-owned/shared files on a new branch; always appends its ignored log

The `prodockit init-mathjax` command is the narrower website asset command; use `init-tools` when preparing both Mermaid and maths for PDF output.

15.3 Build and preview

Zensical owns the live website commands:

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

`serve` watches the source and rebuilds a local preview. The strict build is the final website check: it starts clean and treats broken links, missing anchors, and other validation warnings as failures.

Prodockit builds the additional artifacts:

```
prodockit pdf
prodockit source-bundle
```

When building both the complete PDF and site, keep this order:

```
prodockit pdf
zensical build --clean --strict
```

Zensical copies the finished PDF into the site directory. Reversing the order can publish the PDF from the previous build while every command exits successfully.

To render one page while developing PDF styles, use:

```
prodockit pdf --markdown-file extensions/tables.md
```

That ignores `nav` and is a quick diagnostic, not a substitute for the final complete build.

15.4 Maintain without changing files

Begin with report-only forms:

```
prodockit sync-repo --check
prodockit pins --check --offline
prodockit template-sync
prodockit bootstrap
```

These answer four different questions:

1. Does repository metadata match `origin`?
2. Do declared build versions agree across files?
3. Has the source template changed files it owns?
4. Is this machine and checkout ready to build?

Do not replace one with another merely because they all use the word “check”.

15.5 Apply and verify a maintenance change

1. Read the report

   Run the non-writing or check form first. A maintenance command should tell you which files or stages are involved before you authorise writes.

2. Apply only the reported change

   Examples:

   ```
   prodockit sync-repo
   prodockit pins --set zensical=0.0.55
   prodockit template-sync --apply
   prodockit bootstrap --apply
   ```

   `pins --set` is unattended and leaves unnamed packages untouched. `template-sync --apply` stages its work on a branch but does not commit it.

`bootstrap --apply` performs only outstanding stages and verifies each one.

3. Repeat the check

```
prodockit sync-repo --check
prodockit pins --check --offline
prodockit template-sync
prodockit bootstrap
```

The second run should be clean or explain any remaining manual work. Do not treat a changed file as proof that the intended state was reached.

4. Build and inspect

```
prodockit pdf
zensical build --clean --strict
git diff --check
git status --short
```

Open the website and PDF when the change can affect rendering. Automated checks catch known failures; visual review answers whether the output is the document you intended to publish.

15.6 Use commands in automation

Automation must not wait for a prompt. Use explicit non-interactive forms:

```
prodockit sync-repo --check
prodockit pins --check --offline
prodockit pins --set zensical=0.0.55
```

Important exit-status behaviour:

Command	Exit zero means
`sync-repo --check`	Managed repository metadata is already current
`pins --check --offline`	Every discovered declaration agrees; no network comparison was attempted
`pins --check`	Declarations agree and none of the selected PyPI packages is behind
`zensical build --clean --strict`	The site built without a strict validation error

Command	Exit zero means
`pytest`	The selected source or built-output checks passed

The ordinary interactive `prodockit pins` command is for a terminal, not CI. Likewise, `template-sync --push` asks before committing, merging, and pushing; it is an assisted maintainer operation rather than an unattended deployment step.

15.7 Find the next guide

- Maintain prodockit provides the complete recurring cycle.
- Set up a machine takes a new computer through its first successful publish.
- Repository metadata explains every derived link and badge.
- Version pinning and drift covers controlled upgrades and scheduled comparisons.
- Staying in step with the template protects project-owned writing while updating shared infrastructure.
- Build and release covers the package release from branch to PyPI and verified Pages deployment.

16. Start with prodockit-template

The `prodockit-template` project (GitHub repository) is a ready-made Zensical project for coursework, assignments, and professional reports. Its central promise is **one source, two outputs**: write the report as Markdown under `docs/`, then build both a browsable website and a single PDF from the same pages and navigation.

Use the template when you want the publishing structure supplied for you. It does not prescribe the subject or wording of the report, and it does not turn your project into a live copy of the template.

The template is maintained on GitHub. Its Surrey GitLab repository is a student-facing mirror of that source, not a separate edition with an independent set of fixes. Surrey students clone the nearby mirror; other projects normally clone GitHub.

16.1 See what the template provides

The starter repository already connects authoring, rendering, testing, and deployment:

- docs — content and appearance
  - index.md — report cover
  - originality.md — originality and AI-use statement
  - section1.md — starter report section
  - acronyms.md — acronym list
  - glossary.md — glossary
  - references.md — formatted reference list
  - stylesheets — website and PDF styles
- zensical.toml — site, navigation, extensions, and PDF settings
- requirements.txt — Python build dependencies
- tools — pinned Mermaid and MathJax Node tooling
- overrides — Zensical theme customisations
- macros.py — shared macros and Surrey environment detection
- .github/workflows/docs.yml — GitHub Pages build and deployment
- .gitlab-ci.yml — GitLab Pages build and deployment

The sample pages demonstrate the enabled prodockit extensions. Replace their starter prose and headings with your report; keep the publishing files until you have a specific reason to customise them.

16.2 Adapt one template to its host

The project-specific `macros.py` defines an `is_surrey` value. It becomes true when the build sees the Surrey GitLab CI host, a Surrey `origin` remote, or a

Surrey address in the Zensical environment. The template uses that value to select the Surrey cover and Surrey logos; otherwise it renders the generic cover and logos.

The choice made in the template cover

```
    Generic cover and branding
```

This keeps the report structure, extensions, build commands, and workflows the same on both hosts. Branding is enabled by where the project is built rather than by asking students to maintain a second configuration file.

16.3 Create a project safely

The recommended route is `prodockit bootstrap`. It checks the computer, clones the correct template for the selected host, separates the clone from the template's Git history, points it at your own empty repository, installs the project environment, and verifies the first push and published site.

1. Install enough to run bootstrap

   Install Python and prodockit first. The machine setup guide provides commands for macOS, Ubuntu, and Windows and explains why bootstrap cannot install the command that is currently running.

2. Configure the project

   ```
   prodockit bootstrap --configure
   ```

   Choose the host, namespace or organisation, project name, local directory, and commit identity. Configuration is written beside the project so a later check uses the same answers.

3. Review the plan

   ```
   prodockit bootstrap --dry-run
   ```

   This prints the outstanding stages and commands without applying them. One stage deliberately replaces the template's Git history with a history of your own; bootstrap marks that destructive stage clearly and does not accept the default answer for it.

4. Apply and verify

   ```
   prodockit bootstrap --apply
   ```

   Bootstrap guides the two browser actions it cannot perform without holding

your credentials: uploading the SSH public key and creating an empty project. It then verifies both rather than assuming a click succeeded.

The source template depends on the selected host:

GitHub

Bootstrap clones `github.com/buckwem/prodockit-template`, then points `origin` at the empty repository you create in your account or organisation.

GitLab.com

Bootstrap uses the GitHub template as its public source, then points `origin` at your GitLab namespace. Your finished project and Pages site remain on GitLab.

University of Surrey GitLab

Bootstrap clones the synchronised Surrey student mirror at `gitlab.surrey.ac.uk/mb0105/prodockit-template`. A GitHub account is not required. The `is_surrey` macro detects that remote and enables the Surrey presentation automatically.

If a course or organisation has already created a repository for you, set `source_url` during configuration. Bootstrap clones that project instead of starting from the public template and keeps its existing history.

16.4 Know what becomes yours

After creation, the repository is your project. The template manifest, `.prodockit-template.toml`, classifies files so a later `prodockit template-sync` can update shared publishing infrastructure without guessing about ownership.

Classification	Examples	Later template update
Project-owned	Markdown and assets under `docs/`, bibliography files, licence, editor and prose-lint choices	Never read for comparison and never written
Template-owned	Pages workflows, `.gitlab-ci.yml`, styles, JavaScript, `macros.py`, `overrides/`, and `tools/`	Updated when the project has not edited the file; a local edit is kept for review

Classification	Examples	Later template update
Shared	`zensical.toml`, requirements files, `.gitignore`, and `README.md`	Merged by setting or delegated to the command that owns that content
Excluded	Template changelog, contributor files, issue templates, and the template's sample regression suite	Not delivered to generated projects

For shared files, the merge is deliberately narrow. Template extension and PDF settings can arrive in `zensical.toml`, but project content such as the author's PDF copyright is not replaced. Dependency versions are left to `prodockit pins`; repository badges are left to `prodockit sync-repo`.

16.5 Keep the project current

A generated project does not change when the source template changes. Check periodically and before a final publication:

```
prodockit template-sync
```

This first run is a report. If an update is useful, the template-sync guide explains how to apply it on a branch, compare `.new` sidecars for files you edited, build both outputs, and publish through the normal review gate.

The template version you started from is not a prodockit package version. Template releases describe starter files; prodockit releases describe the installed extensions and commands. The two can move independently.

17. Machine bootstrap

`prodockit bootstrap` turns the User Guide's install sequence into a list of stages that can be checked individually and repaired one at a time, rather than followed top to bottom and hoped over. `prodockit bootstrap` reports how many there are; the table below names them.

The install is long, sequential, and easy to get half-right in ways that only surface much later - a missing Pango that looks fine until the first `prodockit pdf`, a Node without `npm` that fails in an apparently unrelated step two sections on. Every stage here answers "is this actually set up?", which is the question a written instruction cannot answer for its reader.

17.1 Before you start

> ⚠ **This cannot be the first thing you run**
>
> `prodockit bootstrap` is a prodockit subcommand, so Python and `pip install prodockit` necessarily come first - it cannot install the interpreter it is running on. That is the boundary: **you** install Python, **bootstrap** does the rest.

Which is three steps, on any platform: install Python, make a virtual environment, install prodockit into it.

The virtual environment is not optional on macOS and increasingly not on Linux either. A Python installed by a package manager is marked PEP 668 *externally managed*, and `pip install` outside a virtual environment is refused outright:

```
error: externally-managed-environment
× This environment is externally managed
```

That message is correct and unhelpful in equal measure - it explains what it will not do without saying what to do instead. A virtual environment is what to do instead.

macOS

Install Homebrew if you do not have it, then:

```
brew install python
```

Change to the directory you want to keep your projects in - the one you will later point bootstrap's `project_dir` at - and create the environment there:

```
cd ~/GitLab
/opt/homebrew/bin/python3 -m venv .venv
source .venv/bin/activate
pip3 install prodockit
```

Homebrew's `python3` is named by its full path deliberately. macOS ships a `python3` of its own, and which one a bare `python3` finds depends on `PATH` ordering you did not choose - naming it explicitly makes the environment reproducible rather than dependent on how the shell happened to be configured. On an Intel Mac the prefix is `/usr/local` rather than `/opt/homebrew`; `brew --prefix` prints yours.

Windows

Install Python from python.org/downloads, and on the installer's screens:

- Tick **Add python.exe to PATH** on the first screen. Without it, `python` is not a command and nothing below works.
- Click **Disable path length limit** on the final screen. Node's render toolchains in stage 14 nest deeply enough to hit the 260 character limit.

Then allow PowerShell to run scripts. Windows blocks all of them by default, and activating a virtual environment *is* a script - so without this the very next step fails:

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy
RemoteSigned
```

Depending on your PowerShell version it may ask you to confirm; answer `Y`. Often it simply returns to the prompt, which means it worked.

> **What this changes, and what it does not**

> Without it, activating the environment fails with `... cannot be loaded because running scripts is disabled on this system` - which names the script rather than the policy blocking it, so it reads as a broken file. `RemoteSigned` allows scripts you wrote locally while still requiring a signature on anything downloaded; `-Scope CurrentUser` limits the change to your own account, so it needs no Administrator window and is done once.
>
> Rather not change it at all? Use classic **CMD** instead of PowerShell and run `.\.venv\Scripts\activate.bat` - `.bat` files are not covered by execution policy.

Then, in PowerShell, in the directory you want to keep your projects in:

```
cd ~\GitLab
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install prodockit
```

> ⚠ **Windows ships a `python` that is not Python**
>
> Typing `python` on a machine without it installed opens the Microsoft Store rather than reporting that nothing is there. That placeholder is still ahead of a real Python on `PATH` in some installs, so if `python --version` opens a shop window, the install did not take - repeat it with **Add python.exe to PATH** ticked.

Ubuntu

```
sudo apt install python3 python3-venv python3-pip
```

`python3-venv` is a separate package on Debian and Ubuntu, and its absence does not show up until `python3 -m venv` fails - by which point the error looks like a broken Python rather than a missing package.

Then, in the directory you want to keep your projects in:

```
cd ~/GitLab
python3 -m venv .venv
source .venv/bin/activate
pip install prodockit
```

17.1.1 Every new terminal needs the environment activated again

A virtual environment lasts as long as the shell it was activated in. Open a new terminal window - or come back tomorrow - and `prodockit` is an unknown command again until you activate it.

A new terminal also starts in your home directory, so **change to the directory holding `.venv` first** - the path below is relative to it, and from anywhere else it simply is not there:

macOS / Ubuntu

```
cd ~/GitLab
source .venv/bin/activate
```

Windows

```
cd ~\GitLab
.\.venv\Scripts\Activate.ps1
```

Fails with `running scripts is disabled on this system`? The execution policy has not been set on this account - see Before you start.

Use whichever directory you made the environment in - `~/GitLab` here, matching Before you start above.

You can tell it worked because the prompt gains a `(.venv)` prefix. If it is not there, nothing you install or run is going where you think it is.

17.1.2 Check what you actually got

```
prodockit --version
which prodockit          # `where prodockit` on Windows
```

Worth the two seconds, because the failure this catches is silent. An older `prodockit` already on `PATH` from a different Python shadows a newer one completely: `pip install` reports success, `prodockit --version` reports a version from two releases ago, and `prodockit bootstrap` fails as an unknown command with nothing to suggest why. If the version is not the one you just installed, `which` will show you which Python is winning.

> **`pipx` is the other reasonable answer**
>
> For a command-line tool used across several projects rather than a library imported by one, `pipx` is arguably the better fit - it gives each tool its own environment and puts the command on `PATH` without an activation step, so there is no `(.venv)` to forget. It is one more thing to install, which is why it is not the route above, but if you already have it, `pipx install prodockit` works and skips this whole section.

17.2 Quick start

Six steps from a machine with nothing on it to a published site. Each one is safe to repeat: bootstrap checks before it changes anything, and a stage already done is left alone.

1. Install Python

   prodockit is a Python program, so this is the one thing that cannot be automated - there is nothing to run it with yet. Any version from 3.10 works; 3.13 is what the template's CI uses.

   macOS

   Install Homebrew if you do not have it, then:

   ```
   brew install python@3.13
   python3 --version
   ```

Windows

```
winget install --id Python.Python.3.13 -e
py --version
```

python.org works equally well. Avoid the Microsoft Store build: it is a stub that cannot always create the virtual environment the next step needs.

Ubuntu

`python3-venv` is a separate package on Debian and Ubuntu, and the next step will not work without it.

```
sudo apt install python3 python3-venv
python3 --version
```

2. Install prodockit in an environment of its own

   In the directory you want to keep your projects in - the one you will later point bootstrap's `project_dir` at - creating it if it is not there yet. Not alongside your system Python: Debian and Ubuntu refuse `pip install` outside a virtual environment, and the first stage checks this before anything else, because the project's own environment is built by whichever Python is running bootstrap.

   macOS

   ```
   mkdir -p ~/GitLab
   cd ~/GitLab
   python3 -m venv .venv
   source .venv/bin/activate
   pip install prodockit
   ```

Windows

```
New-Item -ItemType Directory -Force ~\GitLab | Out-Null
cd ~\GitLab
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install prodockit
```

Ubuntu

```
mkdir -p ~/GitLab
cd ~/GitLab
python3 -m venv .venv
source .venv/bin/activate
pip install prodockit
```

`(.venv)` in your prompt means it is active. Every command below is run from here, with it active - see Before you start for the longer version, including what to do when PowerShell refuses to run the activation script.

3. Check what needs doing

```
pdk boot
```

The first run asks what it needs and saves the answers beside the project as `.pdk-bootstrap.toml`. Then it stops, so what it tells you to note down stays on the screen - run it again to see the stages.

On `gitlab.surrey.ac.uk` that is seven questions: your name, the ID you log in with, your course code, and whether the work is assessed. Assessed work is then asked which stage - first attempt, SRA or LSA - and which year the module starts in, and its group and repository name follow from those. Unassessed work is asked for its group and its repository name instead, each offered as your own.

On any other host it is eight, since none of that can be derived there.

`pdk` is `prodockit` and `boot` is `bootstrap`, so `prodockit bootstrap` is the same command typed in full.

4. Read the commands first

```
pdk boot --dry-run
```

Every command it would run, and every step it would ask you to do yourself, without running any of them. Worth one read on a machine you care about.

5. Apply it

```
pdk boot --apply
```

It asks before each stage and shows the commands first. Two steps need a browser - uploading your SSH key, and creating the project on the host - and those ask you to type `yes` when you have done them, because pressing Enter through a browser step is how a run finishes with a stage that never happened.

Stop whenever you like. The next run picks up from wherever it got to.

6. Confirm

```
pdk boot
```

Every stage `ok`, and the last one names the address your site is published at. If a stage still reports work to do, its line says what and why - and running `--apply` again does only that stage.

17.3 What it covers

#	Stage	Automated?
1	prodockit runs in an environment of its own	yes, after a step of your own
2	Visual Studio Code	yes, after a step of your own
3	Git, installed **and** configured	yes
4	SSH keypair	yes, after a step of your own
5	SSH config points at the key	yes
6	Key loaded into the ssh agent	yes, after a step of your own
7	SSH key on the host	**guide and verify**
8	Where the project comes from	**a choice**
9	Project cloned	yes
10	A history of your own	yes

#	Stage	Automated?
11	Your own project on the host	**guide and verify**
12	Pages switched on	**guide and verify**
13	Clone pointed at your project	yes
14	Commit identity in the project	yes
15	Pandoc, and the libraries WeasyPrint needs	yes
16	Project environment and its dependencies	yes
17	Node.js and the render toolchains	yes
18	VS Code extensions	yes
19	VS Code settings for the project	yes
20	Citation style for the first build	yes
21	MathJax for the website	yes
22	First commit pushed	yes, after a step of your own
23	Documentation site published	**guide and verify**

Stages 8 to 14, 18, 19, 21 and 23 do the same thing on every operating system - they are about your project and your host rather than about the machine. The rest differ, because installing software does.

The stages deliberately separate checks, automated plans, and actions that need a signed-in person. The fresh-history stage removes only the template's Git history and requires explicit confirmation; uploading an SSH key and creating the remote project are guided and then verified.

Use `--dry-run` before `--apply` to see which stages are outstanding and which commands will run. Contributors changing stage ordering, check/plan behaviour, subprocess prompting, or destructive-action safeguards should read Bootstrap design.

17.4 Checking without changing anything

Checking is what you get by default - run it with no options at all:

```
prodockit bootstrap
```

`--check` is accepted too, and does the same thing. The read-only behaviour is the default deliberately: the alternative, once applying is implemented, is a

command that starts installing software because somebody typed it to see what it did.

```
 1  MISS  Visual Studio Code - the `code` command is not on PATH
 2  ok    Git, installed and configured - Ada Lovelace
<al01234@surrey.ac.uk>
 3  ok    SSH keypair - /Users/al01234/.ssh/id_ed25519_gitlab
 4  ok    SSH config points at the key - gitlab.surrey.ac.uk uses
id_ed25519_gitlab
 5  ok    Key loaded into the ssh agent - id_ed25519_gitlab is
loaded
 6  ok    SSH key on the host - authenticated to gitlab.surrey.ac.uk
 7  ?     Template cloned - needs project_name
 ...
5 of 18 stages need work.
```

Four states, and the difference between them matters:

	Meaning
`ok`	Set up correctly. A rerun leaves it alone.
`MISS`	Not there at all.
`WRONG`	Present but not usable - git installed with no `user.email`, Node installed without `npm`. **Not** the same as missing, and telling you to install something you already have would send you the wrong way.
`?`	Cannot be judged yet, because it needs a configuration answer you have not given.

Exits non-zero when anything needs work, so it is usable as a check in a script - the same convention as `prodockit sync-repo --check` and `prodockit pins --check`.

17.5 Seeing what it would do

```
prodockit bootstrap --dry-run
```

Prints the exact commands each unsatisfied stage would run, and the instructions for the two that need you. Nothing is executed.

This is worth running before trusting any install tool with your machine, and it is also how a stage is reviewed here: the commands are the thing under test.

17.6 Setting it up

```
prodockit bootstrap --configure   # answer the questions, then stop
prodockit bootstrap --apply       # set up what needs it, asking
first
```

`--apply` walks the stages that need work, showing what it will run before it runs it, and asks each time. The defaults differ by state, and deliberately:

What the plan does	Prompt
Anything that can be undone	`Apply? [Y/n]`
Anything that cannot - stage 8, and only stage 8	`Apply? [y/N]`

One rule, and a visible one. The default used to follow the *check's status* - `MISS` meant yes, `WRONG` meant no - which is a rule you cannot see from the prompt, so the same key press meant different things at different stages for reasons that were never on screen.

Now a plan says whether it destroys something, and only one does.

Every stage is re-checked after it is applied. A command exiting zero says the installer ran, not that the thing it installed works - which is the distinction behind most of the failures this project has had. If a stage runs but still does not check out, bootstrap says so and stops rather than continuing on a broken foundation.

A failing command stops the run too. Later commands in a plan generally depend on earlier ones, so pressing on turns one clear failure into several confusing ones.

17.6.1 Where your part comes in the order

Some stages are part automated and part yours, and *when* your part happens is not cosmetic - it is whether the stage can work at all:

	Example
Before the commands, because they depend on you	The keypair stage: the advice on choosing a passphrase is no use once `ssh-keygen` has already asked for one.
After them, because it depends on the commands	macOS's VS Code: the Command Palette you are asked to open belongs to the application `brew install` has just put there.

Both orderings have been wrong in a shipped release - the install skipped entirely in one direction (#230), and the run stopped dead at the SSH key stage in the

other (#234) - so each stage now states which it needs rather than leaving it to be inferred.

The two guide-and-verify stages are wholly yours, and their verification is the stage's own check rather than a command in the plan. That distinction is what stopped the run in #234: a check is allowed to say "not yet" and be asked again, whereas a command that exits non-zero is a failure and ends the run - and `ssh -T` exits non-zero even when it succeeds.

17.7 Which repository gets cloned

By default, this host's copy of the template - for Surrey that is `gitlab.surrey.ac.uk:mb0105/prodockit-template.git`, its own mirror, so you never need a GitHub account to start.

If you have already been given a repository - a taught module usually issues one per student - put its URL in `source_url` and that is cloned instead:

```
source_url = "git@gitlab.surrey.ac.uk:comm058-2026/report-
al01234.git"
```

The later stages follow on their own: a clone made from `source_url` already has the right `origin`, so the repoint stage reports `ok` and does nothing.

17.8 Configuration

Bootstrap normally stores answers in `.pdk-bootstrap.toml` in the directory where you run it. The file is kept out of Git when that directory is already a repository. An older per-user bootstrap file is still read when no local file exists, so existing setups continue to work.

Pass `--config PATH` when you deliberately want to read and write a different file:

```
prodockit bootstrap --config path/to/bootstrap.toml
```

You should not normally need to open the file. When a run finds an answer missing it offers to ask for it, and only for the ones actually blank:

```
Some details are not set yet: project_name, project_dir.
Answer them now? [Y/n]:
```

`prodockit bootstrap --configure` re-asks everything, with each current value as the default, so pressing Enter through confirms an unchanged setup.

The host is the first question, and deliberately so. Everything else is shaped by it: which URLs the browser steps send you to, which key file is looked for, whether

the thing you are creating is called a project or a repository. Answering it sixth would mean five questions about a setup that might not be buildable at all.

It is asked as a **hostname** - the thing in your address bar - rather than a nickname, and it is judged twice before it is stored.

Is it a supported host?

```
The git host your project lives on [gitlab.surrey.ac.uk]:
bitbucket.org
  bitbucket.org is not a supported host. Choose gitlab.surrey.ac.uk,
  gitlab.com, or github.com.
```

Surrey GitLab remains the default. GitHub.com and GitLab.com can be selected explicitly:

```
The git host your project lives on [gitlab.surrey.ac.uk]: github.com
```

Does it answer?

```
The git host your project lives on [gitlab.surrey.ac.uk]:
  could not reach gitlab.surrey.ac.uk on port 22 - Operation timed
out.
  If this host is only reachable from your university network,
connect
  the VPN and press Enter to try again.
```

That last one earns its place. Without it the first sign of an unreachable host is stage 6 reporting that your key was rejected - after you have made a key and pasted it into a web page - and "I cannot reach this server" looks nothing like "this server refused you". These stages have produced that confusion three times already (#234, #239, #246), so it is worth one connection attempt to tell the two apart at the point the host is named.

Re-asking with the same answer is a real retry, not a loop: connect the VPN, press Enter, and the second attempt succeeds.

Port 22 rather than 443, because every URL bootstrap builds is `git@host:path`, which is ssh.

A configuration written before hostname support stored a key - `host = "surrey"` - and those files are on real machines, so they still resolve. The prompt stores a hostname from now on.

Press Enter to accept Surrey GitLab.

A piped or scripted run never prompts - it reports what is missing and carries on, rather than blocking on a question nobody is there to answer.

The file is stored per **directory**, beside whatever is being set up:

Where	Path
This directory	`./.pdk-bootstrap.toml`
Older, per user (macOS / Linux)	`~/.config/prodockit/bootstrap.toml`
Older, per user (Windows)	`%APPDATA%\prodockit\bootstrap.toml`

One config per directory is one per project. There was a single file per user until 0.32.1, so setting up a second project overwrote the answers for the first - its namespace, its name, the directory it lives in - and the original could not be re-checked without answering everything again.

The per-user file is still read where a directory has none of its own, so a setup already answered keeps working and nothing has to be moved. It is never written to once a local file is possible.

Where the directory is a git repository, `.pdk-bootstrap.toml` is added to `.gitignore`: it holds your name, email and username, and the first push commits everything else in the project.

```
full_name    = "Ada Lovelace"
email        = "al01234@surrey.ac.uk"
username     = "al01234"
host         = "surrey"
namespace    = "comm058-2026"
project_name = "report-al01234"
project_dir  = "~/GitLab/report-al01234"
source_url   = ""
```

> **Never put a secret in this file**
>
> There is no field here for a password, token or passphrase, and that is a design constraint rather than an oversight - the guide-and-verify approach means bootstrap never needs one. A plain file in a synced home directory is the wrong place for a credential, so if a future version ever needs API access, the token belongs in your operating system's keychain and this file holds at most a reference to it.

A malformed line is an error naming the file and line number, not a setting silently ignored - a config that quietly reverted to defaults would re-prompt for everything with no explanation of why.

17.9 Hosts

Bootstrap supports the University of Surrey's GitLab (`gitlab.surrey.ac.uk`), GitHub.com, and GitLab.com. The completed manual end-to-end platform matrix covers Surrey GitLab and GitHub.com. GitLab.com is implemented and tested at command level, but has not yet received the same reported manual coverage across Ubuntu, Windows, and macOS.

Everything host-specific is a *value* rather than a branch: the hostname, the greeting `ssh -T` prints on success, the settings and new-project URLs, and the vocabulary (GitLab's *project* in a *group*, GitHub's *repository* in an *organisation*).

17.10 Status

Checking, `--dry-run`, `--configure`, and `--apply` are covered by the automated test suite. Beyond that command-level coverage, bootstrap has completed manual end-to-end testing on Ubuntu Linux, Windows, and macOS with both Surrey GitLab and GitHub.com.

The manual testing covered two starting points:

1. **A new document repository:** bootstrap prepared the machine and the workflow continued through creating a new hosted document repository to a usable local build.
2. **An existing online repository:** bootstrap prepared the machine and the workflow continued through installing the existing repository locally to a usable local build.

This is a stronger level of evidence than unit tests alone: it exercises real package managers, shells, SSH configuration, repository hosts, clones, and local project setup as one connected workflow. It remains point-in-time manual integration coverage, not an automated cross-platform regression matrix. Linux runs the complete automated suite in hosted CI for every push and pull request. The full test suite is also run locally on macOS, but macOS does not currently have an equivalent hosted pull-request job. Windows has neither a hosted full-suite job nor the same locally repeated regression coverage.

Platform-specific manual stages still remain where the operating system requires them. On Windows, the `ssh-agent` service needs an Administrator window and the PDF fonts have no package-manager installation path. Bootstrap guides those steps and verifies their outcome rather than treating an instruction as proof that it was completed.

18. Staying in step with the template

A project generated from `prodockit-template` is a copy, not a link. The moment it is created it starts to age: the template gains a CI fix, a stylesheet rule, a newer Node pin, and the copy keeps the version it was born with. Nothing tells you, because nothing breaks - the site still builds, the PDF still renders, and the difference only shows up as a document that looks slightly unlike everyone else's.

The `prodockit template-sync` command closes that gap without touching a word of your writing.

Use it periodically while a project is active and before a final release. A long gap is supported, but it produces a larger change that is harder to review and more likely to combine a CI migration with a visual change.

18.1 Choose how far the command should go

Command	Stops after	Use it for
`prodockit template-sync`	A report	Routine checking and learning what changed upstream
`prodockit template-sync --apply`	A new branch with changes staged	The normal reviewed pull-request workflow
`prodockit template-sync --apply --push`	Confirmed commit, merge to the host's default branch, and push	A project where you may merge your own maintenance directly
`prodockit template-sync --apply --force PATH`	Staged update including the named edited file	A deliberate decision to replace your local version with the template's

Protected repositories should normally use `--apply`, inspect the branch, and open a pull request. `--push` is not a bypass for branch protection; it is an assisted path for repositories whose maintainer is allowed to merge directly.

18.2 Complete a template update

1. Start clean and preview

   ```
   git status --short
   prodockit template-sync
   ```

   Uncommitted chapters are allowed, but uncommitted changes to template-

owned files stop the run before it writes. Read the classification counts and any files reported as locally edited.

2. Apply on the generated branch

```
prodockit template-sync --apply
```

The command creates or safely resumes `template-update-...`, writes only the permitted files, and stages them. It does not commit.

3. Resolve kept files and sidecars

For an edited template-owned file, compare the current file with its `.new` sidecar. Merge the useful upstream change by hand, or rerun with one explicit `--force PATH` if the template's complete version should win. Delete resolved sidecars so they cannot be mistaken for unfinished work.

4. Build the complete outputs

```
prodockit pdf
zensical build --clean --strict
```

Run the project's tests as well. A template update commonly changes workflows, stylesheets, Node tooling, or pins; a source diff alone cannot show whether the published PDF and site still look right.

5. Commit and publish through the normal gate

```
git diff --cached
git commit -m "Update from prodockit template"
git push -u origin HEAD
gh pr create
```

Merge after CI passes. If the repository deliberately permits the automated finish, use `prodockit template-sync --apply --push` instead and confirm the printed commit, merge, and push plan.

6. Confirm the next run is clean

After merging or pushing the updated default branch:

```
prodockit template-sync
```

“Already in step with the template” is the verification. A remaining report usually means an unresolved sidecar, an intentionally kept file, or a baseline that was not advanced by the expected branch.

18.3 What it will and will not write

The split is decided by a **manifest** that lives in the template, not by the command, and every file the template ships is classified in it. A file that is in neither list stops the run rather than being guessed at.

Group	Examples	What happens
Template-owned	`.github/workflows/`, `docs/stylesheets/extra.css`, `macros.py`, `tools/`	Replaced, unless you have edited it
Project-owned	`docs/*.md`, `bibliography.bib`, `docs/assets/`	Never written, never read
Seeds	`.vale.ini`, starter pages	Written only if absent
Shared	`.gitignore`, `zensical.toml`	Merged line by line, never replaced
Excluded	`CONTRIBUTING.md`, issue templates	Not delivered at all

> ⚠ **Your writing is not in scope**
>
> The report, its figures and its bibliography are never written, and never even read for comparison. A sync cannot lose your work because it never opens it.

18.4 Running it

Run it from the root of the project - the same directory as `.git`. It refuses anywhere else rather than half-working, because every path it writes is relative to where it started.

```
prodockit template-sync
```

That reports and writes no project file. Read the report, then:

```
prodockit template-sync --apply
```

`--apply` starts a branch of its own, writes, and **stages without committing**. The commit is yours to write, so nothing lands in your history that you have not read first.

18.4.1 The branch it works on

The branch is named after the template version you are moving from, so a second run against the same version continues on the branch the first one made rather than starting again.

That name outlives the run, though, and a branch left over from months ago will not contain anything you have committed since. Continuing on it would sync your project against older files and report success, so a leftover branch that does not contain the commit you are on is refused:

```
Error: the branch template-update-6fbbbbeb8 already exists and does
not
contain the commit you are on, so continuing would run this against
older
work. Merge it, or delete it with `git branch -D template-
update-6fbbbbeb8`,
and run this again
```

You are left on the branch you were already on, with nothing written.

A follow-up run can also make a *second* branch, and that surprises people. The name comes from the baseline you are moving from, so once a run has recorded a stamp, the next one is moving from a different place and branches accordingly - a `--force` run straight after an ordinary one lands on `template-update-<new baseline>` rather than back on the first branch.

Nothing is lost or duplicated: the second branch is made *from* the first, so it contains it, and one merge picks up both. Merging them separately just makes the first look like it did nothing.

18.4.2 Finishing where the pipeline can see it

`--apply` stops at staged, on its own branch. That is deliberate - the commit is yours - but it also means **nothing is published**. Both hosts build only from the default branch, so a sync sitting on a `template-update-...` branch produces no pipeline and no rebuilt site, even after you commit and push it. The steps are spelled out below if you would rather not use this flag.

`--push` finishes the job:

```
prodockit template-sync --apply --push
```

It commits the staged sync, merges it into the branch your host builds from, and pushes - which is what starts the pipeline. It always shows what it is about to do and waits:

```
--push would now, on your confirmation:
  commit  9 file(s) on template-update-6fbbbbeb8
```

```
  merge   template-update-6fbbbbeb8 into main
  push    main to origin - which is what starts the pipeline

Go ahead? [y/N]:
```

Answer anything but yes and the run stops with everything still staged and nothing merged or pushed.

> **Uncommitted writing is fine**
>
> You do not have to commit your chapters first. A project being written always has work in progress, and it travels across the branch switch untouched. Only uncommitted changes to *template-owned* files stop a run, and those are refused earlier, before anything is written.

Which branch it merges into comes from the remote itself, not from your local `origin/HEAD` - that is a cache written when you cloned, and it goes stale. A merge into the branch the sync was written on is refused outright.

Or finish it by hand

If you would rather do it yourself, the steps are the ones `--push` performs — and the middle one is the one people miss:

```
git commit -m "Sync with the template"
```

```
git checkout main && git merge --no-ff template-update-6fbbbbeb8
```

```
git push
```

Committing alone does nothing, and neither does pushing the update branch. A commit is local, so the host never sees it - and neither host builds from a branch like this one. GitLab's `pages` job is guarded by `if: '$CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH'`; GitHub's docs workflow lists `branches: [master, main]`. Different mechanisms, same answer: a pushed `template-update-...` branch produces no pipeline at all.

It is the merge into the default branch, and the push of *that*, which rebuilds the site. A run that appears to have had no effect has almost always stopped at one of those two steps.

> ⚠ **This merges straight into your default branch**
>
> `--push` assumes you merge your own work directly - no merge request, no review, no approval step. That is the assumption this is built on, and it matches how a student works on their own report.
>
> If your project *does* gate the default branch behind merge requests, do not use `--push`: it would either bypass that gate or be rejected by a protected branch. Use `--apply` on its own and open a merge request from the `template-update-...` branch instead.

18.4.3 A file you have edited

A template-owned file you have changed is *kept*, and the template's version is written beside it as `<name>.new` for you to compare. Nothing is overwritten silently.

"Edited" does not always mean you changed it. A file counts as edited when it does not match the baseline the run settled on - which is equally true of a file you customised and one you simply never received an update for. The tool cannot tell those apart, and you usually can, at a glance:

```
diff .gitlab-ci.yml .gitlab-ci.yml.new
```

If the differences are all yours - your module code, your group, your wording - keep what you have. If they are all *the template's own history* - newer pins, a step you have never seen, a comment referring to an issue you did not raise - then you are simply behind, and taking the template's version is right.

> 🧪 **What that looks like in practice**
>
> Syncing a real assignment repository, both files reported as edited turned out to contain nothing project-specific whatsoever. One was a `.gitlab-ci.yml` still pinning `prodockit==0.21.0` against the template's `0.39.0`, missing a `prodockit init-mathjax` step added months earlier - so every page that pipeline had built showed raw TeX instead of maths. Neither file had been edited at all; both had just never been updated.

Taking the template's version

```
prodockit template-sync --apply --force .gitlab-ci.yml --
force .github/workflows/docs.yml
```

Three things to know about `--force`:

- **Exact paths, one flag each.** No globs, and no bare `--force` meaning "everything". That friction is the point: this is the only option that can overwrite your own work, so each file is named deliberately.
- **Paths are as the report prints them**, relative to the project root. A leading `./` is tolerated; anything else will not match.
- **A `--force` that matches nothing is ignored**, not warned about. So the check is the report itself - it should say `forced 2` where it previously said `keep 2`.

Then delete the sidecars

Once you have taken the template's version, the `.new` file beside it is byte-identical to the real one and has no further use. **The tool will not remove it** - it never deletes anything from your project - so it lingers, and gets committed:

```
git rm .gitlab-ci.yml.new .github/workflows/docs.yml.new
```

If you decided to keep *your* version instead, delete the sidecar just the same once you have read it. Leaving it behind means the next reader cannot tell whether it is a decision you made or one you have not got to yet.

18.4.4 Where the template comes from

By default it follows your `origin`: a project on Surrey's GitLab tracks the Surrey mirror, because a student there may have no GitHub access at all. Everything else tracks the canonical GitHub copy. Override it with `--github` or `--surrey`, bare for that host's usual template or with a `group/repo` to name another.

You do not need a copy of the template yourself. The first run clones it into a cache - `~/Library/Caches/prodockit` on macOS, `~/.cache/prodockit` on Linux, `%LOCALAPPDATA%\prodockit\cache` on Windows, or wherever `PRODOCKIT_CACHE` points - and later runs bring that copy up to date. Each host and namespace gets its own entry, so the Surrey and GitHub templates never stand in for one another.

The line under the remote says which of three things happened:

`fetched just now`	first run - the template was cloned
`fetched, up to date`	the cached copy was brought current
`cached copy - could not reach the host…`	the host was unreachable; the run continued on what was already cached

The third is a real answer, not a failure. A run on a train still shows you what your

project would do - it just says plainly that the template it compared against may be behind.

> **A checkout beside your project wins**
>
> If a `prodockit-template` checkout sits next to the project, that is used instead of the cache. This is how the repositories are laid out during development, so a maintainer working across them gets the copy they are editing. `--template-path` names one outright.

18.5 Running it through a project

This is meant to be run repeatedly - every few weeks through a report's development, not once at the start. Most of those runs find nothing, and that case is the one built for.

A run with nothing to do says so and stops:

```
Already in step with the template - nothing to write.
```

No branch, no staged change, nothing to commit. That matters more than it sounds: a run that branched regardless left an empty branch behind, and the branch name comes from the template version, so the *next* run found it in the way.

When the template has moved on, the run branches, writes and stages as usual. A file you have edited keeps its `.new` sidecar from the previous run rather than being rewritten with identical bytes, so `git status` only ever shows what genuinely changed.

> **Simulated over six weeks**
>
> Six syncs across two template versions, with writing committed between each: two produced real updates and branched, four reported "already in step" and left the working tree clean. Two `.new` sidecars at the end - one per edited file, not one per run - all six chapters intact, and the recorded baseline moved forward with the template.

A template release that only reclassifies files leaves every file identical, so nothing is written - but the recorded baseline still moves forward, because leaving it stale would make the next run compare against the wrong version and report unedited files as edited.

18.5.1 After a long gap

A project that has not synced for months takes every upstream release at once. That is the point, but it is worth knowing before you look at the result: in one real catch-up the CI pin moved from `prodockit==0.21.0` to `0.39.0` and `zensical==0.0.53` to `0.0.55` in a single commit - eighteen prodockit releases and two Zensical ones.

So after a large catch-up, **look at the built output**, not just at the diff. If something in the site or the PDF renders differently, the upstream jump is a far more likely cause than the sync itself, and version pinning and drift is the page that covers comparing before and after.

18.6 The log

Every run - reporting or applying, succeeding or failing - appends a full account of itself to `.prodockit-template.log`, and adds that file to `.gitignore` if it is not already there.

```
=== 2026-08-19T14:37:35+01:00  started  prodockit template-sync
template  git@gitlab.surrey.ac.uk:mb0105/prodockit-template.git

  template      15 files  (replace where unedited)
      .github/workflows/docs.yml
      ...
=== 2026-08-19T14:37:39+01:00  finished
```

Two things about it are deliberate:

- **It always holds the `--verbose` form**, listing every file, whatever the terminal was asked for. The run someone reports a problem with is almost always the one they ran with no flags at all.
- **It is written even when the run fails.** A run that stopped partway is the one most worth reading afterwards.

> **Reporting a problem**
>
> Send `.prodockit-template.log`. It carries the command line, both timestamps and the full classification, which is most of what anyone would otherwise have to ask you for.

Entries are appended, never overwritten, so the log is a history rather than a snapshot. Delete it whenever you like; the next run starts a new one.

19. PDF generation

`prodockit.pdf` builds a standalone PDF from your Zensical site - the kind of downloadable, submittable document professional and academic reports commonly need alongside the website itself. It reads the same `zensical.toml` your site already has, so there's nothing new to learn or configure beyond a couple of optional settings.

19.1 Build your first PDF

From the project root—the directory containing `zensical.toml`—run the `prodockit pdf` command:

```
prodockit pdf
```

The command reads every page in `nav`, keeps that order, and writes `docs/site_documentation.pdf` by default. Open the result and check its cover, contents, headings, page breaks, diagrams, and final page before changing any layout setting.

If the command reports a missing program or native library, install the requirements in the next section and repeat the same command. A project that already completed `prodockit bootstrap --apply` should have them.

19.2 Requirements

The PDF is built via Pandoc and WeasyPrint, so both need to be installed and on your `PATH`:

```
pip install weasyprint
```

then follow Pandoc's own install instructions for your platform (e.g. `brew install pandoc` on macOS).

> ⚠ **WeasyPrint is not a pure-Python package**
>
> `pip install weasyprint` installs the Python half. WeasyPrint draws text through Pango and a few related native libraries - `libgobject-2.0`, `libpango-1.0`, `libpangoft2-1.0`, `libharfbuzz`, `libharfbuzz-subset` and `libfontconfig` - which pip cannot install, because they belong to the operating system.

Platform	
macOS	`brew install pango`
Debian/ Ubuntu	`sudo apt install libpango-1.0-0 libpangoft2-1.0-0 libharfbuzz-subset0`
Windows	`pacman -S mingw-w64-x86_64-pango` under MSYS2, with `C:\msys64\mingw64\bin` on `PATH`

One package covers it on macOS and Windows because glib, HarfBuzz and fontconfig arrive as dependencies of Pango. On Debian, `libharfbuzz-subset0` is a *separate* package from `libharfbuzz0b` and is the one usually missed.

On Apple Silicon macOS, Python's dynamic loader may still not search the Homebrew library directory after Pango is installed. Export the path in the terminal where you run `prodockit pdf`:

```
export DYLD_FALLBACK_LIBRARY_PATH=/opt/homebrew/lib
```

Use `/usr/local/lib` on an Intel Mac. If `cannot load library 'libgobject-2.0-0'` appears after `brew install pango`, check this variable before reinstalling anything.

Missing them looks like a Pandoc problem rather than an install one:

```
Error: pandoc exited with status 43 building 'docs/
site_documentation.pdf' (only pass)

Output from the failing command:
...
OSError: cannot load library 'libgobject-2.0-0'
```

Status 43 is Pandoc's own `PandocPDFError` - Pandoc ran, and the PDF engine it handed off to did not start. The detail beneath the error is WeasyPrint's, and names the library it could not load. `python -c "import weasyprint"` is the quickest way to confirm the stack before building.

Back-of-book indexes additionally need `pymupdf` - `pip install prodockit[index]` (or plain `pip install pymupdf`) - but only if you set `include = true` for `prodockit.index`.

Mermaid diagrams and TeX maths need a little Node tooling on top - see below. Every other feature on this page needs nothing beyond Pandoc/WeasyPrint.

19.2.1 Mermaid diagrams and TeX maths

WeasyPrint has no JS engine, so neither can be rendered the way your live website renders them. Both are pre-rendered to static images before Pandoc sees them - Mermaid via mermaid-cli, maths via a small MathJax script.

Set both up with the `prodockit init-tools` command:

```
prodockit init-tools
```

That writes `tools/mermaid/package.json`, `tools/mathjax/package.json` and `tools/mathjax/tex2svg.js` - the exact layout `prodockit pdf` looks for - then prints the `npm` commands to install them, the `.gitignore` lines you want, and the environment variables a CI run needs. It won't overwrite files you already have unless you pass `--force`, and `--no-mermaid` / `--no-mathjax` skip either half.

Once that's in place, maths is ordinary markdown - write `$...$` for an inline formula and `$$...$$` for a display one, and both are rendered wherever the page is read. The right-angle case, inline: $c = \sqrt{a^2 + b^2}$, and as a display formula:

$$c = \sqrt{a^2 + b^2}$$

Those are live rather than illustrative, and are the only maths in these docs. They earn their place: without a real formula somewhere, this project's own rendering checks cannot tell a working MathJax toolchain from a missing one - `assert_no_unrendered_tex` passes trivially on a document with no maths to leave unrendered. Read them in the PDF and they are static SVG; read them on the website and MathJax typeset them in your browser.

Deliberately shown *rendered* rather than as a fenced markdown sample: a code block's LaTeX and an unrendered formula are the same characters once both are plain text in a PDF, so quoting the source here would fail this project's own check - the same trap that made the "contains TeX maths" warning fire on prose describing it. `assert_no_unrendered_tex` matches the handful of TeX command sequences listed in `prodockit.testing.checks`, so keep literal LaTeX out of any page a `-m built` run inspects - naming even one of them in this sentence was enough to fail the check.

> ⚠ **The failure here is quiet by default**
>
> If a renderer isn't found, the content is left exactly as it is rather than failing the build - the right default for a project using neither feature. A project that *does* use them would otherwise get a PDF full of raw `flowchart LR ...`

source or literal LaTeX with nothing having gone wrong as far as the build is concerned, so since 0.12.0 `prodockit pdf` prints a warning naming the missing renderer whenever that combination occurs.

In CI, use `PUPPETEER_SKIP_DOWNLOAD`

mermaid-cli drives Chrome through Puppeteer. The older `PUPPETEER_SKIP_CHROMIUM_DOWNLOAD` is the one most people reach for, and puppeteer 25.x - what mermaid-cli 11.x resolves to - ignores it, so a full Chrome build is downloaded on every run before being discarded in favour of `PUPPETEER_EXECUTABLE_PATH`. `init-tools` prints the correct pair.

19.3 Configure the PDF

Everything is read from your project's own `zensical.toml` - nothing is passed on the command line beyond, optionally, which config file to use:

```
prodockit pdf --config-file zensical.toml   # -f for short; this is
the default
```

19.3.1 Building a single file

To build a PDF from just one markdown file - a single chapter, say, rather than the whole site - pass `--markdown-file` (`-m` for short), a path relative to `docs_dir`:

```
prodockit pdf --markdown-file chapter1.md   # -m for short
```

This ignores `nav` entirely and renders only that page. Everything else - fonts, page size, margins, `heading_numbering`, and so on - still comes from `zensical.toml` exactly as it would for a full build. The output defaults to that file's own name with a `.pdf` extension inside `docs_dir` (e.g. `docs/chapter1.pdf`) instead of `site_documentation.pdf`, unless `pdf_output` is set, in which case that always wins.

Most of what the PDF needs, it already gets from settings your site likely has for other reasons: `site_name`, `copyright`, `repo_url`, `docs_dir`, `theme.font.text` / `.code`, `theme.icon.admonition`, and `extra_css` - your site's own stylesheet(s) are passed straight through, so a `@media print` rule (e.g. hiding a website-only "Download PDF" link/button, since WeasyPrint always renders in print mode) applies in the PDF too. The rest lives under `[project.extra]`, all optional:

Setting	Default	What it does
`pdf_output`	`"<docs_dir>/site_documentation.pdf"`	Where the PDF is written.
`pdf_copyright`	falls back to `copyright`	Overrides `copyright` for the PDF's own footer only - see Copyright text.
`pdf_page_size`	`"A4"`	Any WeasyPrint-supported CSS page size (`"Letter"`, ...).
`pdf_margin_top` / `_right` / `_bottom` / `_left`	`"2cm"`, except `_bottom` at `"2.5cm"`	Page margins, as CSS lengths. The bottom is deeper because the running footer sits in it - see Copyright text.
`pdf_double_sided`	`false`	Duplex-printing layout - see Double-sided (duplex) printing.
`pdf_margin_inner` / `_outer`	`"2cm"` each	Spine-side/fore-edge margins, used instead of `pdf_margin_left` / `_right` when `pdf_double_sided` is on.
`pdf_header_footer_font_size` / `_color` / `_divider_color`	`"10pt"` / `"#555555"` / `"#e2e8f0"`	Running header/footer styling.
`heading_numbering`	`true`	Chapter/appendix numbering on headings and captions.
`reference_style`	`"european"`	`"european"` (tight, single-line citation entries) or `"global"` (double-spaced, hanging indent - the common APA/MLA/Chicago style).
`pdf_include_table_of_contents`	`true`	Whether to generate and insert a table of contents.
`pdf_table_of_contents_title`	`"Table of Contents"`	That page's own heading text.

Setting	Default	What it does
`pdf_mmdc_bin`	auto-detected	Path to a mermaid-cli `mmdc` binary, for pre-rendering Mermaid diagrams. Diagrams are left unrendered if none is found - see Mermaid diagrams and TeX maths.
`pdf_tex2svg_script` / `pdf_math_dir`	auto-detected	A local MathJax `tex2svg`-style Node script, for pre-rendering TeX math (WeasyPrint has no JS engine to run MathJax client-side). Formulas are left as literal text if none is found - see Mermaid diagrams and TeX maths.
`pdf_extra_css`	none	A list of `docs_dir`-relative stylesheet paths, same shape as `extra_css` above but meant *only* for the PDF - e.g. a rule that would look wrong on the live website, or one overriding something `extra_css` itself sets (concatenated after it, so it wins the cascade).

A page's own front matter `is_appendix: true` gives it letter-based numbering ("A", "A.1", ...) instead of numeric, matching prodockit.headings' own `appendix_attr` convention. A page's own front matter `recto_title: "Short Title"` overrides its running header text from the *next* page onward - see Double-sided (duplex) printing. Your `nav`'s index page can also use `{WORDCOUNT}` / `{REPOURL}` / `{RELEASE}` / `prodockit` markers - see Cover page markers.

19.3.2 Web-only / PDF-only content

Mark any block or inline element `{.web-only}` (via `attr_list`) for content meant only for the live website - a "Download PDF" link/button is the common case, since linking to the very PDF you're already reading doesn't make sense once it's embedded in that PDF. `{.pdf-only}` is the opposite: content meant only for the PDF, e.g. an automated word count or release tag on a cover page that only makes sense in a standalone document.

```
[Download this page as PDF](chapter1.pdf){.web-only}

Word count: {WORDCOUNT}{.pdf-only}
```

`.web-only` needs no configuration - `prodockit.pdf`'s own generated CSS always hides it, in every build, whether you're using `prodockit pdf` or calling `build_pdf()` directly. `.pdf-only` is the one half prodockit can't provide automatically (its own CSS has no reach into your live website), so add this one line to your project's own website stylesheet:

```
.pdf-only {
  display: none !important;
}
```

(see this project's own `docs/stylesheets/extra.css` for a working example). If your project doesn't yet use `.pdf-only` for anything, there's nothing to add until it does.

19.3.3 Copyright text

`copyright` (a native Zensical setting, not one of prodockit's own - see Building a single file above) feeds both the live website's own footer *and* the PDF's running footer, by default - whatever you set once shows, unchanged, in both places.

`pdf_copyright` (under `[project.extra]`) overrides it for the PDF's own footer only, leaving the website's copyright text completely untouched either way. Unset by default, so an existing project's PDF and website keep matching unless you deliberately add it - useful when you want the PDF to show something the website version wouldn't make sense showing (or vice versa), without having to keep two near-identical strings in sync by hand.

Both `copyright` and `pdf_copyright` accept a real HTML fragment, the same as Zensical's own website-side `copyright` setting already does - a real `<a href="../...">` link renders as a real, clickable link in the PDF too, not flattened to plain text. Use a real `<br>` for a forced line break. The obvious use: crediting the tools your report was built with, on their own second line, without touching the copyright/licence text itself:

```
[project.extra]
pdf_copyright = 'Author: Jane Doe. Licensed under the MIT
License.<br>Made with <a href="https://zensical.org/">Zensical</a>
and <a href="https://buckwem.github.io/prodockit-
extensions/">prodockit</a>.'
```

Links and line breaks remain real PDF content rather than being flattened to plain

text. Contributors changing the footer implementation should read PDF pipeline and API.

The equivalent website-side credit (if your project wants a "Made with Zensical and *X*" line on the live site too) isn't a prodockit setting at all - prodockit has no reach into the website's own Jinja partials. It's a Zensical theme override instead: with `custom_dir` set (see Zensical's own docs), drop your own `overrides/partials/copyright.html` based on the bundled version, adding a second credit line after the existing "Made with Zensical" one.

> **Why the bottom margin is deeper than the others**
>
> The footer is top-aligned in the bottom margin and grows *downward* as it gains lines, so whatever the margin does not use is the space left before the paper edge. A two-line footer at a 2cm bottom margin ends about 6.1mm from the edge - inside the 5-6.4mm many consumer and office printers cannot print at all, so the second line risks being cropped even though the PDF itself is correct.
>
> `pdf_margin_bottom` therefore defaults to `2.5cm`, which leaves about 11.1mm. A footer of three or more lines needs more again: set `pdf_margin_bottom` explicitly and check the result on paper, not just on screen.

19.3.4 Cover page markers

Drop any of these literal strings into your `nav`'s index page - typically a cover page, e.g. wrapped in `{.pdf-only}` as in the example above - and `prodockit pdf` substitutes a real value once that page's HTML exists, no configuration needed:

Marker	Becomes
`{WORDCOUNT}`	The site-wide word count (the same value a `54,092` website macro variable would show), so a submission's PDF and its live website page never disagree.
`{REPOURL}`	The git-detected repo URL (the same value `https://github.com/buckwem/prodockit-extensions` gives a website macro).
`{RELEASE}`	The latest published GitHub/GitLab release tag (e.g. `v1.2.0`). The *whole line* containing this marker is dropped instead if there isn't one - most projects never publish a release at all, so nothing shows a bare `"Release: "` label by default.

Marker	Becomes
`prodockit`	Your project's own `site_name`, substituted literally - `prodockit pdf` never evaluates Jinja, so the exact same `prodockit` text a website macro variable uses works here too, one line of markdown for both outputs.

Skipped entirely for a `--markdown-file`-scoped build, or if your `nav` has only one page - there's no separate "cover" vs "content" to compute a word count from either way.

19.3.5 Landscape pages

Anything too wide for a portrait page - a reference table, a diagram, a chart - can be given landscape page(s) of its own instead: wrap it (and its own caption) in `<div class="landscape-page" markdown="1">`, using `md_in_html` (the `markdown="1"` is required - without it, the content inside is left as literal, unconverted text):

```
<div class="landscape-page" markdown="1">

**A wide reference table**

| ID {: width="15%" } | Description {: width="70%" } | Due {:
width="15%" } |
|---|---|---|
| 1 | ... | Q1 |

</div>
```

It is not limited to tables. A Mermaid diagram, an image, a wide code block - whatever is in the block gets the page:

```
<div class="landscape-page" markdown="1">

![Architecture overview](assets/images/architecture.png)

</div>
```

The content prints on its own landscape-sized page(s) - the same configured page size, width and height swapped. A page break is always forced immediately before and after the block, so it never shares a page with anything else.

Content longer than one page simply carries on. A table spanning several landscape pages repeats its header row on every one of them, exactly as it would on a portrait page - measured directly: a 90-row table produced five landscape

pages, each carrying the header (see prodockit.tables for the `width` syntax above, which works the same way here).

A document mixing portrait and landscape pages prints without any special handling - a PDF reader rotates each page to fit the paper on its own.

This is PDF-only - the same wrapped content renders completely normally on the live website, the same way `.web-only` content elsewhere in this project only ever affects one of the two outputs.

19.3.6 Double-sided (duplex) printing

Set `pdf_double_sided = true` under `[project.extra]` for a document meant to be printed and bound on both sides - a book or handbook, rather than a web-printed report. Left-hand (verso) and right-hand (recto) pages mirror their header/footer content and page margins, and every numbered heading starts its own recto page:

```
[project.extra]
pdf_double_sided = true
pdf_margin_inner = "3cm"   # spine side - wider, to leave room for
binding
pdf_margin_outer = "1.5cm" # fore-edge (outer) side
```

`pdf_margin_inner` / `pdf_margin_outer` replace `pdf_margin_left` / `_right` once `pdf_double_sided` is on - the "inner" (spine) side is the left margin on a recto page but the right margin on a verso page, and vice versa for "outer" (fore-edge), so a single pair of settings covers both without you having to think about which physical side is which for any given page. `pdf_margin_top` / `_bottom` are unaffected either way.

Every corner of the running header/footer mirrors between recto and verso, keeping the chapter title and page number on the outer, fore-edge corner and the site name/copyright on the inner, spine-side corner, whichever physical side that happens to be for a given page - confirmed directly, by rendering a real double-sided document and inspecting facing pages, that this is how it actually looks.

Every numbered heading (chapter start) also always starts on its own recto page - a blank page is inserted automatically if the previous chapter ended on an odd page, exactly like the blank pages you'd expect at the start of each chapter in a real printed book. This needs no configuration; it's part of what `pdf_double_sided` turns on.

A page's own front matter `recto_title: "Short Title"` overrides that page's own running header text with a shorter title, from the *next* page onward (the heading's own page still shows its full title) - handy when a chapter's real title is too long to comfortably fit the running header:

```
---
recto_title: "Ch. 1"
---

# Chapter One: A Rather Long Title That Wouldn't Fit In A Running
Header
```

This setting is meaningful whether or not `pdf_double_sided` is on - the running chapter title appears in the header either way, just in a different corner.

19.3.7 Bundling source into a PDF

The `prodockit source-bundle` command builds a second PDF - your Markdown content and `zensical.toml`, one file per page - for a submission that needs the underlying source alongside the rendered document:

```
prodockit source-bundle
```

This is separate from `prodockit pdf` because the two files serve different purposes: one is the rendered document and the other is a record of its source. Run each command only when you need that output.

Writes `docs_dir/source_bundle.pdf` by default, so Zensical serves it with no separate copy step. Override with `pdf_source_bundle_output` under `[project.extra]`:

```
[project.extra]
pdf_source_bundle_output = "dist/source.pdf"
```

The running header's report name is your `site_name`; the page size is `pdf_page_size` - the same setting `prodockit pdf` reads, so both PDFs a project publishes share one physical page size rather than needing it set twice.

Every file is rendered in 8pt Courier with wrapped lines (a genuinely long line wraps rather than running off the page or getting cut off), starting on its own page, with a running header (that page's own file path on the right) and a "Page N of M" footer.

Which files are included: every `.md` file under `docs_dir` (recursively) plus `zensical.toml` itself - your documentation's own source, not the project's tooling around it. A file that isn't valid UTF-8 text is silently skipped rather than failing the build, though in practice that never applies here (Markdown and TOML are always text).

> **Need to bundle more than the document source?**
>
> The command deliberately includes only Markdown pages and `zensical.toml`. Contributors building a custom bundle can use the Python API described in PDF pipeline and API.

19.3.8 Table of contents and bookmark outline

A PDF built by `prodockit pdf` has two separate tables of contents, built by two different tools:

- The **Table of Contents page** itself (`pdf_include_table_of_contents` / `pdf_table_of_contents_title` above) - generated by Pandoc from every heading it sees, via `pandoc.structure.table_of_contents()`.
- The **bookmark outline** - the navigation pane a PDF reader shows down the side, e.g. Adobe Reader's or a browser's own PDF viewer's sidebar. This is built separately by WeasyPrint, which bookmarks every `h1`-`h6` straight from its own UA stylesheet.

prodockit.headings's `unlisted` class (Pandoc's own, not a prodockit invention) keeps a heading off the Table of Contents page. It has no effect on the bookmark outline - an `.unlisted` heading still becomes an outline node, and because outline nesting follows heading level, every later heading of lower level nests underneath it instead of under its real chapter. Add `unbookmarked` too (prodockit's own generated CSS gives `h1.unbookmarked`-`h6.unbookmarked` `bookmark-level: none`) to remove a heading from the outline as well:

```
# Illustrative Example {: .unnumbered .unlisted .unbookmarked }
```

Nothing `prodockit pdf` generates itself carries `unbookmarked` - the back-of-book index's own A/B/C letter headings, the Table of Contents title, and every cover-page heading are all `unnumbered unlisted` (so a reader can still navigate to them) but deliberately *not* `unbookmarked`, since they belong in the outline. Reach for `unbookmarked` yourself only for a heading that shouldn't be there at all - e.g. an illustrative heading in documentation that demonstrates real rendered output rather than a code sample (see this project's own docs/extensions/refs.md).

19.3.9 Back-of-book index

A traditional, two-column back-of-book index, generated from every `\index{Term}` marker when `include = true` in the `prodockit.index` extension's settings. It is PDF-only; there is no equivalent on the live website. Marking terms, turning the setting on, and what the generated page itself looks like are all covered together in Index (PDF only), since (unlike every other feature on this page) marking and generation are two different extensions - see that page

for the full syntax and worked examples. Contributors scripting a custom build can read about the two-pass implementation in PDF pipeline and API.

Contributors calling the Python API or changing the HTML, Lua, CSS, Mermaid, source-bundle, or index stages should use PDF pipeline and API.

19.4 Limitations and workarounds

`prodockit.pdf` pipes your site's own rendered HTML through Pandoc and WeasyPrint to produce the PDF - two tools with their own reader/writer quirks and no JS engine, quite different from a browser rendering your live website. See Known limitations for the confirmed limitations this shapes in `prodockit.pdf.html` / `.lua` / `.css`, and the workaround each one gets.

For supported tool versions, platforms, and the pre-1.0 stability boundary, see Support and compatibility.

20. Publish automatically

This page is for a document author who wants a reviewed Markdown change to become a website and PDF automatically. continuous integration (CI) runs the publishing commands on a clean hosted machine after a push, then hands the static website to GitHub Pages or GitLab Pages.

You should not have to design that automation. `prodockit-template` supplies the maintained files; this page explains how to use them, what they build, and how to tell whether publication really succeeded.

On GitHub, GitHub Actions runs the workflow. On GitLab, the equivalent pipeline is GitLab CI.

20.1 Use the maintained automation files

Host	Maintained source	What runs
GitHub Pages	`docs.yml`	Builds the outputs, uploads the site, deploys Pages, and verifies the public site
GitLab Pages	`.gitlab-ci.yml`	Builds the outputs and publishes the `public/` Pages artifact

The comments in those files explain settings that must remain beside the commands they control: pinned system tools, browser variables, fonts, build order, Pages permissions, and artifact paths. Refer to the files for exact YAML rather than copying a workflow from this guide.

Projects created from `prodockit-template` already contain both files. If an older template-derived project is missing later workflow fixes, preview them without writing:

```
prodockit template-sync
```

Review and apply the result using Staying in step with the template.

20.2 Publish the first change

1. Prove the project builds locally

```
prodockit pdf
zensical build --clean --strict
python -m pytest
```

Fix a local failure before pushing. CI starts from a clean machine, so it cannot repair a missing page, broken link, or failing test that is already reproducible in the checkout.

2. Confirm the publishing file is present

GitHub Pages

Confirm `.github/workflows/docs.yml` exists. It should run when the repository's default branch changes and should allow a manual run for recovery.

GitLab Pages

Confirm `.gitlab-ci.yml` contains a `pages` job. GitLab reserves the name `public/` for the directory that job publishes.

Use the maintained links above to compare a file that appears incomplete. Do not replace a customised workflow until you have reviewed the difference.

3. Push through the review gate

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

Open a pull request or merge request. After its checks and review are complete, merge it into the branch the publishing workflow watches—normally `main`.

4. Watch the publishing job

GitHub Pages

Open **Actions**, select **Documentation**, and open the run for the merged commit. Check both the deployment and later live-verification result.

GitLab Pages

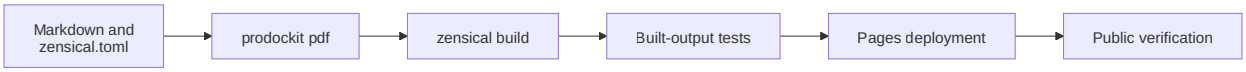
Open **Build > Pipelines**, select the pipeline for the merged commit, and inspect its `pages` job. Then use **Deploy > Pages** to find the published address.

5. Verify delivery as a reader

   Open the Pages address in a private browser window. Find the known change, follow a navigation link, and download the PDF from the site.

   A successful build proves an artifact was produced. A successful deployment proves the host accepted it. Opening the public result proves a reader can retrieve the intended version; these are three separate checks.

20.3 Understand the build order



The PDF is built before the website because it lives under `docs/` by default. Zensical copies it into the static site together with other downloadable files. Reversing the commands can publish the PDF left by an earlier run.

Some projects also run `prodockit source-bundle` before Zensical. That creates a second downloadable PDF containing the Markdown and configuration rather than the rendered report.

20.4 Know what the hosted machine needs

Requirement	Used for	Failure when absent
Python requirements	Zensical, prodockit, WeasyPrint, and tests	The command normally fails
Pandoc	PDF conversion and `prodockit.bibliography`	The build fails
WeasyPrint native libraries	PDF layout	Import or PDF build fails
Document fonts	Correct PDF typography and pagination	A fallback font may be substituted silently
Node, Mermaid CLI, and Chrome	Mermaid diagrams in the PDF	The PDF can contain raw diagram source

Requirement	Used for	Failure when absent
Node and MathJax	TeX maths in the PDF and website bundle	The output can contain raw TeX
Citation style files	Bibliography formatting	A configured missing style stops rendering
Suitable Git history or release metadata	Version text used by a cover or macro	The field can be empty or one release behind

The maintained workflow files install these in dependency order. A document author normally changes the content and requirements, not the operating-system recipe.

20.5 Catch failures that still produce output

20.5.1 Render diagrams and maths

WeasyPrint has no JavaScript engine. The PDF build therefore turns Mermaid and TeX maths into static images before Pandoc sees the pages. If either renderer is missing, the command warns but can still create a PDF.

Build-output tests make that warning enforceable:

```
from prodockit.testing import assert_no_unrendered_mermaid,
assert_no_unrendered_tex


def test_diagrams_and_maths_rendered(prodockit_pdf_page_texts):
    assert_no_unrendered_mermaid(prodockit_pdf_page_texts)
    assert_no_unrendered_tex(prodockit_pdf_page_texts)
```

The template workflow uses `PUPPETEER_SKIP_DOWNLOAD`, not the obsolete `PUPPETEER_SKIP_CHROMIUM_DOWNLOAD`, and points Mermaid at the Chrome installed by the job. Keep those details in the workflow where future package updates can change them.

20.5.2 Check embedded fonts

The website can download fonts when a browser opens it. A PDF must embed fonts available on the build machine. WeasyPrint may silently substitute another font, changing appearance, line wrapping, pagination, and index page numbers.

Use the template's font packages as the starting point and add an output test when the project requires a particular typeface.

20.5.3 Fetch tags only when the document uses them

The `prodockit-v0.41.0` macro reads Git tags reachable from the checked-out commit. GitHub's default shallow checkout has no tags, so the value becomes an empty string without failing the build. Set a full checkout only when the document uses tag-derived macros; the template's PDF-only `{RELEASE}` marker uses host release metadata instead.

20.5.4 Test the artifact, not only the source

`prodockit.testing` opens the generated site and PDF. Start with checks that prove the required outputs exist, then add document-specific expectations:

```
def test_the_pdf_was_built(prodockit_pdf):
    assert prodockit_pdf.page_count > 1


def test_the_site_has_the_cover(prodockit_soup_for):
    cover = prodockit_soup_for("index.html")
    assert cover.title is not None
```

The template's own sample-output checks describe its starter document and are advisory because real projects replace that content. A generated project should replace them with assertions about its own required output and decide which must gate publication. See Test the built output.

20.6 Troubleshoot a publishing run

Symptom	Start with
The same command fails locally	Fix the source or local configuration before investigating CI
A tool is absent only in CI	Compare the workflow with the maintained template file and its pinned requirements
The PDF contains raw Mermaid or TeX	Check the Node installs, Chrome path, and built-output tests
The PDF uses the wrong font	Check the operating-system font packages and inspect embedded fonts
The website builds but the PDF link is stale	Confirm the PDF command runs before Zensical and both use the same artifact directory
GitHub deploys but the public page is old	Inspect the workflow's live-verification job and rerun the maintained workflow against the default branch
GitLab succeeds but no site is visible	Open **Deploy > Pages**, then check project/instance Pages visibility and the `public/` artifact

Repository release numbering, dependency drift, and workflow maintenance are covered in Maintain prodockit. Existing links to the former sections above remain valid.

21. Test the built output

`prodockit.testing` gives a project pytest fixtures pointing at its own *built* output - the site directory and the PDF - plus checks for the failure modes that are the same in every prodockit project.

Install it with:

```
pip install prodockit[testing]
```

The fixtures test artifacts that already exist; they never build anything. Run your builds first:

```
prodockit pdf
zensical build --clean --strict
python -m pytest
```

21.1 Quick start

No `conftest.py` wiring is needed - the fixtures register themselves through pytest's plugin entry point:

```
from prodockit.testing import assert_no_unrendered_mermaid,
assert_no_unrendered_tex


def test_the_pdf_built(prodockit_pdf):
    assert prodockit_pdf.page_count > 5


def
test_diagrams_and_maths_actually_rendered(prodockit_pdf_page_texts):
    assert_no_unrendered_mermaid(prodockit_pdf_page_texts)
    assert_no_unrendered_tex(prodockit_pdf_page_texts)
```

21.2 Why the rendering checks exist

`prodockit.pdf` pre-renders Mermaid diagrams and TeX maths to static images, because WeasyPrint has no JS engine. When a renderer isn't installed, the content is left exactly as it is rather than failing the build - the right default for a project using neither feature, and a silent disaster for one that does.

Three separate projects published PDFs containing raw `flowchart LR ...`

source and literal LaTeX before anyone noticed. `prodockit pdf` warns about it since 0.12.0, but a warning in build output is easy to scroll past. These checks turn it into a test failure.

They are deliberately narrow about what counts as evidence. Several Mermaid diagram types are also ordinary English words - `graph`, `pie`, `journey`, `timeline` - and line breaks in a PDF fall wherever the text happens to wrap. A check that flagged any line starting with one of those read "a visual commit graph and richer history browsing" as an unrendered diagram, passing locally and failing in CI only because different fonts there wrapped the sentence differently. So a diagram-type keyword is only evidence when Mermaid's own link syntax follows shortly after it.

21.3 Fixtures

All are session-scoped and prefixed `prodockit_`, so they can't collide with names in your own `conftest.py`.

Fixture	What it gives you
`prodockit_paths`	Resolved `root`, `config_file`, `docs_dir`, `site_dir`, `pdf`
`prodockit_config`	Your Zensical config as plain parsed TOML
`prodockit_resolved_config`	The same, through Zensical's own loader - `nav` resolved to a tree
`prodockit_nav_pages`	Every nav markdown file, `docs_dir`-relative, in nav order
`prodockit_pdf`	The built PDF, opened with `pymupdf`
`prodockit_pdf_page_texts`	The PDF's text, one string per page
`prodockit_site_dir`	The built site directory
`prodockit_site_html_files`	Every built HTML page, sorted
`prodockit_soup_for`	Factory: parses one built HTML file with BeautifulSoup

Paths come from your config rather than an assumed layout: `site_dir` defaults to `site` but is commonly set to `public`, and the PDF follows `pdf_output` when you set it.

21.4 Configuration

Two `pytest` ini options, both usually unnecessary:

Option	Default	Purpose
`prodockit_config_file`	`zensical.toml`	Your Zensical config, relative to the pytest rootdir.
`prodockit_pdf`	from the config	Override the PDF location.

```
[pytest]
prodockit_pdf = dist/report.pdf
```

Paths resolve against pytest's rootdir

Not against the test file. If your tests live outside the project root, or you invoke `pytest` from elsewhere, set `prodockit_config_file` or pass `--rootdir`.

21.5 Checks

From `prodockit.testing`:

Function	Purpose
`assert_no_unrendered_mermaid(page_texts)`	Fails if any page carries raw Mermaid source.
`assert_no_unrendered_tex(page_texts)`	Fails if any page carries raw TeX.
`find_unrendered_mermaid_pages(page_texts)`	The offending page indexes, for a custom message.
`find_unrendered_tex_pages(page_texts)`	As above, for maths.
`contains_unrendered_mermaid(text)`	Single-page predicate.
`contains_unrendered_tex(text)`	Single-page predicate.

Both assertions name the fix (`prodockit init-tools`) in their failure message rather than only reporting the symptom.

Contributor guidance for keeping the automatically discovered plugin lightweight lives under Development and code map.

22. Website macros

`prodockit.zensical_macros` provides a handful of Jinja variables and macros for Zensical's own macros plugin - the pieces a professional/academic report's website commonly wants that aren't specific to any one project: a site-wide word count, the git-detected repository URL, the latest release tag, chapter/appendix numbering that continues across pages, and reference/acronym/glossary list spacing that matches `prodockit.pdf`'s own PDF output.

22.1 Quick start

Add it alongside your own project's `macros.py` (which keeps anything genuinely project-specific - a custom macro, institution branding, and so on):

```
[project.markdown_extensions.zensical.extensions.macros]
module_name = "macros"
modules = ["prodockit.zensical_macros"]
```

Zensical's macros plugin loads `module_name` and every entry in `modules`, merging all of their variables/macros into the same Jinja environment - so a project with no macros of its own can drop `module_name`/`macros.py` entirely and just use:

```
[project.markdown_extensions.zensical.extensions.macros]
modules = ["prodockit.zensical_macros"]
```

22.2 Variables

Variable	Description
`54,092`	Prose word count across every nav page except the first (assumed to be the cover page) and any page flagged `exclude_from_word_count: true` in its own front matter - a comma-formatted string (e.g. `"9,971"`).
`https://github.com/buckwem/prodockit-extensions`	The fully-qualified `https://` URL for the current checkout's git `origin` remote (converted from `git@host:path.git` SSH syntax, with any embedded CI credentials stripped) - `""` if there's no git remote configured.

Variable	Description
`prodockit-v0.41.0`	The latest git tag reachable from `HEAD` (e.g. `"1.2.0"`) - `""` if this checkout has no tags at all. Resolves identically for the website and for `prodockit pdf`, since both render through this same macro environment - unlike `prodockit.pdf`'s own `{RELEASE}` cover-page marker, which queries the host's GitHub/GitLab API instead, for a project whose cover page isn't part of a live, macro-rendered site at all.
`prodockit`	`project.site_name` from `zensical.toml`.

22.3 Macros

Macro	Description
``	Place near the top of every page - continues chapter/section numbering (and the matching sidebar numbering) across pages, from this page's position in nav. See below.
`` `| Place once near the top of a `references page - controls`.reference` `` paragraph spacing. See below.	
`` `| Place once near the top of an `acronyms page - matches`reference_style()`'s default spacing.	
`` `| Place once near the top of a `glossary page - matches`reference_style()`'s default spacing.	

22.3.1 `heading_counter_reset(page)`

Continues heading numbering from wherever the previous page left off. The numbering stays aligned with `\ref{}` links and updates when pages are reordered or headings are added or removed.

Set `project.extra.heading_numbering = false` in `zensical.toml` to turn numbering off entirely (content and sidebar) across the whole site. A page flagged `is_appendix: true` in its own front matter gets letter-based numbering instead - "Appendix A", "A.1", "A.1.1" - matching `prodockit.headings`' own `appendix_attr` default.

Contributors changing how page numbers are discovered should read Extension integration.

22.3.2 `reference_style()` / `acronym_style()` / `glossary_style()`

Controls list-entry spacing, driven by the same `project.extra.*` settings `prodockit.pdf` reads for the PDF, so both outputs stay in sync from one configured value:

Setting	Default	What it does
`reference_style`	`"european"`	`"european"`: single line spacing throughout, no indent, entries close together. `"global"`: single line spacing within each entry, double spacing *between* entries, with a hanging indent on wrapped lines (the common APA/MLA/Chicago style). Only `reference_style()` /the References page switches look - acronyms/glossary always use the tight "european" spacing.
`reference_spacing_european`	`"-0.8em"`	Gap between entries, "european" style - also used unconditionally for the acronym/glossary lists.
`reference_indent_global`	`"1.27cm"`	Hanging indent on wrapped lines, "global" style.
`reference_spacing_global`	`"2em"`	Gap between entries, "global" style.

For supported versions and the pre-1.0 stability boundary, see Support and compatibility.

23. Maintain prodockit

This section is for maintainers of the prodockit repository. It covers the package's own source, test matrix, documentation artifacts, pinned build inputs, GitHub Actions workflows, PyPI release, and public documentation deployment.

If you are writing and publishing a document with prodockit, use Publish a document instead. Machine setup, `prodockit-template`, template syncing, PDF generation, and Pages publication belong to that author workflow.

23.1 The maintenance cycle

The maintenance cycle below keeps repository identity, dependency versions, generated artifacts, and release automation under review.

1. Start from a clean default branch

   Update the branch GitHub publishes and confirm that no local work will be mixed into the maintenance change:

   ```
   git switch main
   git pull --ff-only
   git status --short
   ```

   Create a branch for the work rather than maintaining directly on `main`:

   ```
   git switch -c maintenance/update-build-inputs
   ```

2. Check repository identity

   Confirm that the website links, edit buttons, brand icon, and README badges still describe the remote this checkout uses:

   ```
   prodockit sync-repo --check
   ```

   If it reports drift, run `prodockit sync-repo`, review the diff, and repeat the check. See Repository metadata.

3. Check pinned build inputs

   First check that every file agrees without asking the network for newer releases:

   ```
   prodockit pins --check --offline
   ```

Use the scheduled drift workflow to decide whether to adopt a newer release. Do not upgrade merely to clear a notification: rebuild and compare the output first. See Version pinning and drift.

4. Build the deliverables locally

   Build in the same order as GitHub Actions. The PDF comes first because `zensical build` copies it into the published site:

   ```
   prodockit pdf
   zensical build --clean --strict
   ```

   Then run the checks appropriate to the repository. For prodockit itself:

   ```
   ruff check .
   mypy src
   prodockit pins --check --offline
   pytest
   ```

5. Review, push, and use the pull request gates

   Review both source and generated output. Commit the source changes, push the branch, and open a pull request:

   ```
   git diff --check
   git status --short
   git add <reviewed-files>
   git commit -m "Maintain project build inputs"
   git push -u origin HEAD
   gh pr create
   ```

   GitHub Actions runs the test matrix and strict documentation build on the pull request. Merge only after those required checks pass. Package releases have additional gates described in Build and release.

23.2 Choose the right maintenance tool

Need	Start with	What changes
A fork, mirror, or renamed repository points at the wrong place	`prodockit sync-repo --check`	Repository URLs, edit links, host icon, and managed README badges
CI files disagree about dependency versions	`prodockit pins --check --offline`	Version declarations, preserving each file's existing operator

Need	Start with	What changes
A newer dependency may change published output	GitHub's `drift.yml` result	Nothing automatically; it opens or updates an issue with the comparison
A prodockit release is ready	The release checklist	Package version, release notes, GitHub release, PyPI package, and rebuilt documentation

23.3 What automation does—and does not—prove

The workflows deliberately divide responsibility:

Workflow	Trigger	Answer
`ci.yml`	Pull requests and pushes to `main`	Does the code work across supported Python versions, and do the docs build strictly?
`docs.yml`	Pushes to `main` and manual dispatch	Can the complete website and PDF be built, tested, deployed, and verified live?
`drift.yml`	Weekly schedule and manual dispatch	Would newer rendering dependencies change the published artifacts?
`publish.yml`	Published GitHub release	Can the tagged source build and publish to PyPI through Trusted Publishing?
`release-redeploy.yml`	Published GitHub release	Can `docs.yml` be rerun against `main` so the site sees the new tag?

A green workflow proves the question in its own row, not every row. In particular, a passing pull request does not publish PyPI, and a successful Pages deployment does not by itself prove the public URL serves the new bytes. The deployment workflow performs that final delivery check separately.

23.4 Suggested cadence

When	Maintenance
Every pull request	Repository and pin consistency, tests, lint, typing, strict docs build
Weekly	Let `drift.yml` compare pinned and newest renderers; triage the issue it opens
After moving or forking a repository	Run `sync-repo`, rebuild, and inspect canonical/edit links and badges

When	Maintenance
Before a package release	Complete the local build gates, merge the release PR, publish a GitHub release, verify PyPI and Pages

The command-line map inventories the public CLI that a maintainer must keep consistent. The pages after it explain each repository maintenance job in depth.

24. Repository metadata

`prodockit sync-repo` keeps the repo-hosting-specific parts of your project in step with the git remote the checkout actually uses, so forking or mirroring it between GitHub, GitLab and Bitbucket doesn't leave stale links, the wrong brand icon, or README badges pointing at somebody else's repository.

Everything it writes is derived from one authority: the selected git remote, `origin` by default.

- In `zensical.toml`: `repo_url`, `repo_name`, `[project.theme.icon] repo`, `edit_uri`, and `site_url` where it can be known.
- In `README.md`: the badge row between the `repo-badges` markers, if you have them.

24.1 When to run it

Run `sync-repo` after:

- creating a repository from a template;
- forking or transferring a repository;
- renaming a repository or its default branch;
- changing `origin` from GitHub to GitLab, or the reverse;
- adding managed README badge markers;
- noticing that “Edit this page”, canonical URLs, or badges point elsewhere.

It is safe to include the check form in every pull request. The writing form is a maintenance action: run it on a branch and review the resulting metadata as you would any other source change.

24.2 Check, update, and verify

Run it from your project root, wherever `zensical.toml` lives:

1. Check without writing

   ```
   prodockit sync-repo --check
   ```

   Exit zero means the managed values already match the remote. A non-zero result lists what would change, making this form suitable for CI.

2. Update the managed values

   ```
   prodockit sync-repo
   ```

The command reports only what it actually changed:

```
Detected GitHub remote (https://github.com/you/your-repo);
updated: repo_url, repo_name, edit_uri
```

3. Review the source diff

   ```
   git diff -- zensical.toml README.md
   ```

   Confirm that the detected host, namespace, repository, default branch, and public site address are the ones you intend. A syntactically valid URL can still identify the wrong repository.

4. Repeat the check and build

   ```
   prodockit sync-repo --check
   zensical build --clean --strict
   ```

   Running the write command twice should do nothing the second time. The strict build then checks the links within the site; inspect the header repository link, an “Edit this page” link, the canonical URL, and README badges in the rendered or hosted result.

24.3 In CI

Place the non-writing check before the build. It catches a fork whose config still names its source repository before those stale links are published:

```
- name: Verify repository metadata
  run: prodockit sync-repo --check
- run: zensical build --clean --strict
```

Do not run the writing form in an ordinary pull-request job. CI would modify its disposable checkout, hide the source drift, and still leave the repository unchanged for the next run.

24.4 Options

Option	Default	What it does
`-f`, `--config-file`	`zensical.toml`	Which Zensical config to update.
`--readme`	`README.md`	README to update the badge block in. Pass an empty value to skip it.

Option	Default	What it does
`--remote`	`origin`	Which git remote to read the repository URL from.
`--branch`	detected	Default branch for `edit_uri` and GitLab build-badge links.
`--check`	off	Report what would change, write nothing, exit non-zero if anything would.

24.5 What it does, and why

24.5.1 `edit_uri`

Zensical falls back to `edit/master/<docs_dir>` when `edit_uri` isn't set - hardcoding the `master` branch name whatever your repository's default actually is, and only for an exact `github.com` / `gitlab.com` host match, so a self-hosted GitLab gets no default at all and its "edit this page" button simply never appears.

`sync-repo` sets it explicitly instead, from your remote's real default branch and matched by *kind* of host rather than exact hostname - so `gitlab.your-institution.ac.uk` is recognised as GitLab and gets a working edit link. A host it has no edit-URL convention for is left alone rather than guessed at.

24.5.2 `site_url`, and when it is left alone

`site_url` is the address your site is *published* at, which is not the address of the repository. Zensical puts it in every page's `<link rel="canonical">` and in every `sitemap.xml` entry, so a wrong one tells search engines the real home of your documentation is somewhere else - quietly, since the site builds and looks perfect either way.

Only GitHub Pages is derived, because only its shape is knowable from the remote: `https://<owner>.github.io/<repo>/`, or the bare origin when the repository is itself named `<owner>.github.io`.

GitLab is not guessed at. A self-hosted instance serves Pages from `pages_external_url`, an instance setting the remote URL says nothing about, and gitlab.com now gives new projects a unique domain with a random suffix rather than the old `<group>.gitlab.io/<project>` path. Set `pages_base` and the repository name is appended to it:

```
pages_base = "https://mb0105.pages.gitlab.surrey.ac.uk"
```

> **What it will not touch**
>
> An existing `site_url` is only replaced when it is already a Pages URL - one set up to follow the repository, so it keeps following it - or when it points at a code host, which is never a published site and is what the project template used to ship.
>
> Any other value is treated as a custom domain and left alone, with a note saying so. That matters because `--check` is wired into CI as a gate: rewriting a deliberate value would not merely lose it once, it would report drift on every run afterwards and redden builds for a config that was right all along.
>
> A `site_url` that is absent stays absent. It is optional in Zensical, and a project that left it out has no canonical URL by choice.

24.5.3 Nested GitLab groups

GitLab nests groups, so a project can live several levels down - `cs-dept/year3/report` is the `report` project in the `year3` subgroup of `cs-dept`. The whole path is kept, and every generated URL uses it: `repo_url`, the edit links, and the badges all address the real project rather than a `cs-dept/report` that does not exist.

The one exception is the `repo_name` label described below, which shows the immediate parent (`year3/report`). It is a caption, not a link - the header's target is `repo_url` - and printing a deep path verbatim would crowd the header for no gain.

GitHub has no equivalent nesting, so nothing changes there.

24.5.4 `repo_name` keeps the shape you chose

Zensical prints `repo_name` verbatim in the site header, and both `owner/repo` and a bare `repo` are in legitimate use. `sync-repo` looks at what your config already says and keeps that shape, updating only the values - so syncing never silently restyles your header.

24.5.5 README badges

If your README contains these markers, the block between them is replaced with a badge row (build status, stars, forks) pointing at whichever host your remote is on:

```
<!-- repo-badges:start -->
<!-- repo-badges:end -->
```

An empty pair, exactly as above, is the normal starting state: a template ships the markers and the first `sync-repo` fills them in.

GitHub and GitLab have known badge sets. Any other host is reported and left untouched rather than given invented URLs. If you don't want managed badges, leave the markers out - their absence is a normal state, not an error, and `sync-repo` just says so and moves on.

> **Self-hosted GitLab gets a build badge only**
>
> The badge set is chosen by host *kind*, and `gitlab` matches any instance - a university or company GitLab as readily as `gitlab.com`. The links are built from your actual host, so they point at your own instance rather than at a `gitlab.com` project that does not exist.
>
> The build badge is the one GitLab serves itself, at `<your-instance>/<owner>/<repo>/badges/<branch>/pipeline.svg`. That works on a private instance, where the reader is already authenticated against it - which an external badge service can never be.
>
> Stars and forks have no instance-served equivalent, and shields.io resolves its GitLab endpoints against `gitlab.com` only, so those two are emitted for `gitlab.com` alone. On a self-hosted instance they could render nothing but a broken image.

The Python API used by the command is documented for contributors under Development and code map.

25. Version pinning and drift

A documentation build has more inputs than its own source, which creates dependency drift when declared or installed versions diverge. `zensical` renders the site, `weasyprint` lays out the PDF, and both are ordinary Python packages that resolve to whatever is newest unless you say otherwise.

That matters more than it first appears, because the way an upgrade shows up here is **not** a failed build. It is a published document that quietly differs from the one you reviewed.

> **What this looks like in practice**
>
> Zensical 0.0.52 bumped its bundled Font Awesome from 7.2.0 to 7.3.1, which redrew the GitHub brand icon. Every page of this project's own website changed. The PDF was byte-identical. Nothing failed, nothing was committed, and the site simply looked slightly different the next time it deployed.
>
> A `weasyprint` upgrade is sharper still: it decides pagination, so a release that lays out one paragraph differently shifts every page number after it - and those page numbers are *content*, resolved into the back-of-book index and the table of contents.

The `prodockit pins` command supports two halves that only work together:

1. **Pin the build inputs**, so output changes when someone decides it should.
2. **Watch for newer releases**, so pinning does not mean going quietly stale.

25.1 Maintain a dependency safely

1. Check that the repository agrees with itself

   ```
   prodockit pins --check --offline
   ```

   This does not contact PyPI. It answers the pull-request question: do the version declarations already present in `pyproject.toml` and the workflows agree? It is the form used by `ci.yml`, because a new upstream release should not make an unrelated pull request fail.

2. Review drift information

   The scheduled `drift.yml` workflow asks the separate maintenance question: are newer releases available, and would they change the website or

PDF? Read the issue it opens or run the workflow manually before selecting versions.

For a quick inventory from a terminal:

```
prodockit pins
```

This contacts PyPI and offers the newest version, but it does not compare rendered artifacts for you.

3. Move every declaration together

   Set an reviewed version explicitly:

   ```
   prodockit pins --set zensical=0.0.55
   ```

   The tool preserves the role of each declaration: a library floor remains a floor and a publishing workflow's exact pin remains exact.

4. Rebuild in publishing order

   ```
   prodockit pdf
   zensical build --clean --strict
   pytest
   ```

   Compare the website and PDF with the pinned baseline recorded by the drift issue. Review pagination, generated indexes, diagrams, code blocks, and icons—not only whether the commands returned zero.

5. Confirm consistency before committing

   ```
   prodockit pins --check --offline
   git diff --check
   git status --short
   ```

   The final offline check prevents a partial version bump from reaching the pull request. The pull request then runs the same consistency gate in `ci.yml`.

25.2 Where a version gets declared

Pinning creates its own problem: the same version ends up written in several files at once, and nothing keeps them in step.

File	Typically declares	Why that form
`pyproject.toml`	`zensical⩾0.0.55`	A **floor**. An exact pin in a library's metadata propagates to every consumer and conflicts with any project needing a different Zensical.
CI docs/build job	`zensical==0.0.55`	An **exact pin**. The site and PDF are artifacts; they should change deliberately.
CI test job	`weasyprint==69.0`	An **exact pin**. Tests that assert on where things land in a rendered PDF treat the layout engine as an input, not an implementation detail.
Drift job	both, exactly	The baseline it compares the newest release against.

Both forms are correct in their own place. What is not correct is them disagreeing, which is easy to do by hand and invisible when it happens.

25.3 `prodockit pins`

Reads every declaration across all of those files, shows what is currently set against what is newest on PyPI, and moves them together.

```
prodockit pins
```

```
zensical
  pyproject.toml:34  zensical⩾0.0.53
  .github/workflows/docs.yml:164  zensical==0.0.53
  .github/workflows/drift.yml:69  zensical==0.0.53
  newest on PyPI: 0.0.53  ← newer available

weasyprint
  .github/workflows/ci.yml:59  weasyprint==69.0
  .github/workflows/docs.yml:164  weasyprint==69.0
  newest on PyPI: 69.0

markdown
  pyproject.toml:25  markdown⩾3.10.3
  .github/workflows/docs.yml:164  markdown==3.10.3
  .github/workflows/drift.yml:69  markdown==3.10.3
  newest on PyPI: 3.10.3

pymdown-extensions
  .github/workflows/docs.yml:164  pymdown-extensions==11.0.1
  .github/workflows/drift.yml:69  pymdown-extensions==11.0.1
  newest on PyPI: 11.0.1
```

```
zensical: version to set [0.0.53]:
```

`Markdown` and `pymdown-extensions` are in that list even though nothing installs them directly - they arrive under Zensical, which declares only floors for them. See the limitations below for why pinning Zensical alone left them free to move.

Press ++enter++ to take the newest release, or type a version. Each site keeps **its own operator** - the floor stays a floor, the pins stay pinned - so one answer updates every file correctly.

25.3.1 Options

Option	What it does
`-r`, `--root`	Project root to scan. Defaults to the current directory.
`-p`, `--package`	Package to manage, repeatable. Defaults to `zensical`, `weasyprint`, `Markdown`, `pymdown-extensions` and `pandoc`.
`--set PACKAGE=VERSION`	Set a version without prompting, repeatable. Implies `--no-input`.
`--latest`	Take PyPI's newest for every package without prompting. Implies `--no-input`.
`--no-input`	Never prompt. Packages given a version are updated; the rest are reported and left untouched.
`--check`	Report and exit non-zero if anything is behind or inconsistent. Writes nothing.
`--offline`	Skip the PyPI lookup and only report what the files declare.

`--set` is the unattended form, so it suppresses the prompt for **every** package, not only the one it names:

```
prodockit pins --set zensical=0.0.53
```

```
  pyproject.toml:34  zensical⩾0.0.52 → zensical⩾0.0.53
  .github/workflows/docs.yml:136  zensical==0.0.52 →
zensical==0.0.53

Left untouched (no version given): weasyprint

Updated 2 declaration(s). Rebuild and diff before committing.
```

A package it was not given is reported and its files are not opened - name it with its own `--set` to move it, or leave it where it is. That is what makes the command safe in a script: it can neither hang on a prompt nor half-finish because one appeared.

`--check` is the one to put in CI:

```
prodockit pins --check --offline
```

It fails when a package is behind PyPI **or** when the files disagree with each other - the second being the failure that pinning across several files invites.

> **Add `--offline` when it gates a pull request**
>
> Those two failures belong in different places. *Files disagreeing* is a property of the repository: a real mistake, introduced by a commit, fixable by its author. *Behind PyPI* is a property of the world, and turns every open pull request red the day upstream ships a release, with nothing in the branch having changed and nothing the author can do about it. A gate that fails for reasons outside the contributor's control is one people learn to ignore.
>
> `--offline` keeps the first check and drops the second, and needs no network. Leave "is there something newer" to a drift job, which reports on a schedule rather than failing a build. This project's own `ci.yml` runs the offline form for exactly this reason.

25.3.2 What it scans

Both CI hosts, so the same command works either way:

- `pyproject.toml`, `setup.cfg`
- `.github/workflows/*.yml` — GitHub Actions
- `.gitlab-ci.yml` and `.gitlab/**/*.yml` — GitLab CI
- `requirements*.txt`, `constraints*.txt` at the project root

Build output and virtualenvs (`site/`, `public/`, `.venv/`, `node_modules/`, ...) are skipped, so a stale copy of a workflow inside one is not mistaken for a declaration.

Four shapes of declaration are recognised, because a build input is not always a pip package:

Shape	Example	Where
pip specifier	`zensical==0.0.52`, `zensical>=0.0.52`	anywhere

Shape	Example	Where
runner label	`runs-on: ubuntu-24.04`	GitHub Actions
image tag	`image: python:3.13`	GitLab CI, or any container
CI variable	`PANDOC_VERSION: "3.10.1"`	a GitHub `env:` block, a GitLab `variables:` block

Why prodockit is managed by default

It was not, for a long time, and the omission had exactly the consequence the managed set exists to prevent. `prodockit-template` pinned `prodockit==0.35.0` and drifted two releases behind with nothing noticing, because the one command that looks at pins was not looking at this one - moving it needed `-p prodockit` typed by hand, which is the step nobody remembers (prodockit-template#173).

It belongs there on the merits too: prodockit renders the PDF and generates the back-of-book index, so its version changes a project's published output as directly as Zensical's does.

Including it is safe even in prodockit's own repository, where the name appears in `pyproject.toml` as the project's identity rather than as a dependency. The specifier pattern requires a version operator after the name, so `name = "prodockit"` and the adjacent `version = "..."` are not declarations - without that, the command would offer to rewrite the release number of the package being built.

Why pandoc is managed by default

Pandoc is not a Python package, so it never appears as a pip specifier - it is a build-provided binary, pinned as a `<PACKAGE>_VERSION` variable the way `prodockit pdf`'s publishing workflow does. It earns a place in the default set for the same reason Zensical and WeasyPrint do: pandoc is not always compatible with itself across releases, and one of its changes broke every fenced code block in this project's own PDF while the build kept reporting success.

The CI variable's name keeps its case on rewrite - `PANDOC_VERSION`, not `pandoc_VERSION` - since a workflow step reading `${{ env.PANDOC_VERSION }}` needs the name unchanged, only the value.

25.3.3 Pandoc version drift

Distribution Pandoc packages can lag several major versions behind upstream. Ubuntu 24.04, for example, supplies an older release than the one this repository currently tests and publishes with. Installing the distribution package locally while CI downloads a current release means the two builds can parse identical HTML differently.

That happened here when one Pandoc release accepted highlighted `<pre><code>` content that a newer release interpreted differently. Every fenced code block reflowed as ordinary justified prose in one environment, while the other environment continued publishing a correct PDF. Both builds reported success.

Pin Pandoc as a `PANDOC_VERSION` workflow variable and move it through `prodockit pins`, then compare complete PDF and website artifacts before and after the change. Pinning does not prevent incompatibility; it makes the change arrive in a reviewed commit instead of with an unannounced runner update.

> **Only versioned declarations are found**
>
> A dependency installed with no version at all is not a declaration site, so it will not appear - pin it once by hand and the tool manages it from then on. The same applies to `runs-on: ubuntu-latest`, which names no version to move.

25.4 Watching for drift

Pinning trades one risk for another: nothing tells you a newer release exists, or what it would do to your output.

The check worth running is not "is there a new version" - PyPI can answer that - but **"would taking it change what we publish?"** That needs a real build, twice.

The shape is the same on either host:

1. Build the docs with the pinned versions. Keep the result.
2. Upgrade to the newest and build again.
3. Diff the two, byte for byte.
4. Run your built-output checks against the *newer* build.
5. Report - do not fail.

Both builds run in the same job, so pandoc, Chrome, fonts and the runner image are identical between them and any difference is attributable to the upgraded packages alone.

> **Two things make or break this**

> **Build order.** `zensical build` copies the PDF into the site directory, so it must run *after* `prodockit pdf`. Reversed, the copied PDF lags a generation and every comparison is a false positive that looks exactly like nondeterminism.
>
> **Determinism.** The diff only means something if identical inputs give identical output. Verify that first - two builds of the same version should produce a byte-identical site and PDF. They do for this project; confirm it for yours before trusting a diff.

25.4.1 Reporting, not failing

A newer release is information, not a broken build. A scheduled job that goes red every week trains everyone to ignore it, so open an issue instead - and keep **one** open at a time, updating it in place rather than filing a fresh one every Monday until somebody acts.

25.4.2 GitHub Actions

This project ships `drift.yml` doing exactly this. The essentials:

```
name: Dependency drift
on:
  schedule:
    - cron: "0 6 * * 1"   # Mondays, 06:00 UTC
  workflow_dispatch:
permissions:
  contents: read
  issues: write           # needed to open the issue
jobs:
  drift:
    runs-on: ubuntu-latest
    steps:
      # ... same build tooling as your docs job ...
      - name: Build with the pinned versions
        run: |
          pip install -e ".[testing]" "zensical==0.0.55"
"weasyprint==69.0" "Markdown==3.10.3" "pymdown-extensions==11.0.1"
          prodockit pdf                      # PDF first ...
          zensical build --clean --strict    # ... then the site
          cp -R site /tmp/pinned-site
          cp docs/site_documentation.pdf /tmp/pinned.pdf

      - name: Build with the newest versions
        run: |
```

```
          pip install -qU zensical weasyprint
          prodockit pdf
          zensical build --clean --strict

      - name: Compare
        env:
          GH_TOKEN: ${{ github.token }}
        run: |
          cmp -s /tmp/pinned.pdf docs/site_documentation.pdf \
            && echo "PDF identical" || echo "PDF differs"
          diff -rq /tmp/pinned-site site | wc -l
          # ... then gh issue create / gh issue comment
```

25.4.3 GitLab CI

The same job, with GitLab's own scheduling and API. Add a pipeline schedule running weekly, and a project access token with the `api` scope exposed as `DRIFT_TOKEN` so the job can open an issue.

```
drift:
  image: python:3.13
  rules:
    # Only on the schedule - never on a normal push pipeline.
    - if: $CI_PIPELINE_SOURCE == "schedule"
  before_script:
    - apt-get update && apt-get install -y pandoc libpango-1.0-0
libpangoft2-1.0-0 libharfbuzz-subset0 jq curl
  script:
    - pip install -e ".[testing]" "zensical==0.0.55"
"weasyprint==69.0" "Markdown==3.10.3" "pymdown-extensions==11.0.1"
    - prodockit pdf                    # PDF first ...
    - zensical build --clean --strict   # ... then the site
    - cp -R site /tmp/pinned-site && cp docs/
site_documentation.pdf /tmp/pinned.pdf
    - pip install -qU zensical weasyprint
    - PINNED=$(pip show zensical | awk '/^Version:/{print $2}')
    - prodockit pdf
    - zensical build --clean --strict
    - LATEST=$(pip show zensical | awk '/^Version:/{print $2}')
    - |
      if [ "$PINNED" = "$LATEST" ]; then
        echo "Pins are current."; exit 0
      fi
      CHANGED=$(diff -rq /tmp/pinned-site site | wc -l)
      cmp -s /tmp/pinned.pdf docs/site_documentation.pdf &&
PDF=identical || PDF=differs
      # One open issue at a time.
      EXISTING=$(curl -sf --header "PRIVATE-TOKEN: $DRIFT_TOKEN" \
```

```
        "$CI_API_V4_URL/projects/$CI_PROJECT_ID/issues?
state=opened&search=Dependency+drift" \
        | jq -r '.[0].iid // empty')
      BODY="zensical $PINNED → $LATEST. PDF: $PDF. Website:
$CHANGED files differ."
      if [ -n "$EXISTING" ]; then
        curl -sf --request POST --header "PRIVATE-TOKEN:
$DRIFT_TOKEN" \
          --data-urlencode "body=$BODY" \
          "$CI_API_V4_URL/projects/$CI_PROJECT_ID/issues/$EXISTING/
notes" > /dev/null
      else
        curl -sf --request POST --header "PRIVATE-TOKEN:
$DRIFT_TOKEN" \
          --data-urlencode "title=Dependency drift: newer build
inputs than the docs pins" \
          --data-urlencode "description=$BODY" \
          "$CI_API_V4_URL/projects/$CI_PROJECT_ID/issues" > /dev/
null
      fi
  allow_failure: true
```

`allow_failure: true` keeps the pipeline green: the job's purpose is the issue it opens, not its own exit status.

25.5 The input pip cannot reach

`prodockit pins` manages Python packages. The CI runner image is the other half, and nothing in `pyproject.toml` or a workflow's `pip install` line touches it:

- **`pandoc`** comes from the image's own package archive. Distribution packages lag upstream far enough that some Markdown edge cases parse differently—see Pandoc version drift.
- **Fonts** the PDF embeds (`fonts-inter`, `fonts-jetbrains-mono`).
- **Chrome**, which rasterises Mermaid diagrams.

On `runs-on: ubuntu-latest` all three move the day the label migrates to a new LTS, with nothing committed - the same silent change the package pins exist to prevent, one layer down. Naming the image freezes them together:

```
jobs:
  deploy:
    runs-on: ubuntu-24.04   # not ubuntu-latest
```

GitLab's equivalent is the job `image:`, which most projects already pin by habit - `python:3.13` rather than `python:latest`.

`prodockit pins` manages both, so the image is inventoried and moved the same way as everything else - name it with `-p`:

```
prodockit pins -p ubuntu       # runs-on: ubuntu-24.04, across every
workflow
prodockit pins -p python       # image: python:3.13, in GitLab CI
```

```
ubuntu
  .github/workflows/ci.yml:11  ubuntu-24.04
  .github/workflows/docs.yml:69  ubuntu-24.04
  .github/workflows/drift.yml:36  ubuntu-24.04
  .github/workflows/publish.yml:10  ubuntu-24.04
  not on PyPI - set the version yourself

ubuntu: version to set [24.04]:
```

There is no suggested version for these: PyPI has nothing to say about a runner image, and asking it for "ubuntu" would at best miss and at worst find an unrelated package of that name and propose a nonsense upgrade. The default is what is currently set, so ++enter++ is a no-op and you type the new one deliberately - which suits a change you make once every couple of years.

`--check` still applies, and catches the failure that matters here: some jobs left on the old image after a partial migration.

> **What this costs**
>
> `ubuntu-latest` already *is* 24.04 today, so pinning changes nothing immediately. It takes effect at the migration - which is the point.
>
> Pinned images are retired roughly a year after the following LTS, and the job then fails outright rather than drifting. That is the better failure: loud, and at a time you choose. Treat a retirement notice as the prompt to rebuild, diff, and move up deliberately.

25.6 Taking an upgrade

When drift reports something worth having, use the complete maintenance flow above. The short command sequence is:

```
prodockit pins --set zensical=0.0.55
prodockit pdf
zensical build --clean --strict
```

```
pytest
prodockit pins --check --offline
```

Substitute the package and reviewed version reported by drift. Then diff the built output against the previous version before committing. That is the step the whole arrangement exists to make possible: seeing what an upgrade does to your document *before* your readers do.

25.7 Limitations and workarounds

See Implementation limitations for the general list. Specific to pinning:

- **A floor still floats.** `zensical ⩾ 0.0.55` in `pyproject.toml` records a version; it does not control one. Only the exact pin in the build job does. Both exist deliberately - see the table above.
- **A pinned package's own dependencies float too**, which is the sharper version of the same trap: pinning a direct dependency exactly does nothing for the transitive ones underneath it, because *their* versions come from floors in *its* metadata. Zensical is pinned exactly here, and still declares only floors for `Markdown` and `pymdown-extensions` - the two packages that actually turn every page into HTML. A build pinning Zensical alone therefore rendered with whatever those two resolved to on the morning it ran. Both are now pinned alongside it, and both are managed by `prodockit pins` (#178). If your project pins something whose rendering you depend on, check what it pulls in: `pip show <package>` lists its requirements, and a floor there is a version you are not controlling.
- **Only pip packages are watched.** `pandoc` and Chrome arrive from the runner image, so a drift job that installs both builds in one job cannot see them change between weeks. Pinning the image is the lever for those - see below.
- **Comments are not scanned.** A version specifier written in prose - explaining why something is pinned, say - is deliberately not treated as a declaration, so it is neither reported nor rewritten by `--set`. Only the part of a line before a `#` is read (#184); a trailing comment after a real declaration still leaves that declaration findable.
- **`prodockit pins` needs network** for `--latest` and the suggested default. Use `--offline` to report what the files declare without asking PyPI.

26. Build and release

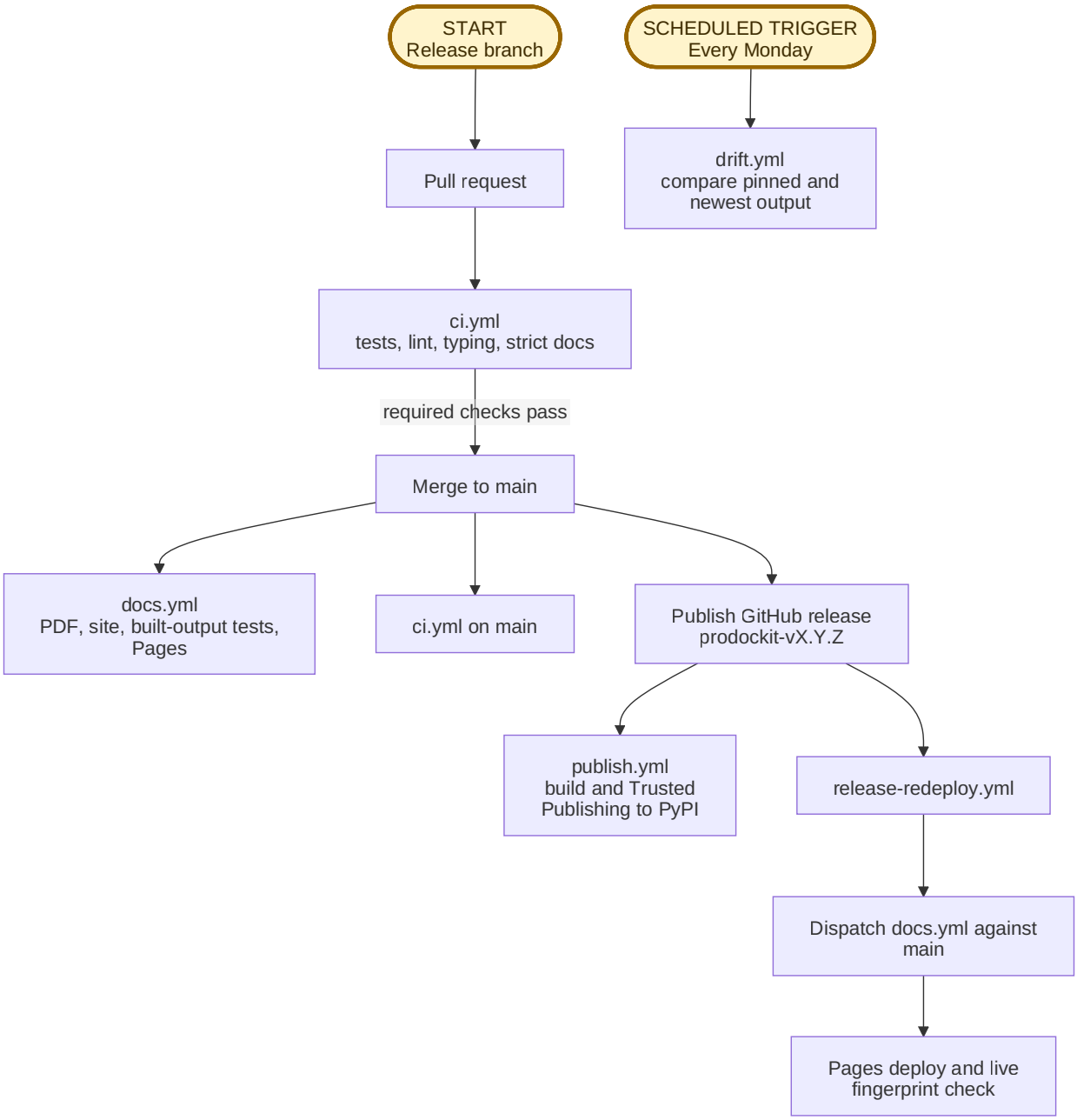
This page documents the release process for taking prodockit from an accepted change to a package on PyPI and a documentation site that shows the same release. It describes this repository's real GitHub Actions workflows rather than a generic Python release.

A release is deliberately split in two:

1. A pull request changes and validates the source.
2. A published GitHub release creates the tag and authorises publishing.

Do not publish a GitHub release from an unmerged branch. `publish.yml` builds the tagged source, while the documentation redeploy is deliberately run from `main`; both must describe the same commit.

26.1 Understand the workflow chain



The rounded gold boxes are entry points: a maintainer starts the release path from a release branch, while GitHub starts the drift path on its weekly schedule. Rectangular boxes are actions or workflow stages that follow.

Workflow	Trigger	Responsibility
`ci.yml`	Pull requests; pushes to `main`	Test Python 3.10–3.13, lint, type-check, verify pins, run the suite, and strictly build the site

Workflow	Trigger	Responsibility
`docs.yml`	Pushes to `main`; manual dispatch	Build the complete PDF and selected single-page PDFs, strictly build the website, run built-output tests, deploy Pages, then verify the live page matches the uploaded artifact
`drift.yml`	Monday schedule; manual dispatch	Build with pinned and newest rendering dependencies, compare artifacts, run checks against the newer build, and open or update an issue rather than failing for mere availability
`publish.yml`	Published GitHub release	Build source and wheel artifacts from the release tag, then publish them to PyPI through Trusted Publishing
`release-redeploy.yml`	Published GitHub release; manual dispatch	Start `docs.yml` against `main` after the new tag exists, so the cover and macros can show the new release without deploying from a tag ref

The five workflows overlap intentionally. `ci.yml` gives quick pull-request feedback; `docs.yml` proves and publishes the complete artifacts; `publish.yml` has the narrow permission needed for PyPI; the redeploy fixes release-tag timing; and `drift.yml` observes future upgrades without changing the current release.

26.2 1. Choose the release version

Use semantic versioning as a decision aid:

Change	Version
Backward-compatible fixes or documentation corrections	Patch
New backward-compatible commands, options, or extension features	Minor
An intentional incompatible public change	Major—or the repository's agreed pre-1.0 policy

Read every change since the previous `prodockit-v...` tag, not only the pull requests carrying a changelog entry:

```
git fetch --tags origin
git log --oneline prodockit-v0.40.0..origin/main
```

The tag prefix matters. Historic tags named only `vX.Y.Z` exist, but current package releases use `prodockit-vX.Y.Z`, such as `prodockit-v0.40.0`.

26.3 2. Prepare the release branch

1. Start from current main

   ```
   git switch main
   git pull --ff-only
   git switch -c release/0.41.0
   ```

   Replace `0.41.0` throughout this page with the version being prepared.

2. Update both code version sources

   The package version is declared twice:

   pyproject.toml

   ```
   [project]
   version = "0.41.0"
   ```

   src/prodockit/__init__.py

   ```
   __version__ = "0.41.0"
   ```

   They serve different readers: build metadata supplies the wheel and PyPI; `prodockit.__version__` supplies Python callers and `prodockit --version`. Leaving either behind publishes two answers about one release.

3. Finish the release notes

   In `docs/about/changelog.md`, replace the one `## Unreleased` heading with:

   ```
   ## 0.41.0 (2026-08-20)
   ```

   Review every merged change since the previous tag. Add missing entries and write for a user deciding whether to upgrade: what changed, why it matters, and any action required. Do not paste commit subjects without context.

   The changelog tests enforce one Unreleased section at most, its position, and newest-first released versions. They do not know whether a human-readable change was omitted, so the comparison with `git log` is still required.

4. Check descriptions exposed outside the guide

If the release changes the public capability set, update the places a reader sees before opening the full documentation:

- `README.md`, rendered on GitHub and PyPI;
- the `[project].description` in `pyproject.toml`, used as PyPI's summary;
- the package docstring in `src/prodockit/__init__.py`, shown by Python help.

Tests guard extension inventories, but prose describing a changed command or capability still needs review.

26.4 3. Run the local release gates

Activate the development environment first. On Apple Silicon macOS, expose Homebrew's Pango libraries in the same terminal that will run the gates:

```
export DYLD_FALLBACK_LIBRARY_PATH=/opt/homebrew/lib
```

Use `/usr/local/lib` on an Intel Mac. If this is missing, the PDF-backed tests typically fail with `cannot load library 'libgobject-2.0-0'` even though `brew install pango` has completed. This is a loader-path problem, not evidence of a code regression or a missing Python package.

Run the same logical gates as `ci.yml`:

```
ruff check .
mypy src
prodockit pins --check --offline
pytest
zensical build --clean --strict
```

Because this repository publishes a PDF as part of its documentation, also build it before the strict website build:

```
prodockit pdf
zensical build --clean --strict
python -m pytest tests/test_built_docs.py -m built -v
```

Then verify the release identity directly:

```
python -m build
python -m zipfile --list dist/prodockit-0.41.0-py3-none-any.whl
prodockit --version
```

The wheel filename and command output should both say `0.41.0`. Do not upload

the locally built `dist/`; `publish.yml` rebuilds from the immutable release tag and publishes that artifact.

26.5 4. Open and merge the release pull request

Review the release diff before committing:

```
git diff --check
git status --short
git diff -- pyproject.toml src/prodockit/__init__.py docs/about/
changelog.md
```

Commit, push, and open the pull request:

```
git add pyproject.toml src/prodockit/__init__.py docs/about/
changelog.md
git commit -m "Release 0.41.0"
git push -u origin release/0.41.0
gh pr create --title "Release 0.41.0"
```

Only add files actually changed and reviewed; the command above is a checklist, not permission to commit unrelated work.

On the pull request, `ci.yml` runs:

- the supported-Python matrix with the real Pandoc and WeasyPrint stack;
- Ruff and mypy;
- `prodockit pins --check --offline`;
- the full ordinary test suite;
- a separate strict documentation build.

Merge only after all required checks pass. After merging, wait for the new `main` run of `ci.yml` and the first `docs.yml` deployment to succeed. That first documentation build may still show the previous release because the new tag does not exist yet; the release-triggered redeploy corrects it.

26.6 5. Publish the GitHub release

Create a release targeting the merged `main` commit. Publishing—not merely saving a draft—is the event that starts PyPI publishing and the documentation redeploy:

```
gh release create prodockit-v0.41.0 \
  --repo buckwem/prodockit-extensions \
  --target main \
```

```
  --title "prodockit 0.41.0" \
  --generate-notes
```

Before confirming, check:

- the tag is exactly `prodockit-v0.41.0`;
- the target is the merged release commit on `main`;
- the release title is `prodockit 0.41.0`;
- the notes describe the same release as `docs/about/changelog.md`.

A tag with the right name on the wrong commit is still the wrong package. Do not delete and recreate tags casually once an artifact may have reached PyPI; package versions there are immutable.

26.7 6. Follow the release workflows

Two workflows start from the published release.

26.7.1 PyPI: `publish.yml`

The `build` job checks out the tag, installs `build`, runs `python -m build`, and uploads `dist/` as a workflow artifact. The `publish` job downloads that exact artifact in the protected `pypi` environment and uses PyPI Trusted Publishing (`id-token: write`), so no long-lived API token is stored in the repository.

Watch it with:

```
gh run list --workflow publish.yml --limit 5
```

After success, verify the public package rather than relying only on the green workflow:

```
python -m pip index versions prodockit
```

26.7.2 Documentation: `release-redeploy.yml` → `docs.yml`

The release event itself runs against a tag ref. GitHub Pages deployments from that ref previously reported success while the public site kept serving the old build. `release-redeploy.yml` therefore builds nothing: it dispatches `docs.yml --ref main` with `actions: write` permission.

`docs.yml` then:

1. checks out full history so `git describe` can see tags;
2. installs pinned system, Python, Mermaid, and MathJax inputs;
3. builds the complete PDF and selected single-page PDFs;
4. strictly builds the website after the PDF;

5. runs the built-output tests;
6. uploads and deploys `site/`;
7. fingerprints the uploaded `index.html` and polls the public Pages URL until it serves those exact bytes.

The workflow uses a `pages` concurrency group with cancellation disabled. A queued later deployment must wait and supersede the earlier one; cancelling it could leave the pre-release-tag build live.

Watch both workflows:

```
gh run list --workflow release-redeploy.yml --limit 5
gh run list --workflow docs.yml --limit 5
```

26.8 7. Verify the release as a user

Automation has separate success conditions, so perform separate public checks:

Check	What it proves
GitHub release page shows `prodockit-v0.41.0`	The release and tag are public
PyPI lists `0.41.0` and both wheel/source files	Trusted Publishing completed
A clean environment installs `prodockit==0.41.0`	Package metadata and dependencies resolve for a user
`prodockit --version` prints `0.41.0`	The installed code agrees with package metadata
Documentation cover shows the new release	The post-release main-branch redeploy completed
`docs.yml` verify job passes	The public Pages URL serves the artifact built by that run

A clean installation check can use a temporary virtual environment:

```
python -m venv /tmp/prodockit-release-check
/tmp/prodockit-release-check/bin/python -m pip install
"prodockit==0.41.0"
/tmp/prodockit-release-check/bin/prodockit --version
```

Use the platform's corresponding activation or executable path on Windows.

26.9 Recover from a failed stage

Failure	Resume from
Pull-request CI fails	Fix the release branch; do not publish the release
`publish.yml` build job fails	Fix through a new commit and release version; the tag is the source of truth
PyPI publish job fails before upload	Correct the environment/Trusted Publishing problem and rerun the failed job
PyPI already contains the version	Never overwrite it; determine whether the existing files are correct and release a new version if code must change
`release-redeploy.yml` fails	Manually run `gh workflow run docs.yml --ref main` after fixing permissions or workflow availability
Pages deploy succeeds but verify fails	Inspect the response headers and rerun `docs.yml`; do not assume successful upload means successful delivery
Drift issue opens after release	Triage it as future maintenance; it does not invalidate the pinned release that just shipped

26.10 After release

Return to ordinary development by creating a new `## Unreleased` section when the next user-visible change is made. Do not create an empty section merely as release ceremony; the changelog test permits it to be absent between releases.

Downstream repositories that pin prodockit—especially `prodockit-template` and the userguide—should update deliberately, rebuild their own site and PDF, and use their own tests before adopting the release.

27. Contributor internals

These contributor internals are for developers contributing code to prodockit or reviewing a change that touches its internal design. It is not required reading for a document author or for a maintainer following the operational release runbook.

The beginner and authoring sections explain how to write with prodockit. Publish a document is for a document author producing and deploying a website and PDF. Maintain prodockit is for maintainers of the prodockit repository reviewing its automation, pins, tests, and package releases.

These contributor notes explain the implementation behind both audiences: which Zensical internals the package relies on and which limitations are deliberate rather than undocumented features.

They are grouped together because they share a failure mode. Almost nothing here breaks loudly. The PDF pipeline shells out to external binaries, reads undocumented Zensical internals, and resolves cross-page references from a registry built during the render - and when any of that goes wrong, the usual result is a build that succeeds and publishes something subtly wrong: a diagram that reached the PDF as raw source, a reference that resolved to `??`, a deploy that reported success and was never served. A green build is weak evidence, which is why these pages spend as much time on what to check as on what to configure.

Read these after the public task guides when you need to change the package itself:

1. **Development and code map** - install an editable checkout, run the contributor checks, and find the code responsible for a feature.
2. **Extension integration** - shared registries, cross-page pre-scans, bibliography delegation, index passes, and block output contracts.
3. **PDF pipeline and API** - the render pipeline, Python entry points, internal modules, and error boundaries.
4. **Bootstrap design** - the check, plan, apply, recheck model and the ordering constraints behind machine setup.
5. **Zensical coupling** - undocumented Zensical APIs prodockit depends on. Read it before taking a Zensical upgrade.
6. **Implementation limitations** - known platform, extension, PDF, and macro constraints and the reason for each workaround.

28. Development and code map

This page is for contributors changing prodockit's Python package, tests, or documentation. Installing prodockit from PyPI is covered under Get started; this editable installation creates a development environment that keeps the checkout connected to the environment.

28.1 Create a development environment

```
git clone https://github.com/buckwem/prodockit-extensions
cd prodockit-extensions
python -m venv .venv
source .venv/bin/activate
python -m pip install -e ".[dev]"
```

On Windows, activate with `.venv\Scripts\Activate.ps1`. The editable install provides `prodockit`, `pdk`, and Zensical while importing package code directly from `src/`.

> **On macOS, expose Homebrew's Pango libraries**
>
> WeasyPrint's Python package still needs the native libraries installed by `brew install pango`. On Apple Silicon, export `DYLD_FALLBACK_LIBRARY_PATH` in the same terminal before running the PDF-backed tests:
>
> ```
> export DYLD_FALLBACK_LIBRARY_PATH=/opt/homebrew/lib
> ```
>
> Without it, tests that import or invoke WeasyPrint fail with `cannot load library 'libgobject-2.0-0'`, even when Pango is installed. On an Intel Mac, use `/usr/local/lib` instead.

Run the ordinary contributor gates before opening a pull request:

```
ruff check .
mypy src
prodockit pins --check --offline
pytest
zensical build --clean --strict
```

28.2 Find the code

This source code map shows where each public feature is implemented.

- src
  - prodockit
    - cli.py — public commands and aliases
    - settings.py — extension configuration helpers
    - headings.py — heading ids and numbering
    - refs.py — heading, figure, and table references
    - citations.py — Markdown-defined citations
    - glossary.py — acronyms and glossary terms
    - bibliography.py — Pandoc citeproc integration
    - tables.py — table layout attributes
    - tree.py — directory-tree block
    - steps.py — numbered-steps block
    - index.py — inline index markers
    - pdf — PDF build and source-bundle pipeline
    - bootstrap — machine setup stages and host model
    - testing — pytest plugin, fixtures, and output checks
- tests — unit, integration, documentation, and built-output tests
- docs — public guides and contributor internals
- tools — Mermaid and MathJax tooling used by PDF builds
- pyproject.toml — package metadata, dependencies, and extension entry points
- zensical.toml — this documentation site's configuration

Each Markdown extension is registered in `pyproject.toml`. A new public extension normally needs its module, entry point, tests, Authoring reference page, navigation entry, README inventory entry, and release note.

28.3 Call maintenance logic from Python

CLI commands should remain thin wrappers around functions that return useful state. For example, `prodockit sync-repo` calls `sync_repo_metadata()`:

```
from prodockit.sync_repo import sync_repo_metadata

result = sync_repo_metadata(check=True)
if result.changed:
    print("out of date:", ", ".join(result.changes))
```

The function returns changes and notes instead of printing them, which keeps it usable from tests and other tooling. The CLI owns terminal formatting and exit status.

28.4 Keep the pytest plugin lightweight

The `prodockit.testing` pytest plugin is discovered in every environment where prodockit is installed, including unrelated test suites. It therefore avoids heavy imports at module import time. PyMuPDF, Beautiful Soup, and Zensical are loaded only inside fixtures that need them, and a missing `zensical.toml` affects the requested fixture rather than test collection.

When adding a fixture, preserve session scope where possible, prefix its name with `prodockit_`, resolve paths from pytest's root directory, and avoid work until a test requests it.

29. Extension integration

The Authoring reference documents Markdown syntax and `zensical.toml` options. This page explains the integration machinery used when changing an extension or embedding it in a renderer other than Zensical.

29.1 Share definitions across pages

Headings, references, citations, and glossary terms need build-wide state:

Extension	Shared object	Registered content
`prodockit.headings` and `prodockit.refs`	`IdRegistry`	Heading, figure, and table ids, labels, and destinations
`prodockit.citations`	`CitationRegistry`	Citation keys and authored reference text
`prodockit.glossary`	`GlossaryRegistry`	Term ids, display text, and definitions

Zensical creates a fresh Python-Markdown instance per page. Prodockit detects the Zensical page context, derives a `source` path, and shares the appropriate registry across the build. A pre-scan reads definitions from every navigation page before conversion so a page can refer forward to a definition rendered later.

Outside Zensical, the caller supplies the registry and source explicitly:

```
import markdown
from prodockit.headings import HeadingsExtension
from prodockit.refs import RefsExtension
from prodockit.util import IdRegistry

registry = IdRegistry()

for path, text in pages:
    html = markdown.markdown(
        text,
        extensions=[
            HeadingsExtension(registry=registry, source=path),
            RefsExtension(registry=registry, source=path),
        ],
    )
```

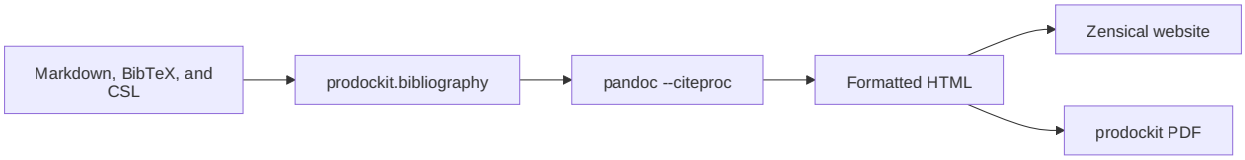
The equivalent constructors accept `CitationRegistry` and `GlossaryRegistry`.

A manually shared registry raises `DuplicateIdError` for a collision between different sources. Zensical's best-effort automatic integration warns and keeps the first registration, so public guides require explicit ids for repeated headings such as “Overview”.

`prodockit.headings.prescan()` exposes the same continuous-numbering scan to template macros and other build tooling. It returns page-keyed starting counts and appendix letters while a Zensical build is active.

29.2 Delegate bibliography formatting

`prodockit.bibliography` does not implement CSL sorting, localisation, or disambiguation. It passes each distinct citation or reference-list request to `pandoc --citeproc` with the configured `.bib` and `.csl` files, then memoizes the formatted HTML for the rest of the build.



Pandoc formats only the requested citation or list. Zensical still renders the surrounding page.

29.3 Generate an index after layout

Index markers exist before pagination, but their page numbers do not. The PDF pipeline therefore uses two PDF passes: the first locates every marked term; the second adds the sorted, deduplicated index with resolved page numbers. `prodockit.pdf.index` owns extraction and index construction.

29.4 Preserve website and PDF block behaviour

`prodockit.steps` follows the PyMdown Blocks API. Its `start` option emits both HTML `<ol start="9">` and CSS `counter-reset: list-item 8`. Browsers honour the HTML attribute while WeasyPrint needs the CSS counter-reset. Removing either representation makes continued numbering disagree between outputs.

Tree and steps blocks expose stable class names for author styles. Their parser and output details can change only with matching website, PDF, and built-output tests.

29.5 Preserve table layout contracts

`prodockit.tables` runs after Python-Markdown's table extension and changes

the generated HTML. Widths create a `<colgroup>` and mark the table with `.prodockit-table-sized`; compact tables use `.prodockit-table-compact`. Additional header rows move into `<thead>`, span placeholders are removed, and rotated headings use a transform because WeasyPrint does not reliably support `writing-mode`.

These transformations have consequences outside the extension. Zensical's theme styles many tables through `.md-typeset table:not([class])`, so adding a class means the project stylesheet must restore the normal border, padding, background, and dark-mode colours. Rotation also needs an explicit width because CSS transforms do not participate in layout. Keep website and PDF fixture coverage together when changing any of these contracts.

29.6 Preserve inline index content

The index inline processor runs early enough to retain nested Markdown such as emphasis, code, and links inside `\\index{...}`. The PDF stage later uses BeautifulSoup text extraction so the filing term is plain text even when the visible term contains nested HTML.

An `attr_list` placed after a linked index marker attaches to the outer index span, not the nested link. The author guide therefore recommends raw `<a>` markup when a linked term needs attributes. A parser change must cover this processor ordering and brace handling explicitly.

29.7 Preserve temporary attributes and public CSS hooks

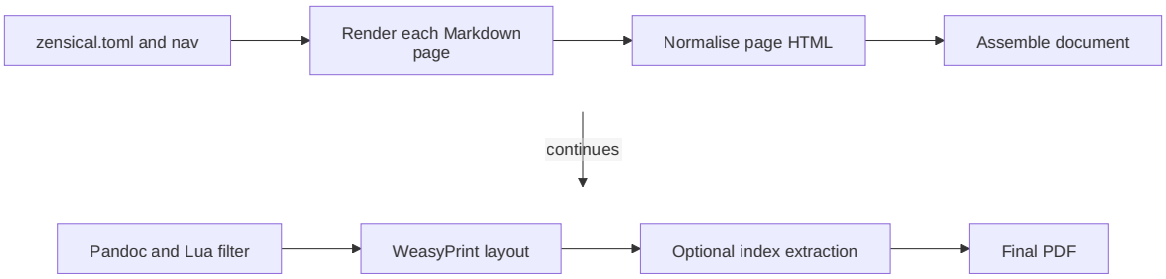
The references, citations, and glossary extensions use temporary `data-*` attributes while resolving definitions. Strip those attributes from the final HTML, while retaining the documented public classes for resolved and unresolved output. The PDF then adds page-number text to resolved `\autoref{}` links with CSS `target-counter()`.

The index is different: `data-index-term` and `data-index-code` remain in the rendered HTML because the PDF index stage consumes them after page layout. Treat those attributes as pipeline data rather than author styling hooks; the public styling hook is `.index`.

30. PDF pipeline and API

This PDF pipeline page is for contributors changing `prodockit.pdf` or calling its Python API directly. Document authors should use Generate a PDF.

30.1 Follow the pipeline



The configuration wrapper reads Zensical settings, renders each navigation page through Zensical, constructs `Page` objects, pre-renders diagrams and maths, and calls the lower-level builder. A generated index adds a second layout pass after term pages are known.

30.2 Use the public Python surface

API	Purpose
`build_pdf_from_zensical_config()`	High-level build using navigation and settings from `zensical.toml`
`build_pdf()`	Lower-level build from prepared `Page` objects
`Page`	One rendered source page plus its path, appendix, index, and running-header metadata
`PdfBuildError`	Build failure carrying the underlying command output
`build_source_bundle_from_zensical_config()`	High-level Markdown/configuration source-bundle build

Prefer `build_pdf_from_zensical_config()` when a caller already has a Zensical project. Use `build_pdf()` only when the caller owns page rendering and can supply complete HTML and metadata.

```
from prodockit.pdf.config import build_pdf_from_zensical_config

output = build_pdf_from_zensical_config("zensical.toml")
print(output)
```

The CLI wraps these functions with progress reporting, captured diagnostics, and non-zero exit status; the functions return paths or raise exceptions.

30.3 Know the internal modules

Module	Responsibility
`prodockit.pdf.config`	Zensical configuration, navigation flattening, page rendering, and high-level entry points
`prodockit.pdf.build`	Pipeline orchestration and external-command execution
`prodockit.pdf.html`	Page fix-ups, front matter, web/PDF-only content, and heading structure
`prodockit.pdf.lua`	Pandoc Lua filter generation
`prodockit.pdf.css`	Page size, margins, running headers/footers, duplex layout, and shared presentation
`prodockit.pdf.icons`	Icon discovery and SVG resolution
`prodockit.pdf.mermaid`	Mermaid CLI invocation and diagram assets
`prodockit.pdf.source_bundle`	Markdown/configuration source PDF
`prodockit.pdf.index`	Marker extraction, term-page mapping, and generated index
`prodockit.pdf.release`	Host release lookup for cover markers

Keep transformation stages narrow. A change to HTML normalisation, CSS, the Lua filter, or an external tool can affect every page, so verify the complete PDF and built-output tests rather than relying on a unit test of the changed module alone.

30.3.1 Preserve source-bundle boundaries

The `prodockit source-bundle` command includes Markdown below `docs_dir` plus `zensical.toml`. It is intentionally separate from the rendered-document pipeline: it discovers files with git, writes a self-contained HTML document, and calls WeasyPrint without Pandoc. The lower-level source-bundle API can discover

every non-ignored text file for a specialised caller, but that is not the command-line default.

30.3.2 Preserve real footer markup

Copyright text can contain links and line breaks, so it cannot be flattened into a CSS string. The PDF pipeline writes it as an HTML element and places it in the repeated footer with CSS Paged Media's `position: running()` and `content: element()`. Check the finished PDF when changing this path; intermediate HTML alone does not prove that links or line breaks survived.

30.4 Preserve actionable errors

`PdfBuildError` reports which external command failed and retains its captured output. Configuration wrappers also guard Zensical result-shape changes with a message naming the installed version and affected page. Do not replace these with an unlabelled subprocess status or raw `KeyError`; callers need to know which boundary changed.

31. Bootstrap design

This bootstrap design page is for contributors changing `prodockit bootstrap`. The author guide documents what to run; this page records the safety model behind its stages.

31.1 Model every stage as evidence and a plan

Each stage has two independent parts:

1. A `check` observes the machine or repository and reports `ok`, `missing`, `wrong`, or `blocked` with evidence.
2. A `plan` describes commands and human instructions that could reach the desired state.

`--apply` executes the approved plan and then runs the check again. A command returning zero is not proof that Python imports WeasyPrint, SSH authenticates, the correct remote exists, or a public site answers.

A check must be able to observe what its plan changes. Otherwise a successful application is followed by the same failing result and bootstrap cannot be safe to repeat.

31.2 Keep commands non-interactive

Every subprocess is non-interactive. Package managers receive unattended flags, Git and SSH disable credential prompts, and commands have bounded timeouts. A blinking prompt cannot be represented as a finding or resumed reliably by a later run.

Work requiring credentials uses **guide and verify** instead. Bootstrap tells the reader how to upload an SSH public key or create an empty project, then checks authentication or repository reachability. It never asks for or stores a personal access token.

31.3 Treat fresh history as destructive

The fresh history stage removes the template's Git records before creating the reader's own repository history. It is offered only when `origin` still points at the known template remote. A clone already pointing at the reader's project must never qualify.

This stage reports `wrong`, not `missing`, so pressing Enter cannot accept the destructive action. The plan explains exactly what is removed and requires an explicit answer.

31.4 Separate bootstrap and project environments

Bootstrap necessarily runs before the target project exists. After cloning, it creates the project's own `.venv` and invokes that environment's Python explicitly when installing `requirements.txt`. A bare `pip` could otherwise install project dependencies into bootstrap's environment and leave the checkout unable to build.

31.5 Preserve stage order

Stage order is dependency order: Git must exist before SSH and cloning; the clone must exist before its environment and Node tools; the remote must exist before a push; the push must trigger a pipeline before the site can be verified. A failure stops the run because later findings would be consequences of the first failure rather than independent work.

Manual instructions also declare whether they occur before or after commands. Passphrase advice must precede `ssh-keygen`; instructions for configuring an installed editor must follow installation.

31.6 Add or change a stage

When changing a stage, add tests for its complete state classification, plan, non-interactive command arguments, re-check behaviour, platform/host branches, and refusal boundaries. Test the real end-to-end path on supported operating systems when the change touches installers, shells, SSH, browsers, or host behaviour that a fake runner cannot reproduce.

32. Zensical coupling

Zensical coupling arises because prodockit imports several Zensical Python APIs that Zensical neither documents nor treats as public. Nothing else in these docs says which, so the coupling is invisible until a Zensical release breaks it. This page is that list.

> **Last verified against Zensical 0.0.55**
>
> Every call site and data shape below was checked against that version. A newer Zensical may have moved any of them - see Regression testing a Zensical upgrade.

32.1 Why this page exists

Zensical 0.0.55's `zensical/__init__.py` exports exactly three names:

```
__all__ = ["build", "serve", "version"]
```

prodockit uses none of them. Everything it reaches for is a module-level import from inside the package. Zensical's documentation site has no Python API reference at all, and upstream's own position is that the module API is not yet stable or generally available.

The practical consequence: any of these can be renamed or removed in a **patch** release without that counting as a breaking change from upstream's point of view. So a `zensical` bump needs a deliberate regression pass, not just a green test run.

32.2 Undocumented Python APIs

API	Call site
`zensical.config.get_config()`	`_zensical.py` `145`

API	Call site
`zensical.config.parse_config()`	`pdf/config.p 291`, `testing/ plugin.py:139`
`zensical.markdown.render.render()` and its `content` / `meta` result keys	`pdf/config.p 289,344,348-3`
`zensical.extensions.context.ContextPreprocessor.from_markdown()` and `.page.path`	`_zensical.py 91-104`
`zensical.version()`	`_zensical.py 60-62`

Three more appear only in the test suite. They are no less coupled - a rename breaks the build, it just breaks CI rather than a user's site:

API	Call site
`zensical.extensions.context.ContextExtension`, `Page`	`tests/ test_zensical_integration.py: 14`
`zensical.extensions.macros.MacroEnv`	`tests/ test_zensical_macros.py:9`
`zensical.config` + `zensical.markdown.render`	`tests/ test_docs_render_cleanly.py: 41-45`

32.3 Undocumented data shapes

Not imports, but just as breakable - prodockit reads structures whose keys are nowhere documented.

The resolved nav tree — `{"url", "is_index", "children"}`, read by `settings.py:26-28` and `_zensical.py`'s own `_flatten_nav()`. `is_index` appears nowhere in Zensical's documentation; it exists only in the loader's output.

`env.conf` inside `define_env(env)` — `zensical_macros.py:199`. The Macros page documents `define_env(env)`, `env.variables`, `env.macro` and `env.filter`, but not `env.conf`; the documented route to config from a macro is the `config` *template* variable.

The packaged icon directory layout — `<site-packages>/zensical/templates/.icons/...`, searched by `pdf/icons.py:66-79`. The docs describe `.icons` under your own `custom_dir`, never the path inside the installed package. This one degrades gently: the lookup already tries `material` and `mkdocs_material` first and is wrapped in its own `except`, so a layout change costs icons rather than the build.

32.4 What is documented, and therefore fine

Worth listing so this page doubles as a triage aid - if something breaks and it is on *this* list, a Zensical rename is unlikely to be the cause:

- `zensical.extensions.macros` with `module_name`, configured in `zensical.toml` — documented configuration, not an import.
- `zensical.extensions.emoji.twemoji` / `.to_svg`, likewise.
- `[project.theme.icon.admonition]` and the rest of the theme config prodockit's PDF pipeline reads.

These are strings in a config file that Zensical resolves itself. prodockit never imports them.

32.5 Robustness notes

32.5.1 Renames are guarded, and reported

Both lookups in `_zensical.py` guard the attribute access and the call, not just the import. A renamed `from_markdown`, a changed signature, or a `Page.path` that becomes `Page.src_path` all import perfectly and then raise `AttributeError` / `TypeError` - so guarding only the import left those surfacing as a stack trace from inside `zensical build`, with nothing pointing at the version bump.

prodockit now warns instead, naming the API, the installed Zensical version and what degrades:

```
prodockit expected Zensical's
ContextPreprocessor.from_markdown(md).page.path,
which raised AttributeError: ... Zensical 0.0.52 appears to have
moved it.
prodockit falls back to non-Zensical behaviour, so cross-page
references,
citations and glossary terms may resolve to their unresolved marker
instead
of the real target.
```

> ⚠ **The guards do not silently swallow the failure, deliberately**
>
> Returning `None` quietly would be worse than the crash it replaces. `page_source()` returning `None` makes every page fall back to the same default source, so each render wipes the previous page's registry entries and cross-page references resolve to `??` — on a site that still builds and exits zero.
>
> A plain `ImportError` *is* silent, because "not running under Zensical" is a legitimate state for any other Python-Markdown consumer. Each API is reported once per process rather than once per page, since `page_source()` runs on every render.

32.5.2 `parse_config()` writes a module-level global

`zensical.config.parse_config()` sets Zensical's module-level `_CONFIG` - the same global `get_config()` reads.

Harmless today: `prodockit pdf` and the pytest fixtures each run in their own process. But calling `build_pdf_from_zensical_config()` or using the `prodockit_resolved_config` fixture *inside a live build process* would silently rebind that build's config. Worth knowing before wiring the PDF build into a Zensical plugin.

32.5.3 The unguarded imports are correct

`pdf/config.py` and `testing/plugin.py` import Zensical without a guard. That is deliberate - Zensical is a hard requirement for both, and failing loudly is right. `render()`'s `result["content"]` / `["meta"]` keys carried that same rename risk unguarded until prodockit-extensions#171: a `KeyError` / `TypeError` there is now caught and re-raised as a `RuntimeError` naming Zensical, the installed version, and the page being rendered - the build still stops (there is no sensible degraded PDF to fall back to, unlike the warn-and-degrade shape above), it just says why. `testing/plugin.py`'s own `parse_config()` call reads no result keys, so it has no equivalent exposure.

32.6 Regression testing a Zensical upgrade

None of the above is a documented interface, so **a green unit-test run is not enough**. A Zensical bump needs a deliberate pass:

1. **Run the full suite**, `tests/test_zensical_integration.py` especially - it exercises the per-page render context directly.
2. **Build the site *and* the PDF, and diff the output** rather than checking both exit zero. Zensical 0.0.52 silently redrew the GitHub brand icon (Font Awesome 7.2.0 → 7.3.1) with no source change, which is exactly the drift a pass/fail check misses. The drift job does this comparison automatically.
3. **Check cross-page `\ref` / `\citeref` / `\gls` resolution specifically.** These depend on the nav pre-scans and on `ContextPreprocessor` reporting the right page path, and they degrade to `??` rather than failing.
4. **Confirm the data shapes.** The resolved nav tree should still carry `url` / `is_index` / `children`, and `render()` should still return `content` / `meta`.

`zensical` is pinned exactly in the build so an upgrade is a decision rather than something that happens overnight - which is what makes a deliberate pass possible at all.

32.7 Related

- Implementation limitations covers the theme CSS-class coupling - `.md-typeset` targeting, glightbox wrappers, and Zensical's own `LinksTreeprocessor` URL rewriting. Those are HTML/CSS shapes rather than APIs, and equally undocumented.
- Version pinning and drift covers pinning `zensical` and watching for a newer release.

33. Implementation limitations

Confirmed limitations across prodockit's three surfaces - the Python- Markdown extensions, `prodockit.pdf`, and `prodockit.zensical_macros` - and the workaround each one gets, so a project hitting unexpected output has somewhere to check *why* before assuming it's a bug.

This contributor reference explains implementation causes and regression risks. Document authors should start with the shorter, symptom-led Known limitations page.

33.1 Extensions

Cross-page resolution can go stale under `zensical serve`'s live reload: every prodockit extension that resolves something defined on a *different* page (`prodockit.refs`' `\ref{id}`, `prodockit.citations`/ `prodockit.glossary`'s definitions, `prodockit.bibliography`'s `\cite{id}`/ `\bibliography`, and `prodockit.headings`' own continuous-numbering pre-scan) does so via a pre-scan of every nav page's raw text, since `zensical build`'s single, one-shot pass can't otherwise resolve a forward reference to a page it hasn't rendered yet. Under `zensical build` that's all it has to do. Under `zensical serve`, the pre-scan itself now picks up an edited or deleted definition correctly (keyed on every page's mtime/ size, not just computed once at server startup) - but that only fixes what a page's *own* re-render sees, not whether Zensical re-renders that page at all: verified directly against a live `zensical serve`, editing page A does not cause page B to re-render, so B's own displayed output still only catches up once B itself is rebuilt (e.g. by editing it, or restarting the server) → no full fix available here, since that half is Zensical's own incremental-rebuild behaviour, not prodockit's.

Duplicate heading names across pages: `prodockit.headings`' automatic Zensical registry sharing (see Sharing a registry across a multi-page build) logs a warning and keeps the first registration rather than raising, and *which* page's registration wins isn't stable across builds - see the warning admonition partway through that same section for why, and the fix (an explicit, unique, page-prefixed id via `attr_list`).

`prodockit.bibliography` matches a single citation key only - `\cite{id1,id2}` isn't supported, unlike `prodockit.citations`' own `\cite{id1,id2,...}` → falls through as literal text rather than being silently mishandled; see Comparing the two approaches for why.

`prodockit.index`'s nested sub-entries are visually capped at three levels: `\index{Parent!Child!Grandchild}` nests correctly to any depth the parser itself supports, but the generated index's own CSS only defines an indent step up to the third level → a fourth level and beyond is clamped to that same, deepest available indent rather than continuing to step outward.

33.2 PDF generation

`prodockit.pdf` pipes your site's own rendered HTML through Pandoc and WeasyPrint to produce the PDF - two tools with their own reader/writer quirks and no JS engine, quite different from a browser rendering your live website. This section documents the confirmed limitations that shape `prodockit.pdf.html` / `.lua` / `.css`, and the workaround each one gets.

No JS engine (WeasyPrint can't run client-side JS)

- Mermaid diagrams: no JS engine to run Mermaid.js client-side → each ```` ```mermaid ```` fence is pre-rendered to a static SVG via `mermaid-cli` before Pandoc ever sees it (see PDF internal modules).
  - Mermaid's default node/edge labels are HTML `<foreignObject>` content, which WeasyPrint's SVG renderer can't display (text silently vanishes) → `htmlLabels` is forced off, so Mermaid emits plain SVG `<text>` / `<tspan>` labels instead.
- Math (`$...$` / `$$...$$`, `pymdownx.arithmatex`): no JS engine to run MathJax client-side → each formula is pre-rendered to a static SVG via a Lua filter `Math()` handler piping to a `tex2svg` script (see PDF internal modules).
  - `arithmatex`'s *generic*-mode math (`<div class="arithmatex">` / `<span class="arithmatex">`) has no native Math AST node in Pandoc's *HTML* reader (unlike its *markdown* reader, which recognises `$...$` as a real Math node) → matched by CSS class in dedicated `Div()` / `Span()` Lua handlers instead of the `Math()` function.

> ⚠ **Both renderers are optional, and their absence is announced**
>
> `mermaid-cli` and the `tex2svg` script are external Node tools, not Python dependencies, so neither is guaranteed to be present. When one is missing, the affected content is left exactly as it is rather than failing the build - a document with no diagrams and no maths should never need either tool installed.
>
> The catch is that a document which *does* use them then gets a PDF containing raw `flowchart LR ...` source or literal LaTeX, with nothing having gone wrong as far as the build is concerned. Since 0.12.0, `prodockit pdf` prints a warning naming the missing renderer and how to install it whenever that combination occurs.

Pandoc's HTML reader decides what is still a code block

- Zensical's highlighter emits per-token `<span>`s, a `__codelineno` anchor per line, and a leading empty `<span></span>` inside `<pre><code>`. Pandoc's HTML reader only treats `<pre><code>` as a code block when that `<code>` holds nothing but text, and each of those constructs defeats it independently

→ every `<pre>` is reduced to a single plain-text `<code>` child before Pandoc sees it (in PDF internal modules).

A version of pandoc decided this, and CI could not see it

This is the clearest example the project has of an external tool changing under it, so it is worth stating in full.

Pandoc **3.1.3** accepted that markup as a code block. Pandoc **3.10** does not. Nothing in this repository changed - not Zensical, which emits byte-identical markup from 0.0.50 through 0.0.55, and not prodockit, which had never touched the construct.

When the reader gives up, the `<pre>` is absent from what Pandoc hands WeasyPrint, so `white-space: pre-wrap` has nothing to apply to. Every newline collapses, the block reflows and justifies like a paragraph, and each token becomes its own inline `<code>` - carrying the inline-code background with it. A six-line install snippet came out as four wrapped rows with `". [dev]"` split across two of them. It was reported as stretched word spacing, which is what it looks like; the spacing was justification padding a gap a fixed-pitch font should never have.

CI published perfect PDFs throughout. The runner image's own `pandoc` package was 3.1.3, so every automated build was correct while every local build on a current pandoc was wrong. The two artefacts disagreed for as long as nobody compared them, and the next runner image bump would have broken the published output with no commit to blame.

Two things came from that. The build pins an upstream pandoc release rather than taking the image's (see Pandoc version drift), so a pandoc change arrives as a version bump that can be bisected. And the check for it measures the finished PDF - the gap between two monospace characters that are adjacent in the text flow - rather than the HTML handed to Pandoc, because the intermediate shape being right is exactly what was true while the artefact was wrong.

- A live site's own header repo widget (release/version info) fetches it client-side via JS; Pandoc/WeasyPrint has no JS engine to do the same → a project embedding similar info in a PDF cover page needs to fetch and substitute it directly before the page ever reaches `build_pdf()`.

The website and the PDF resolve the release differently

The two release values come from deliberately different sources, and they can disagree:

	Source
`{{ release }}` (website, and any macro-rendered page)	`git describe --tags` on the local checkout
`{RELEASE}` (PDF cover marker)	The host's releases API

Each is right for its own context. `{{ release }}` is re-evaluated on every website rebuild, including every save under `zensical serve`, so it must not make a network call. `{RELEASE}` serves a cover page that isn't part of a macro-rendered site at all, so a local git lookup isn't always available to it.

They diverge when a tag exists with no published release (the website shows a version, the PDF drops the line), when a release exists but the checkout has no tags (the reverse - usually a shallow clone), and during the window in a release where the version-bump commit is pushed before its tag.

Since 0.16.0 neither is silent about it: `prodockit pdf` warns when the two will show different things, and the macros pass warns when `{{ release }}` came back empty *because* the clone was shallow, naming `fetch-depth: 0` / `GIT_DEPTH`. A project with no tags at all is a normal state and says nothing.

If you need them guaranteed identical, set the value explicitly rather than relying on either mechanism.

Multi-page → single-document concatenation

- A link that resolves fine on a website (a separate page) has nothing to point at once every page is concatenated into one PDF - Pandoc treats it as a link to an external file at whatever absolute path the PDF happened to be built from → rewritten to in-document anchors instead (see `build_page_anchor_map()` / `build_virtual_page_map()` in PDF internal modules).
- Local image/file references can't depend on relative paths resolving correctly from wherever Pandoc happens to run in a standalone document → base64-embedded as `data:` URIs directly in the HTML (see `to_base64_data_uri()`).
- A PDF has no light/dark toggle to make the mkdocs-material/Zensical `#only-light` / `#only-dark` / `#gh-light-mode-only` / `#gh-dark-mode-only` image convention meaningful → the `#only-dark` / `#gh-dark-mode-only` half of a pair is dropped entirely rather than embedded, since `to_base64_data_uri()`'s resulting `data:` URI has no trace of the fragment

left for any stylesheet to hide it by (this used to leave both halves of every such pair permanently visible, stacked one after the other).
- A CSS `url()` reference (e.g. in your own website stylesheet, passed via `extra_css`) resolved relative to wherever Pandoc runs is meaningless (and can leak a local file path) to anyone reading the PDF → a project passing its own CSS through `extra_css` should rewrite any such reference to a stable, absolute URL (e.g. the file's canonical GitHub/ GitLab "blob" URL) before handing it to `build_pdf()`.

Raw `<svg>` doesn't survive Pandoc's HTML→HTML round trip through to WeasyPrint at all (confirmed directly, isolated test) - affects admonition icons, grid-card title icons, and pre-rendered Mermaid diagrams alike → every `<svg>` is converted to a base64 `data:` URI `<img>` instead (see PDF internal modules).

Content tabs (`pymdownx.blocks.tab`): each tab's label renders as an inline `<label>` sibling with no block boundary between them; Pandoc's HTML reader merges adjacent inline-level siblings with no block boundary into one `Plain` block, collapsing every label in a tabbed-set into one unseparated run of text with no way to recover the boundary afterward in a Lua filter → each `<label>` is rewritten into its own `<p>` *before* Pandoc's reader ever sees it, then the Lua filter reconstructs the `tabbed-set` / `tabbed-labels` / `tabbed-content` structure into a `tabbox` shape (see PDF internal modules).

Figure/table captions in "prepend" position: Pandoc's `Figure` AST node stores `Caption` and content as two separate, independently-typed fields rather than ordered children reflecting DOM position, and Pandoc's own HTML writer always re-emits a `Figure`'s `<figcaption>` *after* its content when serializing back to HTML - confirmed directly (a `<figcaption>` placed first in source HTML still comes out last from Pandoc's own HTML writer), discarding "prepend" positioning entirely regardless of input order → any figure/table whose caption comes first is retagged from `<figure>` to `<div>` before Pandoc parses it (a `Div`'s children *are* emitted in original document order), with the `<figcaption>` unwrapped into the div's first child block.

Pandoc's native `Para` AST node has no attribute field at all (unlike `Div` / `Header` / `CodeBlock` / `Table` / `Figure`, which all carry one) - confirmed: a `<p id="..." class="...">` comes out the other end as a bare `Para` with both the `id` and the `class` silently gone, with no error. This is exactly the shape every `attr_list` citation/acronym/glossary definition takes (see prodockit.citations/ prodockit.glossary) → any `<p>` carrying an `id` or `class` is retagged to a `<div>` instead (which Pandoc's reader does preserve attributes on).

Lightbox-wrapped images: an `<a class="glightbox">` wrapping an `<img>` resolves its `href` one directory level differently than the `<img>`'s own `src` (an artifact of Zensical's URL cleaning), which Pandoc/WeasyPrint then fails to resolve as a broken link → the lightbox `<a>` is unwrapped, leaving just the `<img>`.

Embedded `<iframe>` (e.g. a YouTube video): left as-is, produces a stray unwanted heading in the compiled PDF (WeasyPrint attempts to fetch the iframe's

`src`, and something in that response ends up parsed as real page content) → replaced with a link-styled reference to the video instead - a static PDF can't embed a live video player regardless.

No Jinja evaluation: Pandoc/`prodockit.pdf` never evaluates Jinja - a `{{ site_name }}` placeholder that resolves via macro evaluation on the live site (see prodockit.zensical_macros) is left as literal text unless a project substitutes it directly in its own page HTML before handing it to `build_pdf()`.

No `.md-typeset` wrapper: unlike a Zensical website, Pandoc's HTML output has no `.md-typeset` wrapper element, so website CSS rules scoped to `.md-typeset ...` (reference/acronym/glossary spacing, a `.screenshot` class, and so on) never match in the PDF → `prodockit.pdf.css` duplicates the relevant rules as plain, unscoped selectors instead (see PDF internal modules).

Footnotes: Pandoc's default behaviour collects every footnote in the whole document into one section at the very end of the PDF, rather than at the bottom of the page it's referenced on like a printed book → a Lua filter handler replaces each footnote reference with an inline span styled via CSS `float: footnote` instead (see PDF internal modules).

WeasyPrint's CSS Grid support is too limited to trust for an actual side-by-side multi-column layout → a Zensical grid-cards block renders as one full-width stacked box per row instead of a real grid.

`<figcaption>` centering doesn't extend to its sibling `<img>`: WeasyPrint's UA stylesheet centers `<figcaption>` text by default via `text-align`, but that doesn't affect the sibling image, which stays left-aligned and visibly misaligned under its own caption → explicit CSS centers the whole figure/wrapping element instead.

Two-space vs. four-space nested-list indentation discrepancy: Pandoc's markdown reader nests a sub-list at just 2-space indentation (no 4-space requirement), unlike Python-Markdown's stricter 4-space rule. Only relevant if you write markdown by hand for a *separate* Pandoc-only input rather than feeding `prodockit.pdf` your already-rendered HTML (the normal, documented path) - the HTML-based pipeline sidesteps this entirely, since Pandoc's HTML reader has no such indentation rule to begin with.

General "every markdown extension needs its own bespoke translation" limitation, and why `prodockit.pdf` avoids it: Pandoc is a completely different parser from Python-Markdown/Zensical, so a pipeline built around hand-translating each markdown feature into a Pandoc-compatible dialect needs a new bespoke translation for every extension a project enables (admonitions, tabs, grid cards, captions, `attr_list` spans, `{% if %}` conditionals, and so on) - fragile, and it grows without bound. `prodockit.pdf` sidesteps this by feeding Pandoc your site's own already- rendered HTML (via `zensical.markdown.render.render()`, the same pipeline that builds your live website) instead of raw markdown - Pandoc's own HTML reader already understands standard HTML correctly, with no per-feature translation needed. The

fixups documented above are what's left after that: genuine gaps in Pandoc/WeasyPrint's own HTML handling, not gaps in markdown-dialect translation.

`prodockit.bibliography` is a partial exception to this pattern, worth flagging explicitly: resolving `\cite{id}` / `\bibliography` itself calls out to a *separate*, independent `pandoc --citeproc` invocation at markdown-render time (see prodockit.bibliography) - unrelated to, and already finished well before, `prodockit.pdf`'s own `pandoc --pdf-engine=weasyprint` call below. By the time `prodockit.pdf` sees the page, citations and the reference list are already resolved, ordinary HTML - `id`-bearing `<div>`s and `<a>` links like any other content - so none of the fixups documented above apply to it specially; a build using both ends up invoking Pandoc twice, for two entirely unrelated reasons.

33.3 Website macros

`heading_counter_reset(page)` inherits the same `zensical serve` staleness bound as extensions above: it calls `prodockit.headings.prescan()` directly - the identical pre-scan continuous numbering itself uses - so a page's displayed chapter/section number can lag behind an edit to an *earlier* page's heading count until that later page is itself rebuilt under `zensical serve`'s live reload. Not an issue for a one-shot `zensical build`.

`{{ repo_url }}` reflects the local checkout's own git remote, not `project.repo_url`: computed from `git config --get remote.origin.url` directly, deliberately, so it reflects wherever *this* checkout actually points (e.g. a CI job's own token-embedded remote, stripped before display) - in practice this usually, but isn't guaranteed to, match `zensical.toml`'s configured `repo_url` (used elsewhere for the sidebar repository link). A fork or a differently-configured clone can show a different URL from the two.

`{{ word_count }}` assumes the first page in `nav` is a cover page and unconditionally excludes it from the count, on top of any page explicitly flagged `exclude_from_word_count: true` - a project whose `nav` doesn't start with a dedicated cover page gets a word count silently short by that first page's own prose.

`heading_counter_reset()` / `reference_style()` / `acronym_style()` / `glossary_style()` all emit CSS targeting Zensical/Material for MkDocs' own internal class names and counters (`.md-typeset`, `.md-nav--secondary`, `counter(h1-count)` / `counter(toc1)`, and so on) - undocumented implementation details of the theme itself, not a public API it commits to - so a future Zensical theme restructuring could change or remove them without warning, silently breaking the numbering/spacing display with no error raised.

These CSS-shape couplings are the theme-side counterpart to the Python ones - see Zensical coupling for the full list of undocumented Zensical APIs prodockit depends on, and what regression testing a Zensical upgrade actually needs.

34. About prodockit

This section is for anyone evaluating or using prodockit who needs to understand its scope, dependencies, support, release history, or legal terms.

prodockit is a Python package for professional and academic documentation built with Zensical. It adds Markdown extensions, website macros, PDF generation, output checks, and project-management commands without requiring a document author to build those pieces independently.

34.1 Understand the foundation

prodockit builds on Python-Markdown, Zensical, and PyMdown Extensions. PyMdown Blocks is a direct foundation, not merely a compatible syntax: `prodockit.steps` and `prodockit.tree` are implemented with its Blocks API and use the same slash-fenced container model as PyMdown's own blocks. Installing prodockit therefore installs `pymdown-extensions` too.

This relationship lets prodockit add specialised document blocks while retaining the nesting, configuration, and rendering conventions familiar to Zensical authors. Start with Numbered steps or Directory trees to see the shared model in use.

34.2 Know the project family

Project	Role
prodockit-extensions	The Python package, command-line tools, implementation, tests, and this technical reference
prodockit-template	The maintained starting project, including annotated GitHub and GitLab publishing automation
prodockit-userguide	The task-based guide for people creating and publishing documents with the template

The template is maintained on GitHub and synchronised to the University of Surrey's GitLab for student use. Projects created from it use the same prodockit package documented here.

34.3 Choose where to continue

Page	Use it for
Get started	Install prodockit and build a first local site
Authoring reference	Add document features to Markdown pages

Page	Use it for
Publish a document	Produce and deploy a website and PDF
Support and compatibility	Check maturity, supported versions, platforms, and known constraints
Release notes	See new features, fixes, and upgrade actions by version
Licence	Read the MIT License governing use, modification, and redistribution

Repository maintainers should use Maintain prodockit. Developers changing the package itself should use Contributor internals.

35. Support and compatibility

This page is for anyone deciding whether prodockit fits a project or checking whether their platform and tool versions are covered. It describes the public support and compatibility boundary; implementation-specific constraints remain in Contributor internals.

35.1 Maturity and stability

prodockit is currently classified as **Alpha** and uses pre-1.0 versions. Its document features and publishing tools are functional and tested, but there is not yet a formal, versioned public API stability contract. A pre-1.0 upgrade can therefore include a breaking change when the change is needed to make the public configuration consistent. Such changes are identified in the release notes.

The future stability contract is tracked in issue #7.

35.2 Required versions

Installing prodockit installs these Python dependencies automatically:

The current test matrix covers Python 3.10, 3.11, 3.12, and Python 3.13. The documentation build currently pins Zensical 0.0.55 and pymdown-extensions 11.0.1 so changes to either renderer arrive as reviewed version changes rather than silently altering published output.

Requirement	Supported or tested range	Why it matters
Python	3.10–3.13 tested	The package requires Python 3.10 or later
Zensical	0.0.55 or later	Site configuration, rendering, navigation, macros, and icons
Python-Markdown	3.10.3 or later	The extension engine used by every authoring feature
pymdown-extensions	11.0.1 or later	PyMdown Blocks is the direct foundation for `prodockit.steps` and `prodockit.tree`; the PDF pipeline also preserves PyMdown output

PyMdown Blocks is particularly important when evaluating the authoring model. The numbered-steps and directory-tree extensions are specialised Blocks API implementations, so their slash fences, nesting rules, and option layout follow PyMdown's block syntax. They are not separate parsers that only resemble it.

Pandoc, WeasyPrint, Node tools, fonts, and other optional publishing requirements depend on the output being built. See Installation for that complete tool-by-tool boundary.

35.3 Platforms and test depth

The platform testing described here covers Linux, macOS, and Windows. Bootstrap has now completed manual end-to-end testing on Ubuntu Linux, Windows, and macOS against both the University of Surrey's GitLab (`gitlab.surrey.ac.uk`) and GitHub.com.

Two complete document workflows were exercised:

1. **Start a new document:** prepare the machine with bootstrap, create a new document repository on the selected host, and reach a working local build.
2. **Adopt an existing document:** take an existing online repository, install it locally on the prepared machine, and reach a working local build.

This is practical integration testing across the three operating systems, two hosts, and both common starting points. It verifies that the stages work together in real environments, beyond unit tests or inspection of generated commands. It is not an automated cross-platform regression matrix, however:

Platform	Regression test coverage	Manual bootstrap coverage
Ubuntu Linux	Full test suite on every push and pull request using `ubuntu-24.04`	Both repository workflows on Surrey GitLab and GitHub.com
macOS	The full test suite is also run locally on macOS; there is no hosted macOS job	Both repository workflows on Surrey GitLab and GitHub.com
Windows	No hosted test job	Both repository workflows on Surrey GitLab and GitHub.com

The manual matrix gives confidence in installation and first-use integration, but it is a point-in-time result. The locally run macOS suite adds full regression coverage on that platform, although it is not enforced by a hosted pull-request gate. Windows can still regress between manual runs because it does not yet run the full suite for every change.

Windows requires native libraries for PDF generation that `pip` cannot install. It also uses different default text encodings. The known setup steps are documented under PDF requirements, and `prodockit bootstrap` checks the corresponding tools. Report a Windows-only failure rather than assuming it is local configuration; it may expose a real coverage gap.

Bootstrap also recognises GitLab.com, but the completed manual platform matrix described above covers Surrey GitLab and GitHub.com. See Set up a machine for the host boundary.

35.4 Supported surfaces

Surface	Current support
Markdown extensions	All nine registered extensions are documented and tested
PyMdown Blocks integration	`prodockit.steps` and `prodockit.tree` directly use the Blocks API
Zensical website macros	Implemented and tested; tied to some pre-1.0 Zensical internals
PDF and source bundles	Implemented and tested with Pandoc and WeasyPrint; external renderer versions can affect layout
GitHub and GitLab publishing	Maintained through the annotated workflows in prodockit-template
Repository and template commands	Implemented and tested; commands that write files provide a report or dry-run path first

For observable constraints such as live-reload staleness, unsupported citation forms, and differences between browser and PDF rendering, see Known limitations. If the behaviour is not listed there, search or open a GitHub issue.

35.5 Upgrade safely

Read the release notes before changing a pinned version. Build the site with `zensical build --clean --strict`, build the PDF if the project publishes one, and run its built-output checks before merging. Template-based projects should also review changes reported by template-sync.

36. Known limitations

This page is for document authors. It describes known limitations by what you see and what you can do next. For causes, design trade-offs, and regression risks, see the Contributor internals.

36.1 A cross-page value looks stale during live preview

What you see: after editing a heading, citation, glossary definition, or index-related value on one page, another open page can still show its previous value under `zensical serve`.

What to do: save or refresh the affected page, or restart `zensical serve`. Always run `zensical build --clean --strict` before publishing; a clean build resolves the complete document together.

36.2 A duplicate heading link resolves unpredictably

What you see: two pages use the same generated heading id and a cross-reference can resolve to the wrong page.

What to do: give each referenced heading an explicit, unique id, preferably with a page prefix, such as `{#methods-sampling}`.

36.3 A bibliography citation remains as literal text

What you see: `\\cite{first,second}` is unchanged when using `prodockit.bibliography`.

What to do: cite each BibTeX entry separately. Multiple keys in one command are supported by `prodockit.citations`, not by the BibTeX extension.

36.4 Mermaid or maths source appears in a PDF

What you see: a PDF contains Mermaid source or TeX rather than a rendered diagram or formula.

What to do: run `prodockit init-tools`, install the reported Node tools, and rebuild. The website can render these features in the browser, but the PDF must convert them before WeasyPrint runs.

36.5 Browser and PDF layouts differ

What you see: tabs, grids, captions, footnotes, videos, or interactive elements have a simpler layout in the PDF.

What to do: treat the PDF as a print layout, check the built artifact, and use the PDF-specific CSS hooks documented with the relevant feature. A PDF has no browser JavaScript or interactive controls.

36.6 The word count omits unexpected content

What you see: `{{ word_count }}` is lower than expected.

What to do: keep a dedicated cover page first in `nav`. The first page is excluded automatically, as are pages marked `exclude_from_word_count: true`.

36.7 A limitation is not listed

Run a clean build and check the current local command help first. If the problem remains, search or open a GitHub issue with the prodockit, Zensical, Python, and operating-system versions.

37. Release notes

Entries are arranged newest first. Read the Unreleased section when it is present, then every version between the one a project uses and the one it will install. A change that needs author or maintainer action is described with the release that introduces it. Packaged artefacts and tags are available from GitHub Releases.

The oldest entries record the package's earlier name, `zendoc`. Versioning restarted at `prodockit` 0.1.0 after that rename, so the historical sequence near the bottom appears to move from prodockit 0.1.1 back to zendoc 0.10.0. Those are two package eras rather than duplicate releases.

37.1 0.41.0 (2026-08-20)

- **Changed:** back-of-book index generation is now configured with the extension it belongs to:

  ```
  [project.markdown_extensions."prodockit.index"]
  include = true
  title = "Index"
  ```

  This is a breaking configuration change. Replace the former `project.extra.pdf_include_index` and `project.extra.pdf_index_title` settings; they are no longer read. The extension now declares both options itself, so Zensical, the Markdown extension, and the PDF builder use one public configuration surface.

- **Fixed:** a nested `index.md`, such as the new About overview, is no longer treated as a second PDF cover. Only the first navigation page can be the compiled document's cover; later directory indexes retain their chapter number and PDF bookmark.

- **Fixed:** `prodockit bootstrap --help` now says that `.pdk-bootstrap.toml` in the current directory is the default configuration file. It previously described a user configuration directory even though the command did not use that as its normal default. Existing legacy user configuration remains readable.

- **Changed:** fenced and inline code in the PDF render at 11pt, one point below the 12pt body text. This compensates for the monospace face appearing optically larger at the same nominal size.

- **Fixed:** short content-tab panels can opt into page-safe PDF rendering by wrapping the tab group in `.pdf-keep-tab-pages`. Each reconstructed tab header now stays with its body, avoiding an empty panel fragment at the foot

of one page with all of its content on the next. Ordinary tabs remain splittable because a panel can legitimately be taller than a page.

- **Documentation:** reorganised the site's information architecture around five audiences and tasks: getting started, authoring, publishing a document, maintaining prodockit, and contributor internals. New overview pages route each reader before the detailed reference, while About now centralises maturity, platform coverage, dependencies, limitations, release history, and licence information.

- **Documentation:** rewrote all nine authoring references for a reader with limited Markdown experience. Each guide now follows the same enable, write, and configure progression and provides a rendered result for every syntax section. The numbered-steps and directory-tree guides also make their direct foundation on PyMdown Blocks explicit.

- **Documentation:** added a document-author publishing path from `prodockit-template` and machine setup through local PDF and strict website builds, built-output tests, GitHub or GitLab Pages deployment, and public verification. The guide distinguishes template-owned publishing files from an author's content and explains the Surrey `is_surrey` presentation choice.

- **Documentation:** reorganised project maintenance into a task-based path from a safe report-only check through reviewed updates, local builds, pull request gates, package publishing, and live-site verification. The release runbook follows this repository's five GitHub Actions workflows and explains where each stage resumes after a failure.

- **Documentation:** expanded `README.md` and `CONTRIBUTING.md` into current entry points for package users and contributors. They now distinguish Python packages from external PDF requirements, record the tested Ubuntu, Windows, and macOS workflows, inventory the public commands, and give the complete local verification order.

37.2 0.40.0 (2026-08-19)

- **Added:** `prodockit template-sync --push` finishes a sync where the pipeline can see it (#502).

  `--apply` stops at staged, on its own branch, and that publishes nothing: both hosts build only from the default branch, so a sync sitting on a `template-update-...` branch produces no pipeline and no rebuilt site even after you commit and push it. `--push` commits, merges into the branch your host builds from, and pushes - showing what it will do and waiting for a yes first.

  It assumes you merge your own work, with no merge request in the way. A project that gates its default branch should use `--apply` alone and raise a merge request from the update branch.

- **Fixed:** a stale checkout beside a project no longer stops `template-sync` (#500).

  A checkout beside the project wins over fetching, which is right for somebody editing the template and their project together - but it won unconditionally, so an old clone taken before the template carried a manifest stopped the command outright, with a usable copy one fetch away. A sibling now has to carry a manifest to be preferred, and the run says which checkout it passed over.

- **Fixed:** the installation page lists every extension (#504).

  It explained how to enable an extension and then named four of the nine. `prodockit.tables`, `.tree`, `.steps`, `.bibliography` and `.index` had each arrived with an entry point and a documentation page without reaching the one page a new reader goes to first. A test now treats the entry points as the authority, so a registered extension cannot go undocumented again.

- **Changed:** the `template-sync` manual covers a finished sync, not just `--apply` (#503).

  What to do with a `.new` sidecar and how to tell "you edited this" from "you never received this update"; that `--force` takes exact paths one flag at a time; that a second run branches again; and that committing alone changes nothing on the host.

- **Added:** `\ref{id}` resolves a captioned figure or table, not only a heading (#506).

  A reference to a figure renders its label - **Figure 3.1** - linked to the figure, and updates itself when figures are added or moved. Until now the number had to be typed by hand beside a `\ref` that could not resolve it, so a document said "Figure 3.1" in prose while the figure itself was numbered by the stylesheet, with nothing keeping the two in step.

  Figures and tables count separately, and both take the page's chapter number - the same numbering the caption shows, so a reference and its target always agree. Numbered in the same pass that numbers headings, which is where the chapter number already exists.

  A caption reference is its label alone, where a heading reference keeps its name: "the components in Figure 3.1" reads badly as "Figure 3.1 Component Model", while a bare "1.1" says nothing about where a reader is being sent.

37.3 0.39.0 (2026-08-19)

- **Added:** `prodockit template-sync`, which brings a project back into step with the template it came from (#495, #498).

  A project generated from a template is a copy, not a link. It starts ageing

immediately - the template gains a CI fix, a stylesheet rule, a newer pin - and nothing says so, because nothing breaks. The site still builds and the document simply looks slightly unlike everyone else's.

```
prodockit template-sync           # report; writes no project
file
prodockit template-sync --apply   # branch, write, stage, do not
commit
```

What is written is decided by a manifest in the template, not by the command. The report, its figures and its bibliography are never written and never even read, so a sync cannot lose your writing. A template-owned file you have edited is kept, with the template's version written beside it as `.new` to compare; `--force` takes the template's copy for a named file.

The template is fetched into a per-user cache, so no checkout of it is needed. A project on Surrey's GitLab tracks the Surrey mirror and everything else the GitHub copy, unless `--github`, `--surrey` or `--template-path` says otherwise. A host that cannot be reached is a third answer rather than a failure - the run continues on the cached copy and says that it may be behind.

Built for repeated use through a project rather than once at the start. A run with nothing to do says so and creates no branch; a `.new` sidecar already holding the template's bytes is not rewritten; and the recorded baseline moves forward even when a template release changes only how files are classified.

Every run appends its full account - always the `--verbose` form, and including runs that failed - to `.prodockit-template.log`, which the command adds to `.gitignore` itself.

37.4 0.38.0 (2026-08-18)

- **Added:** multi-row table headers, merged cells and rotated headings (#474).

  A Markdown table has one header row and no syntax for a second, so a heading needing two lines is written as a body row - and then stops repeating when the table breaks across pages, because only `<thead>` repeats. `{: .header }` moves the row where it belongs:

  ```
  | Target {: rowspan=2 } | Measured {: colspan=2 } | | Note {:
  rowspan=2 } |
  |---|---|---|---|
  | | Before {: .header } | After | |
  ```

  Both lines repeat on every page now, verified on a five-page table.

WeasyPrint always could repeat a multi-row header, spans and all; nothing had ever put the second row in the header for it to repeat.

`colspan` / `rowspan` are `attr_list`'s own attributes - what is new is that the empty placeholder cells they need are removed, so the row is not left wider than the header. One with text in it is kept.

Headings turn with `{: rotate=270 width="1.8em" height="105pt" }`, at 90 or 270 and nothing else. `rotate` without `width` is refused: `transform` does not affect layout, so the width is what buys the space and the rotation is what keeps the heading readable once the column is narrow. Rotating alone renders a heading in a full-width column and looks like it worked.

`transform` in both outputs rather than `writing-mode`, which WeasyPrint ignores silently - measured, the text stays horizontal while the column still narrows.

- **Added:** `{: .compact }` for a table with many short columns (#489).

  A wide table is held wider still by the theme itself: every header cell carries `min-width: 5rem`, and every cell 1.25em of padding either side. A column holding `H` is then as wide as one holding a sentence, so the table overflows whatever it contains. Measured on a real 14-column table against 1009px of A4 landscape:

  ```
  as written                    1586.7px   (57% over)
  minimum dropped               1190.7px
  and the padding tightened      993.1px   (fits)
  ```

  Neither is enough alone, so one marker turns off both:

  ```
  | Threat {: .compact } | Likelihood | Impact | Risk |
  ```

  It applies to the PDF as well, where there is no minimum and only the padding is at stake - a marker that changed one output and not the other would be the silent half-failure this project keeps meeting.

  Written on a header cell because that is the only thing `attr_list` can reach in a Markdown table, then moved onto the table and removed from the cell, so it styles the table rather than that one column. Any header cell will do, and it combines with `width`.

  Opt-in on purpose: a table that reads well at its default keeps it, and a table changing shape because a column was added is the kind of surprise worth avoiding.

- **Fixed:** setting a column width no longer changes how the whole table looks (#490).

Two tables on a page looked like different components, and the only difference in the markup was that one asked for a column width:

```
| **RAID** {: width="10%" } | **Description** | **Action** |
```

`prodockit.tables` adds `prodockit-table-sized` to a table with a width, and Zensical scopes its entire table style to `table:not([class])` - so the class took the table out of all of it, not just the width control. The stylesheet rebuilt the gap with the *PDF's* look: a full grey grid at body text size, against the website's outer border, row rules and `.64rem`.

It rebuilds the theme's own appearance now, so a sized table is indistinguishable from an ordinary one - confirmed by comparing the computed style of both on the same page. The border colour was the part that would not have fixed itself: `--md-typeset-table-color` follows the colour scheme and the hard-coded `#555555` did not, so a sized table was wrong in dark mode by construction.

The PDF keeps its full grid, which suits print.

- **Fixed:** a directory tree is indented by one number, and set tighter (#486).

  Each level stepped in 98px on the published page, against the 30px `--tree-indent` asks for. Two thirds of that was accidental: the theme's own list margins were never reset, and a row's inset - the stub, and the hanging indent that keeps a wrapped description clear of its icon - accumulated into every level below it, because a child listing lives inside its parent's `<li>` and so starts from that `<li>`'s text rather than from its name.

  Subtracting a row's inset also settles where a rail hangs from. It now drops from under the icon above it at every depth; before, the top level stepped differently from the rest, because a top-level row carries only the hanging indent while every row below it also carries the stub, so a rule written for one was wrong for the other.

  `--tree-row` goes from 1.9em to 1.45em in the same pass: a tree is a dense index rather than body copy, and 1.9em left it looking airier than the prose beside it. The same 44-entry listing is a fifth shorter, and a wrapped description still separates clearly from the next entry.

  Both are now measured rather than reviewed - one test that every level steps by exactly one indent, another that a rail lands under the icon above it. The old stylesheet fails both.

- **Fixed:** a captioned figure is no longer narrower than the text around it (#485).

  An image asking for `width="100%"` came out 410pt in a 470pt column - short of the margin on both sides, while the paragraph directly above it ran the full

width. On the page it read as a picture floating inside a frame that never reaches the edges.

HTML's own default for `<figure>` is `margin: 1em 40px`. No website shows it, because every theme resets it; nothing here did, so the same markdown filled the column on the site and was inset in the PDF. 40px a side is 30pt a side, and 470 − 60 = 410. Measured in one document, captioned against uncaptioned:

```
with a caption      409.9pt
without a caption   469.9pt
```

The horizontal margin goes and the vertical stays, since the space above and below a figure is wanted. This reaches captioned tables too - a table caption is a `<figure>` - which is the same bug wearing different content, so both are tested.

- **Fixed:** the PDF build no longer announces a stage it does not run (#482).

  Every build ended with `[4/4] Rotating landscape pages` and rotated nothing. #469 replaced that post-processing step with a real landscape page box and deleted its module, but the stage title and its announcement stayed - and the announcement was the last statement before cleanup, so a three-stage build called itself four:

  ```
  [1/3] Preparing pages
  [2/3] Assembling the document
  [3/3] Building the PDF
  ```

  The count was self-consistent, which is why nothing caught it: the list of titles and the number announced agreed with each other, and only disagreed with what the build did. What separates a real stage from a leftover is whether anything happens afterwards, so that is what the new test measures - at the moment each stage is announced, the finished PDF must not already be on disk.

- **Fixed:** an image is no longer drawn off the edge of the paper (#480).

  The website holds an image to its column with the theme's own `img { max-width: 100% }`. The PDF stylesheet had no equivalent and nothing else clamped one, so a 1600px screenshot came out 1200pt wide on a 595pt page - most of it past the trim edge. Anything wider than about 627px overflowed, which is most screenshots taken on a modern display.

  The reported case, `{ width="50%" }` on an image, turned out to work: measured end to end through `prodockit pdf`, it renders at 235pt - half the 470pt text column - at every intrinsic size tried, with and without the `.screenshot`/`.pdf-only` classes. Both that and the overflow are now held

by tests, since the two look alike from a reader's chair: an image that ignores a width and one that runs off the page are both "the picture is the wrong size".

- **Fixed:** a hung CI job now fails in minutes rather than hours (#478).

  Three runs sat 35 minutes on `apt-get`, a step that takes nine to thirty-eight seconds. It was not the command: three other jobs in the same run ran the same step and finished, and GitHub reported all systems operational. Cancelling and re-dispatching cleared it.

  No job set `timeout-minutes`, so each inherited the six-hour default. A stall that costs seconds in the good case was free to hold a runner, a required check and a pull request for a working day - and to stay silent doing it, because a job that is running has not failed. Every job now carries a limit sized above its measured slowest run, and the `apt-get` steps carry one of their own so the failure names the stall rather than the job.

  The stall itself has a cause worth naming. The runner image lists `azure.archive.ubuntu.com` first and `https://archive.ubuntu.com` as a fallback; when the first is dead the fallback works, but apt waits out a 120-second connect timeout per source before reaching it. The steps now set ten seconds, well above a healthy mirror's response. Raising the retry count instead - tried first - made it worse: it re-asks the dead mirror rather than moving on.

  The scheduled drift check gets the widest margin relative to its work: it runs unattended, and the run nobody is watching is the one that must not hang.

- **Fixed:** the test suite no longer asks the real internet (#476).

  CI failed twice on a documentation-only branch and passed on a re-run with nothing changed. The bootstrap fixture replaces the command runner so stages answer from a table, and its docstring says that is so a test never goes to the network. That stopped being true when the site and Pages probes moved from launching `curl` to asking a URL through `fetch`: the fixture does not replace `fetch`, so two stages per test went to `github.io`, `api.github.com` or `pages.surrey.ac.uk` for real. Whether they answered decided whether "all 23 stages are set up" held.

  The fixture describes both seams now, `fetch` defaulting to a host that cannot be reached - the same choice `FakeRunner` makes in answering `127 not found`, so a test that has not said what a host does is not quietly given a working one.

  Three tests were passing for the wrong reason and are rewritten: they set `curl` runner keys nothing has read since the probes changed, so they no longer measured what their names claim. A fourth helper was dead code.

  The `sync-repo` tests made a live request each to decide badge visibility.

Nothing turned on the answer - stubbing it two ways gives the same passes - but it is a request per test that fails where there is no egress, so it is stubbed too.

Verified by running the suite with off-machine sockets refused: 1200 pass, the same as with them open. - **Docs:** `prodockit.steps` has a page of its own (#471).

It was the only user-facing extension without one - used once, in the bootstrap quick start, and otherwise documented only in its own docstring. That docstring also pointed at `docs/devcons/steps.md`, which has never existed, so the one signpost to the stylesheet a project is meant to copy led nowhere.

The new page covers the syntax, the `start` option, `attrs`, and the two traps the stylesheet exists to avoid - WeasyPrint ignoring `<ol start>`, and `em` resolving differently inside the number than in the connector.

Heading ids on both extension pages are now qualified (`#steps-writing-one`, `#tree-writing-one`), so two pages can share a heading name without one silently losing its anchor.

The tree page now ends with prodockit's own layout as a worked example - 44 entries, each checked to exist - so the extension is demonstrated on something real rather than on a fixture.

37.5 0.37.0 (2026-08-18)

- **Added:** `prodockit.tree` - a directory listing that looks like one (#379).

```
/// tree
docs/ - the documentation source tree
  index.md - the cover page
  stylesheets/ - CSS for both outputs
zensical.toml - project configuration
///
```

Indentation is the structure, a trailing `/` marks a directory, and `-` starts an optional description. Nothing else is typed, so the icon and the emphasis cannot disagree with what an entry actually is - which the hand-written list this replaces had no way to prevent.

Icons come from the project's own set: the block emits a shortcode and whatever icon extension the project already uses renders it, Lucide's folder and file by default. `directory_icon` / `file_icon` name others. Emitting SVG directly would have baked one project's icons into every document.

Ragged indentation is refused rather than guessed at. A listing is read for its

shape, so an entry attached to the wrong parent is a diagram that is wrong and looks right.

- **Fixed:** an inline SVG icon is no longer clipped in the PDF.

  `prodockit.pdf.html` encodes every inline `<svg>` to a data URI for Pandoc, and serialised it straight from BeautifulSoup - which parsed the page as HTML and lowercased `viewBox` to `viewbox`. SVG attribute names are case-sensitive, so the icon lost the coordinate system it scales into and WeasyPrint drew it at native size, clipped by its own box: a folder icon with its right-hand side sliced off.

  This affects every icon in every prodockit PDF, not only trees.

- **Changed:** `prodockit-table-rotated` is now `landscape-page`, and it works for any content (#469).

  The old name described a table and a rotation, and it is neither. The block gets its own landscape *page*, and what goes in it can be a table, a diagram, an image or anything else too wide for portrait.

  ```
  <div class="landscape-page" markdown="1">
  ```

- **Fixed:** that page is displayed landscape (#469).

  The block has always been laid out on a landscape *page box* - that is what makes its pagination and repeating header rows work, where a CSS transform did not. The finished page was then given the PDF's own per-page rotation flag, and a reader honours that by displaying a landscape page as **portrait**, with the content sideways. So the page a reader saw was portrait.

  Measured on the finished file, before and after:

  ```
  before   page 2: displayed 595x842 (portrait)   /Rotate=270
  box=842x595
  after    page 2: displayed 842x595 (landscape)  /Rotate=0
  box=842x595
  ```

  The box was landscape the whole time; only the display flag disagreed. The flag is gone, along with `prodockit.pdf.rotate` and the alternating recto/verso rotation `pdf_double_sided` applied to these pages - so the class carries the behaviour on its own, with nothing to configure. A document mixing orientations prints without special handling: a reader rotates each page to fit the paper by itself.

  Content longer than one page carries on across further landscape pages. Confirmed by building one: a 90-row table produced five landscape pages, each repeating its header row, with a diagram taking a landscape page of its own and the portrait document either side untouched.

37.6 0.36.4 (2026-08-18)

- **Changed:** the project stage says which address it is using, and where the real one is shown (#441).

  A GitLab group keeps its Name and its URL as separate fields, and renaming it changes only the Name. A group reading `assessment-commtest-2026` in the breadcrumb went on serving git at `comm058-2026`, so every derived URL missed - while the host said only `could not be found or you don't have permission to view it`, the same sentence it uses for a project that does not exist and for one you cannot see.

  Detected and reported rather than worked around. The group's real path cannot be read from here without credentials, so guessing a second address would replace a visible failure with a silent wrong answer. Instead the stage names the address it is using, says to read the browser's address bar rather than the breadcrumb, and shows a worked example of the two disagreeing.

  Per host, because both have the split under different names: GitLab's Name against its URL, GitHub's organisation display name against the part after `github.com/` - and a personal account, where there is no split at all.

- **Changed:** the site and Pages probes ask the URL directly instead of running `curl` (#449).

  Both stages made an HTTP request by starting another program, and that cost three fixes which were never really about curl: it arrived four stages after the first check that wanted it (#374), it was told to write to `/dev/null` on a platform without one (#443), and a `curl` typed at a PowerShell prompt resolved to `Invoke-WebRequest` and looked like an absent program, which sent a diagnosis the wrong way for an afternoon.

  `urllib` needs nothing installed, on any platform, at any point in the run - which also dissolves the ordering problem #374 documented, since there is no longer anything to install first. The `/dev/null` class of bug is gone by construction: nothing writes a response body anywhere.

  Two behaviours were kept deliberately and are covered against a real server rather than a mock. Redirects are **not** followed, because a `302` is how a login-walled site is recognised as published - a Surrey Pages URL answers `302` to GitLab's own OAuth consent page, and following it would report the consent page's `200` and call that site publicly readable. And "could not establish" stays distinguishable from "not there": a refused connection, a DNS failure or a timeout returns `None`, which is not a status.

  Tests describe a host through a `fetch` seam on `Context`, the same way they already describe a filesystem through `exists`.

- **Fixed:** the Ubuntu VS Code install no longer stops for a dialog (#428).

The `.deb` asks a debconf question part-way through installing:

```
Add Microsoft apt repository for Visual Studio Code?  <Yes>
<No>
```

`apt install -y` does not answer it - `-y` agrees to apt's own questions, not to a package's debconf ones. And bootstrap captures its subprocesses, so the dialog was invisible: the run simply stopped, looking no different from slow work.

The answer is now preseeded *before* the install, so it is part of what the reader approves rather than something they meet alone. Answered "yes" deliberately: that repository is how VS Code then updates through apt, and declining would pin them to whichever build the `.deb` happened to be.

The answer is written to a file by Python and read by `debconf-set-selections`, rather than piped into it. The pipe is the problem, not the echo: `echo ... | sudo debconf-set-selections` puts `sudo` inside a shell, and a sudo timestamp expiring there prompts where nobody is watching - the same invisible wait this fix exists to end, reintroduced by the fix itself.

- **Fixed:** a `.pdf-only` image is centred in the PDF (#462).

  Sizing an image differently for each medium - `.web-only` at one width, `.pdf-only` at another - left the PDF copy hard against the left edge, under its own centred caption.

  `.pdf-only` sets `display: block` so that a website's `display: none` is undone. That also took the image out of the one mechanism that positions it: `figure { text-align: center }` moves inline content, and a block box is placed by its margins instead. Measured inside the figure at 0.00pt left against 163.96pt right, where the same image without the class sits at 80.98/82.98.

  Images are exempted from that `display: block` rather than centred with `margin: auto`, which reads like the obvious answer and does nothing: WeasyPrint leaves a block-level replaced element at the left edge even with auto margins on both sides. Inline also keeps one centring mechanism for every image instead of two that have to agree.

- **Fixed:** the installation requirements say what the project actually declares (#372).

  The table had drifted from `pyproject.toml`: `Markdown` was recorded at `⩾ 3.4` long after the real floor moved to **3.10.3**, `zensical` carried no version at all, and `pymdown-extensions` was missing entirely - a dependency `prodockit.pdf` matches class shapes from. None of that breaks a build; all of it sends a reader to install the wrong thing.

`weasyprint` and `pandoc` were also filed together as "external binaries". They are not the same kind of thing: weasyprint is a `pip install` away and simply is not a dependency of prodockit, while pandoc genuinely has no Python package. A reader treating them alike goes looking for one that does not exist, or misses one that does. They now sit in a table of their own that says which is which, alongside the Node major version, the browser Mermaid renders through, and pandoc's floor and pinned release.

The citation style is documented too, as a download rather than an install: pandoc resolves it from the working directory and every CI script fetches it before building. It is deliberately not vendored - third-party content with its own licence and release cadence, which a committed copy would let go stale while every build kept succeeding - and `.gitignore` now keeps a local copy out of commits.

A test now holds the table to `pyproject.toml` - every declared dependency documented, every floor matching, and the versions `prodockit bootstrap` enforces stated - because this drift was invisible for months and nothing else would have caught it.

- **Fixed:** unassessed work is no longer asked for a course code (#458).

  The Surrey questions asked for the course code one question *before* learning whether the work was assessed at all - so an unassessed run answered it and the answer was thrown away. Neither `namespace_for()` nor `project_name_for()` runs on that path; it asks for the namespace and the repository name directly instead.

  "Is this an assessed assignment?" now comes first, and the course code is asked only where it is used.

  The two paths necessarily total different numbers once they diverge: assessed asks three more (course, stage, year) and stays at seven, unassessed asks two more (namespace, repository) and drops to six. The change of total is announced where it happens, the same way the host question already announces the eight-to-seven switch when Surrey's GitLab is chosen - a reader who has just seen `4/7` is owed the reason the next line reads `5/6`.

37.7 0.36.3 (2026-08-17)

- **Changed:** `prodockit pins` watches `prodockit` itself (prodockit-template#173).

  Its absence from the managed set had exactly the consequence the set exists to prevent: `prodockit-template` pinned `prodockit==0.35.0` and drifted two releases behind with nothing noticing, because the one command that looks at pins was not looking at this one. Moving it needed `-p prodockit` typed by hand, which is the step nobody remembers to take.

Adding it found a second case immediately: `prodockit-userguide` pins `prodockit[index]==0.21.0`, fifteen releases behind.

Safe in prodockit's own repository, where the name is the project's identity rather than a dependency: the pattern requires a version operator after the name, so `name = "prodockit"` and the adjacent `version` are not declarations. Without that, the command would offer to rewrite the release number of the package being built - which is why there is now a test saying so.

- **Changed:** pandoc is pinned on Windows, and a local pandoc that differs from the builds' is named (#454).

  Only Ubuntu pinned it. macOS took `brew install pandoc` and Windows took whatever winget was serving - so a machine bootstrap had just set up ran 3.10.2 while every build in this family pins 3.10.1, and nothing said so.

  That matters because pandoc is a *rendering* input. #207 was code blocks coming out as justified prose on an older major, and `limitations.md` records 3.1.3 accepting markup 3.10 does not. A student writes on one pandoc, CI publishes on another, and the PDF they checked is not the one that gets marked - silently, because both builds succeed.

  Windows now installs `--version {PANDOC_VERSION}`, matching what Ubuntu has always done. macOS cannot: Homebrew has no way to install an old pandoc. So the check *says* when the local version differs rather than failing - a failing status there would be one no reader could ever clear, which is the stuck-stage failure #443 and #451 were both about. A pandoc too old to render correctly still fails, because that one is fixable.

37.8 0.36.2 (2026-08-17)

- **Fixed:** Pandoc and Node are no longer reported missing when they are installed (#450).

  Both checks ran a bare `pandoc` / `node`, so they saw only this process's PATH. A `winget install` sets the *machine's* PATH and a running process never receives it - so the check failed on precisely the machine that had just installed the software, while git and VS Code, two stages above in the same listing, reported themselves found by full path.

  Not merely cosmetic: the stage then offers to install what is already there, winget answers "already up to date", and the check fails again, so the run cannot move on.

- **Fixed:** `sync-repo` finds git the way the stages do (#451).

  It invoked git by bare name and died on the same stale PATH:

```
Error: could not run git: [WinError 2] The system cannot find
the file specified
```

The same applied to the Jinja macros and the source bundle. All of them now resolve through one list of install locations, shared with the bootstrap stages - two lists would be two answers about one machine, which is how these drifted apart in the first place.

- **Fixed:** a sync check that could not run is no longer reported as a difference (#451).

  `sync-repo --check` exits non-zero both for "there is a difference" and for "I could not look", and the stage said *"origin is right, but the project config still needs syncing"* to both - naming a cause nobody had established and sending the reader to run something that would not have fixed it. It now says what actually stopped the check.

- **Fixed:** the site stage speaks about the host the reader is actually on (#444).

  Every instruction on that stage was a literal describing GitHub, so a GitLab reader was sent looking for a gear beside 'About', a 'Use your GitHub Pages website' tick-box and a Settings > Pages - none of which exist there.

  One of them was worse than merely wrong. It said the site "will be public" and that "a private repository does not make a private site", which is true of GitHub and false of GitLab: a private project's site stays behind the instance's own sign-in. An anonymous request to a published Surrey site answers `302` to GitLab's OAuth consent page. So a student was told their drafts were readable by anyone with the link, contradicting what the project stage had told them about the same project.

  Carried as per-host values rather than a branch in the stage, which is how `ssh_keys_steps` and `project_visibility` already work - the model's own note is that a difference by host belongs as "a value rather than a branch in the stage".

37.9 0.36.1 (2026-08-17)

- Zensical pinned to **0.0.55** (from 0.0.53), after building the site and the PDF under both and diffing the results.

  The PDF is **byte-identical** - 1,469,185 bytes either side. The site changes 23 HTML files, each by exactly two lines: the `generator` meta tag and the JavaScript bundle's hashed filename. No page differs by more than that, and the stylesheets, workers and licence files are byte-identical - so unlike 0.0.52, which redrew the GitHub brand icon across every page, nothing rendered changes here at all.

The bundle grew 624 bytes (165,840 → 166,464), which is the `ui` v0.0.25 bump in 0.0.54.

Upstream between the two: peak memory cut 8-10x, a `webbrowser` CVE dependency bump, `mike` defaults corrected, an autorefs false-positive fixed, code-span detection fixed in the reference extractor, and surrounding whitespace now ignored in inline link targets. Three of those touch Markdown and reference handling and so could have changed the output; measurably they did not.

The coupling pass in Zensical coupling was run rather than assumed: full suite green (1157), the integration tests that exercise the per-page render context green (70), the `zensical/__init__.py` export surface unchanged, and the count of unresolved `??` cross-references identical either side.

- **Fixed:** the first push no longer forces, so a protected branch accepts it (#442).

  A repository created with "initialize with a README" holds a commit this project's history does not, so an ordinary push is rejected. That was answered with `--force-with-lease`, which GitLab refuses outright on a protected branch - lease or no lease, the rule is about the operation:

  ```
  remote: GitLab: You are not allowed to force push code to a
  protected
  branch on this project.
  ```

  The host's commit is now merged in with `-s ours`, which takes this project's tree entire and leaves the README's behind - the same result the force push had, reached by a fast-forward. On an assessed repository the protection is the point and is rarely the student's to switch off.

  It is also the safer half of the trade: a lease fails only if the remote moved past the commit it names, while an ordinary push fails if the remote moved at all, and cannot destroy anything even in principle.

- **Fixed:** the site probe can run on Windows (#443).

  It discarded the response body into `/dev/null`, which Windows has not got. curl does not read that as a device - it tries to create the file, fails, and exits 23 - so the probe reported "could not check ... the probe did not run" about a site that was serving, on every retry, for as long as anyone kept answering yes. `NUL` is the equivalent there.

  Reporting could-not-establish rather than "missing" is what kept this from being a wrong answer. It is still a state nothing could leave.

- **Changed:** the project stage quotes what the host said (#439).

```
11  MISS  Your own project on the host - nothing visible at
          git@gitlab.surrey.ac.uk:assessment-commtest-2026/
report-... -
          gitlab.surrey.ac.uk said: The project you were looking
for could
          not be found or you don't have permission to view it.
```

It used to say "nothing visible at ..." and stop there, so a reader who had just created the project saw the same eleven words on every retry, with nothing new to work from.

The host's own reply distinguishes what the stage cannot: a project at another path, a group you cannot see into, and a key the host will not accept need three different things from you. Git's own wrapper lines are left out - "Could not read from remote repository" is not something anyone can act on - and a host that says nothing gets no words put in its mouth.

37.10 0.36.0 (2026-08-17)

- **Fixed:** the ssh-agent step ended the run without looking (#435).

  0.34.0 stopped that step looping, but overshot: the moment you said the service was started, the run ended and asked you to start again.

  A started service usually *is* visible to the next command that looks - `ssh-add` opens the agent's pipe afresh each time it runs - so the check is given its chance, and a run that can carry on carries on. Only where it still cannot be seen does the run end, because asking again would put the same question to the same unchanged answer.

  The message no longer says to open a new window either. There was never anything a new window would do that running it again would not.

- **Changed:** the assessment question comes before the year, and unassessed work is asked for its namespace (#437).

  The year question explains itself in terms of SRA and LSA, and nothing before it had said what those are - the question that introduces them, and lists them, came afterwards. They are the other way round now, so the year question refers back to something already on screen.

  The stage question is numbered too. It used to arrive after "is this assessed?" without a number of its own, so a reader counting down from six met a question that was not in the count. Seven either way now, and every one of them numbered.

  Unassessed work is no longer asked for a year it has no use for, nor for a stage it has not got. It is asked for its namespace and its repository name instead, offered as `<your login>` and `report-<your login>` - a project

group and a name of your own are perfectly ordinary answers, and a default is only an offer.

- **Changed:** the Surrey questions are worded as they were asked for (#420).

  The login question had lost its example, which was the part that made it answerable - a reader logs in with `ab1234@surrey.ac.uk`, so being asked for "the six-character ID" is a question about something they may never have typed. The year question had lost the rule for a resit, and gained a different one of mine.

  Both now read as specified: the login question shows the address and the six characters to take from it, and the year question says an SRA or LSA should be the year *prior* to the year the retake is being assessed.

- **Fixed:** a first run printed the stage list over the details it had just told you to note down (#433).

  The namespace and repository name are the only two values a reader has to carry from this command to a browser, and twenty-three stage lines printed after them scrolled both away. `--configure` already stopped after saving; a first run answered the same questions by a different route and carried on.

  It stops there too, and says what to run to see the stages. Filling a single missing value still reports, because nothing has been announced that a stage list would bury.

37.11 0.35.1 (2026-08-17)

- **Fixed:** a first run did not take Surrey's shorter configure path (#430).

  ```
  Some details are not set yet: host, full_name, email, ...
  1/8 The git host your project lives on [gitlab.surrey.ac.uk]:
  3/8 The email address used for your gitlab.surrey.ac.uk login
  []:
  ```

  Eight questions, asking for the email 0.35.0 exists to derive. The shorter path was chosen only when `--configure` asked for everything, and a first run does not arrive that way: nothing is set, so bootstrap offers to fill the gaps and asks for the fields by name.

  Nothing set at all is the configure arriving by a different door, not a repair, and is treated as one. Coming back later to correct a single value still asks for that value.

37.12 0.35.0 (2026-08-17)

- **Changed:** VS Code is driven from inside the application when `code` is not on PATH (#424).

  On macOS the application and the command are two different installs. The app is dragged to Applications; `code` arrives only when somebody runs "Shell Command: Install `code` command in PATH" from inside it - which readers routinely have not, while the binary sat in the app bundle the whole time.

  It is found there now, so the stages that drive VS Code work without it. The stage still says `code` is not on PATH, and how to add it, because that is a convenience worth having in your own terminal - it is no longer a thing that stops the run.

  Windows keeps its own wording, which is true there and would be false here: an app bundle is not on `PATH`, and a new terminal will not change that.

- **Fixed:** a repository created with a README was reported as pushed when nothing had been pushed (#423).

  Ticking "initialize this repository with a README" makes a commit your project's history does not contain. The stage asked only whether the remote had *anything* on it, so it answered `ok` about a project that had never been pushed - and the site stage then found nothing published, with nothing on screen to connect the two.

  It now compares the commit here with the commit there. Where the only thing on the host is that README, the push replaces it, pinned to exactly the commit that was inspected - so anything arriving in between fails the push rather than being overwritten. Where the host has anything else at all, nothing is pushed over it and the stage says to go and look.

- **Fixed:** the bootstrap page described eighteen stages when there were twenty-three (#413).

  Five stages missing from the table, the numbering shifted under the ones that remained, and a sentence about which stages are platform-independent naming numbers that had moved. A reader on a finished setup was told `All 23 stages are set up.` by a page listing eighteen, with no way to tell which was wrong.

  The table now names every stage, and a test holds it against the stage list itself - count, order and wording - so it cannot drift again. The count is no longer written into the prose: a number in a sentence goes stale in silence, which is how this lasted five releases.

- **Added:** a shorter `--configure` for Surrey's GitLab (#420).

```
1/8 The git host your project lives on [gitlab.surrey.ac.uk]:

  gitlab.surrey.ac.uk fills in the rest from your login ID and
course
  code, so this is 6 questions rather than 8.

2/6 Your full name, as it should appear on commits:
3/6 The six-character ID you log in to Surrey with, e.g.
`ab1234`:
4/6 Your course code, e.g. `comm058`:
5/6 What year does the module start in? [2026]:
6/6 Is this an assessed assignment? [Y/n]:
```

Your email, GitLab username, group and repository name all follow from those, so they are no longer asked for. Assessed work goes to `assessment-<course>-<year>`, with `-sra` / `-lsa` after the year for a resit; unassessed work stays in your own namespace, where no year applies.

The year is the one the module *starts* in, offered as this one. Both the awkward cases are in the question rather than in a handbook: a semester 2 module should be the year after the Christmas break, and for SRA and LSA the year should be the year prior to the year the retake is being assessed.

The repository, and the folder it lands in, are named `report-<course>-<year>-<login>`, with `-sra` / `-lsa` on the end for a resit. The name carries the year even for unassessed work, where the namespace does not: those repositories sit side by side in one personal namespace, and two years of the same module - or a first attempt and its resit - would otherwise be two repositories with one name between them. The repository is `report-<course>-<login>`, and the run ends by telling you both, since you have to find them on a website afterwards.

Every free-text answer removed is one fewer chance to type a namespace that does not exist and hear about it six stages later, from a host that says only "not found".

Nothing here applies to github.com or gitlab.com, where these rules are simply untrue - those keep the eight questions.

37.13 0.34.0 (2026-08-16)

- **Added:** `prodockit.steps`, numbered steps a reader works through in order (#378).

```
/// steps
    start: 9
```

```
//// step | Load the key into the agent
```bash
ssh-add --apple-use-keychain ~/.ssh/id_ed25519_gitlab
```

////

/// ```

A procedure is not a list of facts. Each step is a thing to stop and do, so it gets a number to find your place by, room for a command and its explanation, and a line joining one step to the next.

Built on pymdownx's Blocks API - the machinery Material's own admonitions and tabs use - so `attrs` works as it does everywhere else, and a step's body holds paragraphs, code or a table without indentation arithmetic.

`start` exists because a long procedure is often split across sections. It is written into the HTML twice, deliberately: a browser reads `start`, and WeasyPrint ignores it entirely and numbers from 1. Emitting both from one setting is the reason this is an extension rather than a documented HTML snippet - a pair maintained by hand drifts, and the failure is silent and PDF-only.

The bootstrap page now opens with a six-step quick start written in it, which is what exercises it.

`pymdown-extensions` is now a dependency. It was deliberately not one before, on the grounds that prodockit never imported it; this imports it, and python-markdown constructs an extension by importing its module, so there is no lazy route.

- **Fixed:** a push the host refused left the stage unable to try again (#414).

```
[main (root-commit) 5e98636] Initial commit
...
remote: You are not allowed to push code to this project.
failed: exit status 128 - see the output above
```

The commit was made and the push declined, which leaves the project committed here and empty there. On the next run `git status` is clean, so `git commit` exited 1 with nothing to commit and the run stopped *before* the push - failing for a reason that was neither true nor the obstacle. The commit is now run only when there is something to commit.

The refusal is named, too. Bootstrap runs those commands with the terminal attached, so their output goes to you and not to it, and `exit status 128` was all it could say - after sixty-eight `create mode` lines had scrolled the real sentence off the screen. From that one state, and no other, it asks the host
```

whether a push would be accepted, and says so: the key is fine, the account is not allowed to write there.

37.14 0.33.0 (2026-08-16)

- **Fixed:** the VS Code extensions stage said VS Code might not be installed, having just installed four extensions through it (#410).

  ```
  Extension 'ms-python.python' v2026.4.0 was successfully
  installed.
  ...
  ran, but still not right: could not list extensions - is VS Code
  installed?
  ```

  The plan found `code.cmd` where the installer puts it; the check asked for a bare `code`, which on Windows cannot run at all - `CreateProcess` appends `.exe` and nothing else. So the stage could never pass there, whatever was installed, and the run stopped on it because later stages depend on it.

  Third instance of one shape - after git (#390) and npm (#405) - so there is now a test that no check asks for a tool by a name it knows how to resolve.

- **Added:** `prodockit pdf` says which stage it is on, and how long the build took (#375).

  ```
  Building PDF from zensical.toml...
    [1/6] Preparing pages
    [2/6] Assembling the document
    [3/6] Building the PDF
    [4/6] Collecting index entries
    [5/6] Rebuilding with page numbers
    [6/6] Rotating landscape pages
  Wrote docs/site_documentation.pdf in 1m 35s
  ```

  One line at the start and one at the end left minutes of silence in between, and a silent terminal is indistinguishable from a hung one.

  A build with an index has two more stages than one without: the page numbers an index needs do not exist until the document has been laid out, so the whole thing is built, read, and built again. That is worth seeing rather than wondering about.

37.15 0.32.1 (2026-08-16)

- **Fixed:** the Node stage failed with `npm: not found` having just installed Node (#405).

Two things were true at once. A plan is written before any of it runs, so npm was resolved while Node was still absent and fell back to the bare name - and a bare `npm` can never run on Windows, because `CreateProcess` appends `.exe` and npm is a `.cmd`.

Refreshing `PATH` could not help, because the problem was the name rather than the path. Names are now resolved as each command is about to run, which is the only point at which the answer can be right.

- **Changed:** your answers live beside the project, as `.pdk-bootstrap.toml` (#373).

  There was one config per user, so setting up a second project overwrote the answers for the first - its namespace, its name, the directory it lives in - and the original could not be re-checked without answering everything again.

  One file per directory means one per project. The old `~/.config/prodockit/bootstrap.toml` is still read when a directory has no file of its own, so nothing has to be moved and a setup already answered keeps working.

  The file holds your name, email and username, and the first push commits with `git add -A` - so where the directory is a repository, it is added to `.gitignore`.

- **Changed:** the closing line says how to apply the setup, not only how to look at it (#376).

  ```
  14 of 23 stages need work.
  Run with --dry-run to see the exact commands that would fix
  them.
  ```

  `--apply` is the one a reader has come for, and it was the one option not mentioned. Both ends of a run had the same gap: after a dry run, nothing said how to run what had just been printed either.

- **Changed:** the run says which prodockit is doing the work (#399).

  ```
    Running:  pdk from C:\users\buckwem\GitHub\.venv\Lib\site-
  packages\prodockit
  ```

  A report arrived headed with one version, showing commands that version had not contained since the release before - an older install doing the work, in a window that had never been reopened. The version alone could not show that. The path can, and it is the first thing to check when a run does something the source says it cannot.

- **Fixed:** git installed a moment ago was reported as not installed (#390).

```
[2/22] Git, installed and configured
        git is not installed
...
Found an existing package already installed.
```

The installer puts git on the *machine's* `PATH`, and `PATH` is read when a process starts - so a window that has not been reopened since cannot see it. VS Code has had this answer since 0.27: find the executable where the installer puts it and use it by its full path.

Git now shares it, and every stage that runs git uses the same answer. Fixing only the check would have moved the failure a dozen stages down to the clone.

- **Fixed:** the ssh-agent step asked again after you had answered it (#397).

```
Have you started the ssh-agent service? (yes/no): yes
not there yet - no ssh agent is running
Try again? [Y/n]:
```

On Windows the service is started from a separate Administrator window, and the run in your own window cannot see that happen. So it re-checked something that could not have changed, reported it missing, and asked again - which reads as the tool ignoring the answer it was just given.

The run now ends there, saying what to type next, and exits zero. Nothing failed: the step was done, and a new run is what it takes to see it.

37.16 0.32.0 (2026-08-16)

- **Fixed:** Windows on ARM stopped at the Pandoc stage, having just installed MSYS2 successfully (#393).

```
Successfully installed
MSYS2 is not at C:\msys64 - install it there, or run ...
failed: exit status 1 - see the output above
```

Two assumptions, both about the machine rather than about this project. `C:\msys64` is the installer's default, not a promise; and an arm64 Windows gets the arm64 build of MSYS2, which has no MINGW64 environment at all - its native one is CLANGARM64, with different package names and a different DLL directory.

Neither can be known from here, so both are settled on the machine when the step runs: several locations are searched and the one found is used, the

environment follows the processor architecture, and the directory added to `PATH` follows the environment. When nothing is found it says where it looked.

- **Added:** Surrey's GitLab Pages address is derived rather than asked for (#392).

```
Note: cannot derive a published URL for GitLab; site_url left
unchanged
```

printed for an instance whose layout was perfectly well known. `https://gitlab.surrey.ac.uk/mb0105/report` publishes at `https://mb0105.pages.surrey.ac.uk/report/`, so `sync-repo` writes that, and the site stage has an address to check.

Only instances somebody has actually run against are derived, and they are listed. GitLab's default layout is `<namespace>.pages.<instance domain>/<project>`, but the instance domain is an administrator's setting that nothing in a remote URL reveals - a confidently wrong canonical URL is worse than none, so every other instance still declines and uses `pages_base`.

The site stage now reads a login wall as proof rather than absence. A university instance publishes behind its own sign-in, and "is not answering yet" of a site that is plainly up would leave every Surrey run one stage short for ever.

```
Note: could not tell whether GitLab is public; badges left as
they are
```

is gone too, on hosts where it could never be answered. A self-hosted instance shows a stranger a login page, and nothing turned on the answer: the badges that do come from shields.io, which cannot read that host either. Assumed private, and no longer reported.

- **Added:** the first stage checks that prodockit is running from a virtual environment of its own (#381).

```
 1  MISS  prodockit runs in an environment of its own - running
from
          /usr/bin/python3, which is not a virtual environment
```

There are two environments in a finished setup, and they are easy to confuse: **prodockit's own**, which `pdk bootstrap` runs from, and **the project's** `.venv`, which holds Zensical and everything else in `requirements.txt`. The second is built by the first.

So the first is a prerequisite for the run rather than a step within it, and it is asked first. On a system Python the run used to fail fifteen stages later -

Debian and Ubuntu refuse `pip install` outside a virtual environment and ship `venv` without `ensurepip` besides - in words about the project rather than about the interpreter building it.

Where the missing piece is a package, it is installed. Either way the exact commands for your platform are printed, including the three that put prodockit in an environment of its own. Nothing can repair this in place: a new environment needs a new process, so the steps are shown and the next run confirms them.

The header said "with the Python virtual environment active" without saying which. It now names it. Twenty-three stages.

37.17 0.31.1 (2026-08-16)

- **Fixed:** the GitHub Pages stage reported itself done without having looked (#374).

```
11  ok    Pages switched on - cannot be seen from outside a
private repository
```

Pages had never been switched on. A private repository answers `404` to every anonymous caller, and "cannot be seen" was being printed as though it meant "is set up" - so the one stage on the one host that has to be done by hand was skipped in silence.

It is a finding now, and shows the steps. It still does not claim to have looked: a check that cannot settle itself says so, takes your word once, and leaves the proof to the site check at the end of the run. A project that has already published is recognised from its own site, so a private repository no longer carries a finding it could never clear.

- **Fixed:** three checks reported a look they had not taken.

curl is installed by the Pandoc stage, several stages below the first check that wants it, so on a machine part-way through a setup the probe can simply be missing - and `curl: not found` was reaching you as "cannot be seen from outside a private repository" for Pages, and "is not answering yet" for the site. Both now say the probe did not run.

The stage that looks for your repository no longer reports "is not reachable" when the host has answered. github.com says `Repository not found.` for a repository that is missing *and* for one your key cannot see, so the steps now tell you to look before creating anything: an issued repository carries the permissions that decide who can read your work, and a second one will not have them.

- **Changed:** the manual steps ask you to type `yes` (#374).

`[Y/n]` is answered by pressing Enter, and a reader twelve stages into twenty-two presses it in rhythm - which at a browser step means claiming to have done something they have not. All three now want the word. `no` is a real answer: it leaves the stage outstanding rather than pretending it was done.

- **Fixed:** the configure prompt named github.com whatever host you had answered (#370).

```
5/8 The group, organisation or user the project lives under
    (e.g. comm058-2026, or your own username on github.com) []
```

asked of a setup against gitlab.surrey.ac.uk. The host is the first question so that the rest can be phrased in terms of the answer, and naming a different service reads as a question about a different account somewhere else. It now says your own gitlab.surrey.ac.uk username, or whichever host you gave.

- **Changed:** the stage that looks for your project now says which address it asked about (#377).

```
 7  ok    Where the project comes from - nothing visible at
          git@github.com:mb0105/report-ubuntu-v1.git - the
template
          will be cloned
```

It used to report "no existing repository found", which gave a reader whose project plainly exists nothing to work with. The namespace is one setting shared by every host, so a setup that changes host carries the previous host's namespace to the new one - and the address on screen is what makes that visible.

It no longer claims the repository is absent, either. Both hosts answer a private repository your key cannot see with the same words they use for one that does not exist - github.com says `Repository not found.` either way - so "nothing visible at" is as much as the question can establish.

- **Fixed:** setup stopped on Windows at "Clone pointed at your project", saying prodockit was not found (#371).

```
Commands finished, checking the result...
failed: prodockit: not found
Stopping - later stages depend on this one.
```

Said of a machine where prodockit was installed and running the bootstrap that reported it. Two stages run prodockit commands of their own - `sync-repo` when the clone is repointed, and the MathJax install - and both named `prodockit` bare, leaving the machine to find it a second time. A virtual

environment's scripts are reachable when it launches one; they are not necessarily on the `PATH` a child process inherits.

Both now run through the interpreter already running them, which also settles *which* prodockit: the one doing the work, rather than a different install earlier on `PATH`. `python -m prodockit` works as a command in its own right.

- **Fixed:** a finished setup was told its project needed a decision (#368).

  Every stage done, `prodockit boot` run once more to confirm, and the stage that asks where the project comes from reported `MISS` and put three choices to somebody whose project was already cloned, pushed and published. None of the three could change where the contents had come from: they were on disk.

  It now says where they came from. A clone still on the template keeps the question, because there the decision really is ahead - and an `origin` that cannot be read stays unknown rather than being taken for an answer.

  A check pass on a finished machine now costs three connections to the host rather than four.

- **Added:** `source`, a short name for `source-bundle` (#366).

  The longest name on the list, typed at a prompt while a submission is being checked. `bootstrap` already answered to `boot`; both now come from one table, so a third alias is one line rather than three places to keep in step.

  The long names all stay - they are what the User Guide, the changelog and anything scripted use.

- **Fixed:** the brand logo stayed the template's until something cleared the cache (#364).

  `prodockit sync-repo` rewrites the icon in `zensical.toml`, and a build served from `.cache/` kept showing the old one - readers were clearing it by running `zensical serve` and wondering why it changed.

  The commit-and-push stage builds once with `--clean` before committing. It also means the first push is the first time the project is *proved* to build, rather than that being discovered from a red pipeline minutes later.

37.18 0.31.0 (2026-08-13)

- **Added: gitlab.com is a supported host** (#361).

  Declared since the beginning and refused, because nothing had been run against it. It is covered by tests rather than by a machine - Surrey's own instance has been unreachable, so no GitLab path has been run end to end - and that is worth knowing before relying on it.

A self-hosted instance of either family is still refused. What cannot be guessed for one is where it publishes its Pages and where its API lives, and inventing either would send a reader somewhere that was never going to answer.

- **Changed:** a green stage says why, and the history reset defaults to yes (#356).

  "Where the project comes from" reporting `ok` read the same whether the host had been searched and found empty or never reached at all - the ambiguity behind #344 and #351. It now says `no existing repository found - the template will be cloned`, or `could not reach <host> to look`, and the apply loop prints the detail rather than discarding it.

  The history reset defaults to yes. It deletes only ever the *template's* history - a clone carrying the reader's own is never offered it, and the stage is blocked while the decision is unmade - so defaulting to No left anyone who pressed Enter with the template's commits behind their project. - **Removed: the `gh` / `glab` requirement** (#357).

  Verifying Pages through a host CLI meant installing a tool, authenticating it in a browser, from the right directory, on every machine - four ways to go wrong, each of them hit in testing.

  A public repository reports `has_pages` to any anonymous caller, and the published site answers anonymously even when the repository behind it is private. So Pages is read without a token where that is possible, and where it is not the stage says so and leaves the proof to the site check at the end of the run - which needed no token in the first place.

  The About > Website link is an instruction now rather than an authenticated call: open the repository, click the gear beside 'About', tick 'Use your GitHub Pages website'.

  The Pages stage is GitHub's alone now, too. GitLab configures its own Pages from the CI job, so a GitLab reader was being shown GitHub's steps - an instruction to do nothing - and the metadata URL those steps were checked against was `api.github.com` written into the stage, which a gitlab.com project would have been asked about. Both are fields on the host.

  Nothing that was proven stops being proven; it moves one push later. Bootstrap is 22 stages.

- **Fixed:** bootstrap asked you to sign in with a tool it had not installed yet (#354).

  The stage reported `gh is not installed` and the next line said "Run `gh auth login`". Both halves were in `instructions`, which are printed before a plan's commands - so the install came after the request to use it.

The sign-in is `follow_up` now, which exists for work that only makes sense once the commands have run.

37.19 0.30.1 (2026-08-12)

- **Fixed:** `--apply` trusted checks taken before any stage had run (#351).

  A project with work on the host was cloned over with the template, and the reader was never asked: "Where the project comes from" reported `ok` because it had been checked *before* the SSH stages ran, when the host could not be reached.

  Each stage is checked again when the loop reaches it now. Earlier stages change the machine the later ones are about - that is what a setup tool is - so a single snapshot taken up front was never going to hold. The memo keeps repeats free within a pass and is dropped between stages, which is what makes the second look real.

37.20 0.30.0 (2026-08-12)

- **Fixed: the clone decision is now actually offered**, as a stage between the SSH stages and the clone (#348).

  `--configure` runs before there is an SSH key, so it could never see whether the project existed and had to say it could not look - which left the three-option question unreachable on every first run, the one where it matters most.

  It is asked at the first point the answer is knowable, and the last at which it still matters. Nothing to decide stays `ok` rather than becoming a question: a project that does not exist, or exists and is empty, gets the template and nobody is asked to choose between one thing. A recorded answer is read, not asked again.

  A plan can put a numbered choice now rather than a yes/no. Three paths as three consecutive yes/no questions invites pressing Enter through them, and one of these deletes commits that cannot be recovered - so it has no default.

- **Fixed:** whether a repository holds a *project* was being judged by "has any commits" again. The check for `zensical.toml` and `README.md` was written for #332 and lost when `stages.py` was reset during #339, so it never reached a release. A pass costs four host connections now rather than three, which the test that tracks that number records.

- **Added:** `boot`, a short name for `bootstrap`.

  `pdk boot --apply` is the form this is typed in most, and it is typed repeatedly while a machine is being brought up. Registered rather than

wrapped - one command object under two names - so the two can never take different options or drift in their help. `bootstrap` stays.

37.21 0.29.0 (2026-08-12)

- **Changed:** CI caches the pandoc download, so an outage at the release CDN cannot stop a build.

  It returned `503` for hours one evening and failed every job in the matrix before a single test ran, three times across three branches. The retry added earlier rides out a blip and did not help with that: it fired five times and still gave up.

  Keyed on the pinned version, so a bump fetches afresh and an unchanged pin never touches the network again. The retry stays for the first run on a new key, and for the day the cache is evicted - a cache is a saving, not a guarantee.

  Only affects building this project; nothing in the package changes.

- **Fixed:** a repository bootstrap could not reach was reported as not existing, and a partial run never asked where the project comes from (#344).

  On a fresh machine there is no SSH key until stage 3, so `git ls-remote` fails on authentication - and that was read as the repository being absent. A reader was told their work was not on the host while it was sitting there.

  Only the host saying so counts as absent now. A refused key, an unreachable network, a name that will not resolve: none of them is evidence about the repository, and "cannot tell" says so plainly.

  Separately, `missing_keys` omits `source_url` - blank is a valid answer - so a run filling in a few gaps skipped the question about an existing project and ended without its summary. Both paths ask it now.

- **Added: the host's own command line** is installed and signed in as a stage (#342).

  `gh` for GitHub, `glab` for GitLab - including a self-hosted one, which is still GitLab. It is the only thing that can answer questions no anonymous caller can: whether Pages is switched on, and what the repository's About panel says. It holds a token so bootstrap does not.

  Signed in counts as much as installed - an unauthenticated tool is installed and useless, and calling that done would leave the stages depending on it failing for a reason two stages away. `auth login` opens a browser, so it stays the reader's to run.

- **Added: Pages is a stage of its own**, straight after creating the project (#341).

It was a trailing item on the "create your project" list and was missed twice, at a cost of a red first build whose error names the site rather than the setting. Now it is asked while the reader is still in the browser, and before the push, where missing it breaks the build.

The site stage also fills in the repository's About > Website field, using `gh` for the same reason. GitHub does not set it from Pages, so a project with a perfectly good published site showed no link at all on the page anybody actually lands on. `sync-repo` cannot do it - it reads and writes local files, and this lives on the host.

Confirmed with `gh` where that is installed, since it already holds a token and bootstrap does not have to. There is no tokenless way to ask: the Pages API answers `404` to an anonymous caller even for a *public* repository with Pages enabled, and the published site cannot be fetched until a push has built it. Without `gh` the stage says it could not look, rather than guessing - the site check at the end of the run is the honest test either way.

- **Added: a stage** that makes the first commit and pushes it (#339).

  Everything before it left a working project on one machine and an empty repository on the host - and a reader trying to finish the job from VS Code's "Publish Branch" could not, because that creates a repository and theirs already existed.

  It runs after the local setup stages, so the first commit carries the CSL style, the MathJax bundle and the VS Code settings rather than needing a second commit for them, and before the site check, because the push is what builds the site.

  It waits on the clone and on `origin`: committing into a directory that is not a repository, or pushing to a remote that was never set, cannot be a plan worth running. - **Added:** `pdk`, the same tool under a shorter name.

  Every command here is typed at a prompt, often several times over while a setup is being repaired - `pdk bootstrap --apply` rather than `prodockit bootstrap --apply`. Both names stay, so anything written against `prodockit` keeps working, and the help text says whichever one was typed.

37.22 0.28.1 (2026-08-12)

- **Fixed:** the "your own project" stage could loop for ever, and still asked for a Pages step the workflow now does itself (#336).

  The repository existed, the reader said so, and was told the clone still points at the template - a fact about their machine that creating a repository cannot change. Blocking that stage on the history reset was wrong: the repository lives on the host, and `rm -rf .git` is local, so nothing about creating it is undone by the reset. Only the repoint is, and only that stage is blocked now.

A blocked stage also ends the confirm loop with `waiting` rather than asking again, since no answer can satisfy a stage waiting on another one.

The Settings > Pages instruction is gone. The template's workflow enables Pages itself, so it described work already done. Stage 19 still checks the published site answers - what has gone is the instruction, not the verification.

- **Changed:** the configure questions are numbered, and their text wraps to the terminal instead of at line breaks typed in by hand - which only lined up for one length of project name.

37.23 0.28.0 (2026-08-12)

- **Added: a nineteenth stage** that checks the documentation site is actually published (#333).

  A test rather than a step. The template's workflow enables Pages itself now, so there is nothing for a reader to switch on - and an instruction telling them to anyway is one more thing to rush past, which is how two projects reached a red first build.

  Last of all, because it can only be true once a push has built the site. Fetched anonymously: a Pages site answers to anyone even when the repository behind it is private, which is what makes it checkable without a token at all.

  The `ok` message says the site is **public**, because that is the part readers get wrong: a private repository does not make a private site - only a GitHub Enterprise plan can restrict who reads one - so anything in `docs/` is readable from the moment it builds.

  A host that publishes at no address bootstrap can work out - a self-hosted GitLab - reports `not checked` rather than claiming a site was found.

- **Changed:** where the project comes from is decided at `--configure` time, as a question naming every path (#332).

  ```
  buckwem/report-windows-v1 already exists on github.com and has
  content in it.
  Do you want to:

  1. clone the full repo 'buckwem/report-windows-v1', then leave
  the existing
     git records and sync origin unchanged
  2. clone the full repo 'buckwem/report-windows-v1', then delete
  the existing
     git records and set up a new remote repo
  3. start from the template instead, discarding nothing on the
  host -
     but a first push would then replace what is in buckwem/
  ```

```
report-windows-v1

Select 1, 2 or 3:
```

Both of the first two clone the repository itself: the difference is what becomes of its history and its remote, which is the only part there is to decide. "Existing project or template" framed that wrongly - somebody starting again still wants the contents that are already there.

The template is named rather than implied. It is what happens when nothing else is chosen, and a reader who cannot see it among the options has to infer it from the absence of anything else.

A repository that is empty, or not there at all, is a *message* rather than a question: cloning an empty one would leave no `zensical.toml`, no `requirements.txt` and no `tools/`, so every later stage would fail on the absence. The permissions an issued repository carries are not lost by that - they belong to the repository on the host, and `origin` is pointed at it either way, which the message says so it does not read as the repository being ignored.

No default. One answer deletes commits that cannot be recovered, and none of them is safe enough to be taken by pressing Enter.

The answer is recorded, so the run follows an explicit path rather than re-deriving the decision from what `origin` happens to say - and a rerun does not ask again.

- **Fixed:** the history stage offered to delete a project's real history because `core.fileMode` was unset, and adopting an existing repository happened silently (#332).

  On a clone that already carried its own history, the stage reported wrong only because `core.fileMode` was off - and its plan was still `rm -rf .git`. Its plan is now that one setting and nothing else, and is not marked destructive, because nothing there destroys anything.

  Adopting an existing project is put to the reader, saying what each answer means: cloning brings their work and its history, and the history stage will not offer to delete it; answering no takes the template, whose history that stage then deletes with `git init -b main`.

  The report says which repository was used - `from the template` or `your own project` - so the decision is still visible once the prompt has scrolled away.

- **Fixed:** answering no to the history reset showed the commands anyway (#330).

```
Delete the template's history and start a new repository? [Y/n]: n
```

printed `rm -rf .../.git` and asked again. The answer was collected and discarded - written as a bare acknowledgement, which is fine for "have you uploaded the key?" and not for a deletion with no undo.

That prompt also defaulted to yes on a destructive plan. The rule that this one plan must not default to yes had been applied to the command prompt below it, and not to the question a reader reads first.

The same stage reported `no clone yet` and then offered to delete a `.git` that did not exist, and run `git init` in a directory that did not either. It waits for the clone stage now, the way the two stages after it already do.

37.24 0.27.0 (2026-08-12)

- **Fixed:** a second machine cloned the template over a project that already existed (#327).

  Set up on one machine, pushed, then run on another, bootstrap cloned the *template* - giving the reader template content in a checkout whose `origin` the next stages repointed at their real work. Nothing errored; every stage reported done.

  The project on the host is cloned instead when it already holds commits. A project that exists but is empty - created in the browser, never pushed to - still gets the template, since cloning it would leave nothing to work on. `git ls-remote` tells the two apart on evidence rather than a flag.

  This is also the case where a taught module hands a student a repository that already holds their starting point. Deleting its history would throw that away, so the history-reset stage is not offered for a clone of the reader's own project - now asserted rather than left to fall out of how `origin` happens to compare.

  An explicit `source_url` still wins: detection is a default, not an override.

- **Added: a Documentation badge**, and dropped two that cannot work on a private repository (#326).

  `prodockit sync-repo` kept `site_url` correct in the config while the README - the page a human actually lands on - had no way through to the published site. It leads the badge row now, reporting whether the site is up where shields.io can reach it and linking plainly where it cannot.

  The star and fork badges rendered `Stars: repo not found` on a private repository, which is what `prodockit bootstrap` tells readers to create - so two of three badges were wrong by default. They are emitted only when an anonymous visitor can see the repository, which is exactly the view shields.io has. A visibility check that cannot be answered - offline, a timeout - changes nothing and says so.

- **Fixed:** nothing enabled GitHub Pages, so the first push failed in CI (#324).

  Bootstrap reported every stage done, the initial commit and push both succeeded, and the Documentation workflow then failed with `Get Pages site failed` - which names the site rather than the setting nobody had been asked to switch on.

  Two GitHub-only problems, both now said while the reader is still in the browser rather than left to be found from a failed build:

  - Pages has to be enabled by hand, under Settings > Pages, with Source set to 'GitHub Actions'. GitLab needs no equivalent - its CI job configures its own.
  - The repository stays private but the site built from it does not, which is what GitHub itself warns when Pages is switched on. Said here because drafts and notes in `docs/` are published the moment they build, not when their author decides they are ready.

  Both are `Host` fields rather than branches in the stage, following the rule that a difference between hosts is a value.

37.25 0.26.7 (2026-08-12)

- **Fixed:** the two browser stages could not see what you had just done in the browser (#321).

  A repository created on the host was reported as missing, and "Try again?" repeated the same answer however many times it was accepted:

  ```
  not there yet - git@github.com:you/report.git is not reachable
  Try again? [Y/n]:
  ```

  A regression from the connection-reuse work in 0.26.4. Answers about the host are remembered within a pass and dropped after each applied command - but a browser stage's plan is instructions only, so no command ran, nothing was dropped, and the verification read an answer from before the reader went to the browser.

  Anything remembered is dropped now before every re-check that follows a human step, in both the verification and the retry loop. The saving those memos exist for is unaffected: it comes from repeats within a single pass, none of which have a person acting in between.

37.26 0.26.6 (2026-08-12)

- **Fixed:** creating the SSH key failed on a machine with no `~/.ssh` directory (#318).

`ssh-keygen` does not create the directory it is asked to write into, and the error names the *key* rather than the missing folder:

```
Saving key "C:\Users\you\.ssh\id_ed25519_github" failed:
No such file or directory
```

The stage creates it first now, on all three platforms - Windows is where it bites, since macOS and Linux usually have `~/.ssh` already from some earlier ssh use, but a genuinely fresh machine of any kind has no such directory.

Only when it is absent, so an existing directory keeps whatever permissions its owner chose. One this created gets `700` on macOS/Linux: ssh refuses a key others can read, and the same applies to the directory holding it.

- **Fixed:** accepting a host's key stopped the run, even though the connection had just succeeded (#316).

  `ssh -T` against a git host exits non-zero *even when it works* - there is no shell to give you, so the exit code says nothing at all. The checks have always known this and match on the greeting instead, but the apply loop was still reading the code:

  ```
  Hi buckwem! You've successfully authenticated...
  failed: exit status 1 - see the output above
  ```

  That probe is never fatal now. A genuine rejection is still caught, by the stage's own check re-running it and reading the greeting - which is the one thing that can tell the two apart.

37.27 0.26.5 (2026-08-12)

- **Fixed:** on Windows, a rerun stopped at "Git install failed" when git was already installed (#309).

  winget reports "already installed, no upgrade available" as exit `2316632107` (`0x8A15002B`), which bootstrap read as a failed install. It stopped there, so the `git config --global user.name` and `user.email` behind it in the same plan never ran - leaving git installed and unconfigured, with the run blaming the install.

  That code from winget is treated as "nothing needed doing" now, and the rest of the plan runs. Deliberately narrow: it matches on the program as well as the code, so the same number from anything else is still a failure, and only codes actually observed to mean "already done" are listed - guessing at more would risk swallowing a real one. - **Fixed:** two stages acted before the history reset, and a run could finish with `origin` still pointing at the template (#311).

`rm -rf .git` deletes every remote. A reader who declined the reset, let the remote stage repoint `origin` and run `sync-repo`, then changed their mind and reset, silently lost the repoint - ending with a fresh repository, no origin, and a `sync-repo` that had run against a state no longer there.

Leaving `origin` at the template was worse. For a student it fails at the first `push`; for anyone with write access to the template it pushes their own work into it, and the template is public and cloned by every new reader.

Both stages now report `blocked` while the clone still points at the template - counting as work outstanding, so nothing reads as finished, but building no plan, so no command runs that the reset would undo. A clone made from `source_url` is unaffected: its origin is the reader's own, so there is nothing to wait for.

- **Added: github.com as a supported host.** Bootstrap could only be pointed at `gitlab.surrey.ac.uk`, so when that server stopped answering there was no way to run or test any of it.

  The `Host` record already carried everything github.com needed - its own greeting string (`ssh -T` exits non-zero on success against both, so the greeting is the only signal), its own key-form labels, and `repository` / `organisation` in place of `project` / `group`. Turning it on was enabling the record and widening the hostname check, not rewriting any stage.

  Only Surrey clones from Surrey: it mirrors the template onto its own GitLab so a student never needs a GitHub account. Every other host clones the GitHub original.

  gitlab.com stays unsupported, and a self-hosted instance of either family still gets the "not supported yet" answer - naming a family is not the same as having been run against that server.

37.28 0.26.4 (2026-08-11)

- **Fixed:** the SSH key was loaded into the agent for one session only, so a machine bootstrap had set up stopped authenticating after the next reboot (#303).

  `ssh-add` loaded the key into the running agent and nowhere else, and the `Host` stanza named the key with `IdentityFile` but had nothing that would reload it. The stage reported itself `OK`, quite correctly at the time, and the machine then broke silently days later.

  The failure does not name its own cause: SSH offers the *public* half of the key without needing a passphrase and only fails at the signing step, so the error looks like a key the host has rejected. It is the same trap as #246, one layer further out.

`ssh-add --apple-use-keychain` is used on macOS now, and the stanza carries `AddKeysToAgent yes` on every platform plus `UseKeychain yes` on macOS. The config check reports a stanza missing them rather than passing it, since a setup that works only until the next reboot is not a finished one.

`UseKeychain` is written on macOS alone, deliberately: an OpenSSH that does not know the keyword rejects the whole config file rather than skipping the line, which would take every other host in it down too. - **Fixed:** bootstrap logged in to the host far more often than it needed to, and a host that stops answering was reported as a rejected key (#304).

Every pass ran `ssh -T` for the check and again to decide whether the plan needed the terminal, plus a `git ls-remote` - and the `--apply` loop re-derives a plan and re-checks after every stage. A single run made dozens of logins within seconds, which is what provokes a server into refusing.

Repeats within a pass are now answered from the first connection. The memo is dropped after any applied command, so the verification re-check - the one claim bootstrap makes that is worth anything - always connects for itself.

A run now also reports how many connections it made, which is the measurement any later throttling should be built on rather than guessed at.

Separately, a connection the host accepts and then closes is reported as what it is, instead of as `could not confirm authentication` or a key the host rejected. `Permission denied` still reports as a rejection: that is a clean answer from a working server, and telling the reader to wait would be wrong.

37.29 0.26.3 (2026-08-11)

- **Fixed:** Windows stopped after installing git, saying git was not installed (#300). winget reported `Successfully installed`, and the `git config` on the next line failed with `git: not found`.

  A Windows installer adds itself to `PATH` by writing the registry and broadcasting a change. A process that is already running never sees it - its environment was copied when it started - so any stage that installs a tool and then uses it failed on a machine where the install had just succeeded.

  `PATH` is re-read from the registry between commands now, which is what opening a new terminal does. It is a different fault from #292 and #295, which were about `.cmd` shims and `PATHEXT`; this one affects `git.exe` and every other real executable too.

37.30 0.26.2 (2026-08-11)

- **Added** `prodockit init-mathjax`, and with it one implementation of

something that had two (#276). The MathJax configuration was written by bootstrap's stage 18 *and* by a template's CI, which never runs bootstrap - two copies of a file whose whole failure mode is being subtly wrong, since both produce valid JavaScript and the site simply typesets one way locally and another when published.

The delimiters are why it mattered: `inlineMath: [["\\(", "\\)"]]` carries four layers of escaping, and a copy that looks right can be wrong.

Stage 18 now runs the command, the same way the repoint stage runs `prodockit sync-repo`, and anything that builds a site without bootstrap can run it too.

- **The Ubuntu VS Code install names your architecture instead of working it out in a shell** (#287). The command carried `case "$arch" in amd64) arch=x64 ;; esac`, which reads as a hardcoded target even though it only maps dpkg's name onto VS Code's. It behaved correctly on arm64; it just could not be trusted at a glance, and a reader is being asked to approve it.

  It is resolved when the plan is built, so the command shows `linux-deb-arm64/stable` on an arm64 machine and `linux-deb-x64` on an amd64 one.

  The download and the install are two commands now rather than one shell line. The download needs no privileges and the install does, so splitting them keeps `sudo` at the front of a command where it can be seen - and where a credential timestamp expiring mid-run prompts visibly, rather than inside a shell whose output nobody is watching, which is how that stage reached its 30-minute timeout.

- **Fixed:** three more Windows commands that could not have worked (#295), found by reading the plans after #292 rather than by reaching them.

  `npm` would not have been found at all. On Windows it is `npm.cmd`, and Python's `subprocess` uses `CreateProcess`, which does not apply `PATHEXT` - so a bare `npm` reports "not found" on a machine where Node is installed correctly, and neither render toolchain installs. It is resolved by path now, exactly as VS Code's CLI is. That is also *why* #292 happened: `code` is `code.cmd` for the same reason.

  MSYS2 was assumed to be at `C:\msys64` - winget's default, not a guarantee. A machine with it elsewhere got "file not found" about a path bootstrap invented, and what breaks is the PDF build much later, since Pango is what WeasyPrint draws text through. It now says where it looked and what to do.

  The citation style download turns PowerShell's progress bar off. On PowerShell 5.1, still the Windows default, `Invoke-WebRequest`'s progress rendering makes a download dramatically slower - which reads as a hang.

- **Fixed:** Windows reported VS Code as broken immediately after installing it

(#292). The installer adds `code` to `PATH` itself - but `PATH` is read when a process starts, so the shell that has just run `winget install` cannot see it:

```
ran, but still not right: VS Code is installed, but the `code`
command is not on PATH
```

and the advice offered was a Command Palette action that exists on macOS and not on Windows.

The executable is looked for where the installer puts it now, and used by its full path - so the extensions stage works in the same session too, rather than the reader being sent to open a new terminal and start again. macOS is untouched: there the application really is installed without the command, and that Command Palette action really is how it is added.

- **Fixed:** a long install looked like a hang (#244). Applying captured every command's output, so `sudo apt update`, a 100 MB download and `apt install` behind it produced minutes of silence after `These need administrator rights.` - and a silent terminal is indistinguishable from a hung one. Installs that were working were interrupted.

  Installers now write to the terminal as they go. The re-check that follows is still captured - it *reads* what a command printed, and there are dozens per run - so both ends of it are announced, since a silent check straight after a visibly finished command reads as a hang of its own.

- **`--apply` shows every stage, numbered by where it actually is** (#284). Stages already set up were skipped in silence, and the ones remaining were numbered by their position in the queue - so `[1/17] Git` appeared while standing at stage 2 of eighteen, agreeing with nothing the reader could check against `prodockit bootstrap`'s own listing.

  ```
   1  ok    Visual Studio Code
   2  ok    Git, installed and configured
   3  ok    SSH keypair
   4  ok    SSH config points at the key

  [5/18] Key loaded into the ssh agent
         id_ed25519_gitlab is not loaded into the agent
  ```

  A stage waiting on a configuration answer is named too, rather than vanishing from the run.

- **Documentation:** the Windows prerequisites now set PowerShell's execution policy (#288). Windows blocks all scripts by default and activating a virtual environment *is* a script, so the step immediately after failed:

```
.\.venv\Scripts\Activate.ps1 : File ...\Activate.ps1 cannot be
loaded
because running scripts is disabled on this system.
```

which names the script rather than the policy blocking it, and so reads as a broken file. `Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned` comes first now, with what it does and does not change, and the CMD alternative for anyone who would rather not change it. The "activate it again" section points back at it.

37.31 0.26.1 (2026-08-11)

- **Fixed:** naming an existing repository to clone did not work (#283). The prompt asks for "an existing repository to clone instead of the template", and a repository is called `report-az1234` - not `git@gitlab.surrey.ac.uk:comm058-2026/report-az1234.git`. That answer reached `git clone` verbatim:

  ```
  Will run:
    git clone report-mb0105-v13 /Users/…/report-mb0105-v13
  failed: fatal: repository 'report-mb0105-v13' does not exist
  ```

  which reads as though the repository were missing rather than the address incomplete. Three forms are accepted now - a full URL used exactly as given, `group/name`, or just `name`, expanded against the configured host and namespace. The prompt says so.

- **Fixed:** a stage's instructions could describe the machine as it was when the run *started*, not as it is when the step is reached (#281).

  `--apply` built every plan up front, before applying anything. The SSH upload step is where it showed: it embeds your public key, and on a fresh machine the keypair stage has not run when the plans are built - so it fell back to "paste the contents of `~/.ssh/id_ed25519_gitlab.pub`" about a key that existed perfectly well by the time the reader got there.

  Each plan is now built when its stage is reached. The commands were never affected - applying a stage already re-derived its plan - so this only ever changed what was shown, which is precisely what made it hard to spot.

- **Fixed:** a first run reached configuration without ever being asked which git host to use (#279). `prodockit bootstrap` offers to fill in what is missing, and `host` has a default - so it is never *empty*, never reported missing, and was never asked on that route. Only `--configure` asked it, which is not the path a first-time reader takes.

  It is now asked on a first run - when there is no configuration file yet - and

still not re-asked of somebody who already answered it, since that is what `--configure` is for. The scripted message lists it too.

- **Accepting a host's fingerprint no longer means opening another terminal.** The SSH upload stage used to end with "run `ssh -T git@gitlab.surrey.ac.uk` in a terminal once and answer `yes`" - a step in the middle of a guided run that sends the reader somewhere else to take it.

  Bootstrap offers to run it now, with the terminal handed over, so ssh shows its own fingerprint and asks its own question in place. What has not changed is who answers: trusting a host key is still the reader's decision, and nothing answers it on their behalf.

  `BatchMode=yes` is dropped for that one command only. It is what makes a *check* safe - a check that can block is a broken check whatever it reports - and it is also exactly what suppresses ssh's fingerprint question, so it comes off deliberately and only after the reader has agreed to connect. `ConnectTimeout` stays, so an unreachable host still fails rather than hanging.

37.32 0.26.0 (2026-08-11)

- **Fixed:** the website showed raw TeX where an equation should be (#263). MathJax was loaded with no configuration at all, so `pymdownx.arithmatex`'s markup was emitted and never typeset. The configuration has to load *first* - MathJax reads `window.MathJax` once at startup, and one that arrives afterwards is ignored.

  A new **stage 18** writes that configuration and installs the browser bundle by copying it out of `tools/mathjax`'s own pinned install - the very one `prodockit pdf` pre-renders through, so a formula cannot typeset one way on screen and another in print.

  Installed, not committed. The bundle is somebody else's code and does not belong in a project's repository, so it is added to `.gitignore` alongside `tools/*/node_modules`, which it comes from. Nothing is fetched from a CDN, so the site typesets offline.

- **The source bundle's footer names the repository it came from** (#262). A bundle is a thing people hand in, and the reader of one could not tell which repository the source belonged to. The git remote now sits on the left of the footer, opposite the page number.

  The remote rather than the directory, because two checkouts of the same project have different local paths and the same remote - and a path on somebody else's machine identifies nothing. A directory with no remote falls back to its absolute path, which at least says which copy. Any credentials embedded in the remote URL are stripped: a bundle is submitted, printed and emailed.

- **Documentation:** the "activate it again" instructions now change directory first (#264). A new terminal starts in your home directory, and `source .venv/bin/activate` is a relative path - so from anywhere else it is simply not there, and the error says the file does not exist rather than that you are in the wrong place.

- **The email prompt names the host rather than a university** (#265). "Your university email address" is wrong for anybody outside a university, and wrong inside one for a reader whose GitLab login is not their university address. It now reads `The email address used for your gitlab.surrey.ac.uk login`, taking the host from the answer given a moment earlier - which is what asking the host first is for.

- **`--apply` no longer prints a Python script at the reader** (#261). Stage 16 carried an entire script as one argument, so the prompt filled the screen with source and asked for approval of it. It says what it does instead:

  ```
    Will do:
      Update ~/GitLab/report-x/.vscode/settings.json so Markdown
  opens in
      Zensical Studio's editor, and LTeX+ checks your writing as
  en-GB
  ```

  Any command carrying a script in an argument is collapsed rather than printed whole, so this cannot come back through another stage. `--dry-run` still prints commands exactly, which is what `--dry-run` is for.

- **Each manual step now asks about the thing it asked for** (#260). Every one of them ended in `Tell me when that is done` - including the step whose entire content is "this deletes your history and cannot be undone", where nothing has been asked of the reader at all and the honest question is whether to go ahead:

  ```
  Delete the template's history and start a new repository? [y/N]:
  Have you added the key to your gitlab.surrey.ac.uk account? [Y/
  n]:
  Have you created the project on gitlab.surrey.ac.uk? [Y/n]:
  Have you run the 'Shell Command' action in VS Code? [Y/n]:
  ```

  A test walks every stage on all three platforms and fails any that still asks the generic question.

- **Every `--apply` prompt defaults to yes, except the one that cannot be undone** (#259).

  The default used to follow the check's status - `MISS` meant yes, `WRONG` meant no - which is a rule a reader cannot see from the prompt, so the same

key press meant different things at different stages for reasons that were never on screen.

A plan now declares whether applying it destroys something, and exactly one does: resetting the template's commit history. Pressing Enter through a run installs things and never deletes a repository's history.

- **`--apply` now says what it is doing before it starts** (#258). It opened straight into `[1/11] Visual Studio Code`, which tells a reader which step they are on and nothing about what they have started or where it will land. It now names the version, the host, the project directory and how many stages need work, and says to run it from the project directory with the virtual environment active.
- **The SSH key form is filled in the order it actually works in** (#257). GitLab fills the Title in from the key's own comment the moment a key is pasted, so a title typed first was silently replaced - and a reader following the steps in order ended up with a list of keys all named after their email address. Key first, then Title, with the suggestion to replace the address GitLab puts there with this machine's name, which is what a key title is for.
- **The git host is now the first configuration question** (#255). Everything else is shaped by it - which URLs the browser steps send you to, which key file is looked for, whether you are creating a project or a repository - and it was not asked at all, only defaulted.

  An unusable answer is refused at the prompt and asked again, rather than stored and refused by the run five questions later, which is the whole value of asking first. The prompt and `build_context` decide through the same function, so one cannot accept what the other rejects.

  It is asked as a **hostname** - `gitlab.surrey.ac.uk`, the thing in the address bar - rather than a nickname, and judged three ways before it is stored: a host that is not a GitLab is refused, one that is not supported yet says so, and one that does not answer on port 22 is reported as unreachable.

  That last check is worth the second it costs. Without it, the first sign of an unreachable host is stage 6 reporting a rejected key - after a key has been made and pasted into a web page - and "I cannot reach this server" looks nothing like "this server refused you", which is a confusion these stages have produced three times. Re-asking is a real retry: connect the VPN, press Enter, and the second attempt succeeds.

  Configurations written before this stored `host = "surrey"`, and still resolve. Pressing Enter still gives Surrey's GitLab, the only host implemented today.

37.33 0.25.0 (2026-08-11)

- **Windows is automated end to end** (#217 phase 4). All seventeen stages now

produce commands or instructions there. MSYS2 and Pango - which WeasyPrint draws text through, and which the User Guide walks the reader through a MINGW64 shell and the Environment Variables dialog to install - are installed unattended, with the `PATH` entry added only when absent.

None of it has been run on a Windows machine, and those two facts belong in the same sentence. The evidence is that the command lists match what the guide prescribes, asserted from macOS.

- **Fixed:** every `winget install` could stop for a human. winget asks for agreement to its source terms on first use, and to a package's own terms when it carries them - on the terminal, so a captured, timed subprocess waits. That is the `sudo` failure of #243 reached by another route, and it would have met every Windows reader at stage 1. All of them now pass `--accept-source-agreements`, `--accept-package-agreements` and `-e`.

- The two steps Windows genuinely cannot automate - the `ssh-agent` service, which needs an Administrator window, and the PDF fonts, which Windows has no package manager for - are now **checked** rather than merely suggested.

- **Fixed:** five stages installed things nothing then checked (#224).

  The pattern behind four earlier bugs - a stage's check narrower than its own plan - had produced three more, all introduced in the two days before this. On a machine with node and pandoc present and nothing else, both stages reported `ok` while their plans would have installed Chromium, written shell exports, run `npm ci` for two toolchains, and installed two fonts. A reader who had installed Node themselves was told the stage was done, got no toolchains, and found out at the first diagram.

  The node check now verifies both toolchains and, on Ubuntu, that Puppeteer has a system Chromium *and* is pointed at it. The pandoc check verifies the PDF fonts - and says nothing when the machine cannot be asked, since a false alarm sends the reader to reinstall fonts they already have. The history check verifies `core.fileMode`.

- **Added:** a test that holds across every stage, so the next one inherits it rather than being audited by hand. Two gates: each stage must declare what its plan produces, and no stage whose plan installs something may report `ok` about a machine where nothing is installed. Both were confirmed to fire on a deliberately careless new stage.

- **Carried the User Guide's three ARM64 findings into bootstrap** (#249, from prodockit-userguide#104). All three were found on the same fresh Ubuntu ARM64 machine, and all three fail in a way that does not point at itself.

- **Fixed:** `npm ci` fetched a Chrome it could not run. Installing the Mermaid toolchain triggers Puppeteer's own postinstall download, and that download is not guaranteed to match the CPU it lands on - on ARM64 it fetches an

x86_64 build. Nothing fails at install time; the symptom is a diagram that will not render, much later, with nothing to connect it to the install. Ubuntu now installs a system Chromium and exports `PUPPETEER_EXECUTABLE_PATH` and `PUPPETEER_SKIP_DOWNLOAD` *before* `npm ci`, and appends them to `~/.bashrc` once - checked first, so a rerun does not leave four copies. macOS and Windows are untouched, where Puppeteer's own download is fine.

- **Fixed:** the PDF's fonts were never installed. The website loads Inter and JetBrains Mono from a CDN when a page is viewed, but a PDF has to embed the files - and WeasyPrint substitutes a fallback **silently** rather than failing. The build succeeds, the PDF looks plausible, and the only symptom is a test reporting `No 'Inter' font found`. They are installed with the graphics stack now, by cask on macOS, apt on Ubuntu, and as an instruction on Windows, which has neither.

- **Added:** the citation style the first build needs. `prodockit.bibliography` is enabled by default and points `csl_style` at `harvard-cite-them-right.csl`, which is fetched rather than committed - so `zensical serve`, `zensical build` and `prodockit pdf` all failed outright on a fresh clone. An empty file counts as `WRONG` rather than done, since a failed download leaves one behind; and a project configured for a different style is told where to find its own rather than given Harvard.

- **Bootstrap now leaves a machine you can start writing on.** Comparing it against the User Guide step by step - prompted by `ssh-add` turning out to be missing entirely in #246 - found six things it did not do at all (#248). Three new stages, and the count goes from thirteen to sixteen.

- **Fixed:** WeasyPrint was never verified, though a stage was named for it. Stage 12 read "Pandoc and WeasyPrint's libraries" and ran `pandoc --version` and nothing else, so it reported `ok` on a machine whose first PDF build would fail at `cannot load library`. Importing WeasyPrint is the test now, and a strict one: the import loads Pango through the system linker, so success proves both the Python package and the native libraries. `pip` exiting zero proves neither. The pandoc stage is renamed to what it actually checks.

- **Fixed:** the VS Code extension list disagreed with the guide. Even Better TOML and LTeX+ - both required - were missing, while Code Spell Checker, which comes from the *optional* tooling page, was installed in their place. Marketplace identifiers were checked rather than guessed: the obvious `valentjn.vscode-ltex` returns 404, and the maintained fork is published under `ltex-plus`.

- **Added:** the project's own virtual environment, and its dependencies. Bootstrap cloned a template shipping a `requirements.txt` and never installed it, so Zensical itself was absent from the project. The new stage creates `<project>/.venv` and installs into it by naming that interpreter

explicitly - a bare `pip install` would find bootstrap's own pip, install the project's dependencies into bootstrap's environment, exit zero, and leave the project's `.venv` empty.

- **Added:** a history of your own. The guide resets the template's commit history; bootstrap carried its whole log and branches into every project. This is the only stage that destroys anything, so it reports `WRONG` rather than `MISSING` - deleting history should not happen by pressing Enter - and is judged by whether `origin` still points at the template, never by whether commits exist. The latter would tell somebody who had been writing for a month that their history needed deleting.

- **Added:** the editor's settings for the project. Markdown is associated with Zensical Studio's language mode, and LTeX+ is set to **the language the machine is in** rather than the guide's `en-GB` - bootstrap runs on other people's computers, and a document checked against the wrong variety of a language is worse than one not checked, because the corrections are confident and wrong. When the locale cannot be read the setting is left out rather than guessed. Existing settings are merged, never overwritten.

- **Fixed:** `git remote set-url` failed on a repository `git init` had just created, which is exactly what the new history stage leaves behind. The repoint now adds the remote when there is none.

- **Fixed:** a passphrase-protected key could never pass the SSH stage, because nothing ever loaded it into an agent (#246).

  Stage 3 tells the reader to set a passphrase, and every ssh command bootstrap runs carries `BatchMode=yes`, which forbids prompting for one. Those two are only compatible if an agent holds the decrypted key - and `ssh-add` appeared nowhere in the code.

  The failure is a quiet one. `ssh -T` reads the `.pub` file and offers the public half without needing a passphrase; the host then challenges it to sign, that needs the private half, and there is nobody to ask. Authentication fails, and the upload stage reports `the host rejected the key` about a key that is correct, uploaded, and simply locked - the third failure in this area to lie about its own cause, after #234 and #239.

  A new **stage 5** checks the key's fingerprint against `ssh-add -l` and runs `ssh-add` when it is absent, with the terminal handed over so the passphrase prompt has somewhere to appear. It is the only command in bootstrap that gets the terminal.

  Starting an agent is not automatable and is not attempted: `eval "$(ssh-agent -s)"` exports `SSH_AUTH_SOCK` into the shell that runs it, and a subprocess cannot export into its parent - so bootstrap would start an agent, set the variable in a shell that then exits, and change nothing. When none is running it says so and gives the line to run. On Windows the agent is a service, and enabling it needs an Administrator window.

- Bootstrap now reports **thirteen** stages, and the numbers after the new stage 5 shift by one.

- **Fixed:** installing the VS Code extensions defaulted to *no* (#242). With none of the three installed the prompt read `Run 3 commands? [y/N]`, so pressing Enter declined the install the reader had run bootstrap to get. The prompt's default follows the stage's state, and `WRONG` defaults to no because reapplying over something can destroy work - but nothing installed is `MISSING`, and there is nothing there to destroy. A partly-installed set stays `WRONG`, and still asks.

- **Fixed:** a slow install was killed and reported as failed while it went on to succeed (#243).

  Two causes, and both needed fixing. `sudo` reads its password from `/dev/tty` exactly as `ssh` does, so it asked from inside a captured subprocess - the reader typed a password into a command whose output was being swallowed, and their thinking time counted against the clock. `sudo -v` now runs first, with the terminal attached, purely to refresh the credential; plans needing no privileges are never asked.

  And an install is now allowed 30 minutes where a check gets 5. VS Code's `.deb` is around 100 MB, and a download plus `apt install` on a virtual machine can legitimately run past five minutes - which printed `failed` next to a VS Code that was, by then, installed. A timeout is also reported in the reader's terms now, rather than as the raw command repr with the useful sentence at the end of it.

- **Fixed:** `apt` gave up instead of waiting for the dpkg lock (#244). On a freshly installed Ubuntu the process holding it is usually `unattended-upgrades`, which starts on boot and can hold it for minutes, so a first run met `Unable to acquire the dpkg frontend lock` - which reads as a broken machine rather than "something else is mid-update". Every apt call bootstrap makes now passes `DPkg::Lock::Timeout`, so apt waits, which is what a human would have done.

- **Fixed:** bootstrap never wrote `~/.ssh/config`, so ssh had no way to know which key belonged to the host and asked for a password instead (#239).

  Without a `Host` stanza, ssh offers its own defaults - `id_rsa`, `id_ed25519` - never tries `id_ed25519_gitlab`, and falls back to `git@gitlab.surrey.ac.uk's password:`. That is indistinguishable from a key the host has rejected, so the reader goes back and re-pastes a key that was never the problem, because it was never offered.

  A new **stage 4** writes the stanza in the User Guide's own shape, before the upload stage that depends on it. It is appended, never written over - an ssh config is the reader's own file - and a stanza that already exists pointing elsewhere is explained rather than edited, since ssh takes the first match and

rewriting somebody's ssh config underneath them is not an installer's business.

The same stage sets `chmod 600` on the key and the config. ssh ignores a private key others can read (`Permissions 0644 ... are too open. This private key will be ignored`) and then falls back to a password - the same symptom from a different cause. Windows has no `chmod`, and restricts a profile file to its owner already.

- Bootstrap now reports **twelve** stages rather than eleven, and the later stage numbers shift by one.
- The SSH upload stage now **prints the public key** between `======= PUBLIC KEY =======` markers, rather than naming the file it lives in (#238).

  "Paste the contents of `~/.ssh/id_ed25519_gitlab.pub`" asked a first-time reader to find a dotfile, open it in something, and copy the right one of two files whose names differ by four characters - where picking the wrong one uploads the *private* key. Only `.pub` is ever read, and the markers matter as much as the key: it is one long line that wraps in a terminal, and a key pasted a character short is rejected exactly like one never uploaded.
- The same stage now follows the **User Guide's own wording**: the keys page is reached through the host's menus (profile avatar → *Edit profile* → *Access* > *SSH Keys*) with the URL kept as a shortcut rather than as the only way in, and the form's fields are listed under one step instead of numbered as separate errands.
- **GitLab's expiry date is now spelled out** (#238). GitLab requires one, fills it in a year ahead, and will not let you clear it - so a reader who accepts the default is locked out mid-course, and the failure arrives months later as a permission error indistinguishable from a misconfigured key. Bootstrap did not mention the field at all. GitHub has no such field, and says so.

37.34 0.24.1 (2026-08-10)

- **Fixed:** `--apply` could not install VS Code on Ubuntu (#233).

  The plan asked the reader to download a `.deb` from the website and then ran `sudo apt install -y ./code.deb` - a file that exists under that name nowhere. The download is called `code_1.132.0-…_arm64.deb` and it arrives in `~/Downloads`, so the command failed whether or not the reader had done their half.

  VS Code is now downloaded like pandoc already is, from Microsoft's own `linux-deb-$arch/stable` redirect - permanent, so there is no version to pin and let go stale - with `dpkg --print-architecture` choosing between them. The stage is fully automated on Ubuntu now, and nothing is asked of the reader.

- **Fixed:** a failing command could be reported by its *warning* rather than its error (#233). apt prints `WARNING: apt does not have a stable CLI interface` every time it runs from a script, and that was the line shown as the reason the stage failed - describing nothing that went wrong, and pointing at the reader's scripting rather than at the missing file two lines below it. Warnings are now skipped while any other line remains.

- **Fixed:** `--apply` stopped the whole run at the SSH key stage on any machine whose key was not yet on the host - the one state that stage exists to fix (#234).

  Stage 4 carried `ssh -T` as a plan *command*, so 0.24.0's commands-before-instructions ordering ran the probe before saying a word about uploading anything, read its exit code as a failure, and ended the run with `Stopping - later stages depend on this one.`

  The probe could never have passed: `ssh -T` against a git host exits non-zero even on success, which is why the greeting - not the exit code - is what the check reads. Both browser stages are now instructions only, and the re-check that already follows every apply does the verifying.

- A plan's manual steps are now ordered against its commands rather than merely coexisting with them. `instructions` come *before* the commands because the commands depend on them ("Download the .deb", then `apt install ./code.deb`); the new `follow_up` comes *after*, for the step that only makes sense once they have run (VS Code's Command Palette, after the `brew install` that provides it).

  Both orderings had shipped broken - instructions-only skipped the install entirely (#230), and commands-first broke the guide-and-verify stages (#234) - so the order is now stated by each plan instead of being a convention the caller has to guess.

37.35 0.24.0 (2026-08-10)

- **Fixed:** `--apply` skipped the install commands when a stage had both commands and instructions — VS Code on a fresh macOS got only the "open the Command Palette" instruction without the `brew install` that puts the application there in the first place (#230).

- Bootstrap on Ubuntu now downloads the pinned pandoc release from GitHub instead of `apt install pandoc`, which installs a version several major versions behind — far enough to render code blocks as justified prose (#207). The download uses `dpkg --print-architecture` so the same command works on amd64, arm64 and under Rosetta.

- The pandoc check now reports `WRONG` when the installed version is too old (< 3.x), rather than `ok`. Ubuntu's apt package is 2.x on some LTS releases,

and a check that merely asks "is pandoc installed?" would pass on those and leave the reader to discover the problem at their first `prodockit pdf`.

- Ubuntu's git plan now runs `sudo apt update` before the first `apt install`, so a clean machine with an empty package index does not fail to find the package.

- `prodockit bootstrap --help` said "ten stages" a release after there were eleven. The count is now asserted against the stage list, so prose cannot drift from the thing it describes.

37.36 0.23.0 (2026-08-10)

- **Fixed:** bootstrap asked for an email and then never applied it (#222).

  A new stage 8 sets `user.name` and `user.email` on the clone, and checks them with `git config --local`. The old check read them without `--local`, which falls back to the global value - so it passed on any machine with any identity at all, the plan never ran, and commits went out under whatever address git already had.

  On Surrey's GitLab a commit whose author address matches no known account is not linked to one, so coursework can appear to be authored by an unrecognised user - with nothing to suggest why, since every stage reported `ok`.

  Per-repository rather than global: a global `user.email` is a legitimate personal preference, and a tool that sets up one university project should not rewrite the identity used for everything else. Eleven stages now, not ten.

- The bootstrap page now explains how to meet its own prerequisite (#223).

  It named Python as the one thing bootstrap cannot install and then left the reader there - the worst place for a gap, since it is the first thing they hit and the point at which they have no working tooling to fall back on.

  Per-platform instructions for Python and a virtual environment, the `externally-managed-environment` refusal and why a venv is the answer to it, the three things Windows' installer gets wrong (PATH, path length limit, the Microsoft Store placeholder), `python3-venv` being a separate package on Debian, reactivating in a new terminal, and checking `prodockit --version` afterwards - an older install on `PATH` shadows a newer one silently.

- **Fixed:** `prodockit bootstrap` could stop dead at a password prompt (#225).

  On a machine whose SSH key was not yet uploaded, the stage 4 check fell back to password authentication and simply waited - a check that can block is a broken check, and it stopped a test run outright.

  The cause was subtler than it looked: `ssh` reads passwords and

passphrases from `/dev/tty` directly, deliberately bypassing stdin, so the existing `stdin=DEVNULL` never could have prevented it. Every command bootstrap runs now also gets an environment that cannot ask - `BatchMode=yes` for ssh, `GIT_TERMINAL_PROMPT=0` for git, both reaching `git clone` and `git ls-remote` through `GIT_SSH_COMMAND`, which had the same hang waiting in stages 5 and 6. A `GIT_SSH_COMMAND` you have set yourself is left alone.

An unknown host key is now reported, with the one command that fixes it, rather than auto-accepted: trusting a host is a decision that belongs to you, not to an installer.

37.37 0.22.0 (2026-08-10)

- `prodockit bootstrap` - phase 2: configuration and installing the template (#217).

  `--configure` asks each question with the stored answer as its default; `--apply` sets up the stages that need it, asking before each. A stage that is simply absent defaults to yes, one that exists but is wrong defaults to **no** - reapplying over something already there is the case that can destroy work.

  Every stage is re-checked after being applied. A command exiting zero says the installer ran, not that the thing works.

  The template is cloned from the configured host's own copy - Surrey mirrors it onto its own GitLab, so a student there never needs a GitHub account. `source_url` overrides it with an existing repository for a reader who has been given one.

- `prodockit bootstrap` - phase 1: check and plan a full machine install (#217).

  The User Guide's install sequence is long and easy to get half-right in ways that surface much later. This turns it into ten stages that can each be checked and repaired individually.

  Nothing installs anything yet. Phase 1 reports by default - `prodockit bootstrap` with no options checks every stage and changes nothing - with `--dry-run` to print the exact commands a real run would use. Read-only is the default deliberately: the alternative, once applying exists, is a command that starts installing software because somebody typed it to see what it did.

  Two stages are deliberately never automated: uploading an SSH key and creating your own project both need an authenticated human in a browser, and the alternative is a Personal Access Token typed into a tool aimed at first-time students. They guide and then *verify* - `ssh -T` and `git ls-remote` - which is the half a written instruction cannot do.

Surrey's GitLab is the only supported host; `gitlab.com` and `github.com` are declared but refused, so adding them later is filling in a record rather than rewriting the stages.

37.38 0.21.0 (2026-08-10)

- A Zensical rename of `render()`'s result keys now stops the PDF build with a message naming Zensical, the installed version, and the page being rendered - not a bare `KeyError` (#171).

  `zensical.markdown.render.render` is undocumented - `zensical/__init__.py` exports only `build` / `serve` / `version` - so both the function and the shape of what it returns can change in a **patch** release without registering upstream as a breaking change. `result["content"]` read unguarded meant a rename surfaced as `KeyError: 'content'`, raised from inside a loop over nav pages, with nothing naming Zensical, the installed version, or the upgrade that caused it. The reader sees prodockit's own traceback and reasonably concludes prodockit is broken.

  Deliberately not #167's warn-and-degrade shape: there is no sensible degraded PDF to produce, and a page silently rendered with empty HTML would be exactly the kind of build-succeeds-output-broken failure this project keeps landing on. The build still stops - it now just says why. `TypeError` is caught alongside `KeyError`: if `render()` starts returning an object rather than a dict, that is what a subscript raises instead, and the diagnosis is identical.

- `prodockit source-bundle` is a new command, split out of `prodockit pdf` (#212).

  The two PDFs a project can produce serve different purposes - a rendered document, and a record of what was written - and previously could not be built independently. `pdf_source_bundle = true` made every `prodockit pdf` run also pay for a `git ls-files` scan and a second `weasyprint` invocation regardless of whether anyone wanted the result, and a project that wanted only an updated source bundle still paid for the far more expensive Pandoc/WeasyPrint document pipeline - Mermaid/TeX pre-rendering included - to get it. `pdf_source_bundle` is gone; `prodockit pdf` never builds a source bundle as a side effect now, under any config.

  The default set of files bundled is also narrower: every `.md` file under `docs_dir` plus `zensical.toml`, rather than every text file `.gitignore` doesn't exclude. A project's custom Python extensions, Lua filters, CSS and tests are source code, not documentation, and a report built from a template has no reason to bundle its own tooling alongside the document it produced. The wider, whole-repository bundle - useful for a project doing its own academic-integrity verification of custom code, not just prose - is still available from Python directly (`build_source_bundle()` with no `files`

argument, or `discover_source_files()`'s own result passed explicitly); the CLI command itself has no flag for it, by design.

Output moves from the project's top-level directory into `docs_dir` by default, so Zensical serves it with no separate copy step - `prodockit-template` and `prodockit-userguide` both carried one before this.

- `prodockit pins` now manages pandoc too, matched as a `PANDOC_VERSION` CI variable (#209).

  Pandoc never appears as a pip specifier - it is a build-provided binary, not a Python dependency - so it needed a fourth declaration shape alongside the pip specifier, GitHub runner label and container image tag `pins` already understood: `PANDOC_VERSION: "3.10.1"`, the same in a GitHub `env:` block or a GitLab `variables:` block.

  It joins the default managed set, alongside Zensical and WeasyPrint, for the reason #209 raised directly: three sibling projects were keeping the same `PANDOC_VERSION` in step by hand, across workflow files, with a comment saying "keep in sync" - which is exactly the kind of drift this module exists to catch instead. `prodockit pins --check` in this project's own CI now verifies its three copies agree with no workflow change required.

  A CI variable's name keeps whatever case it was declared in on rewrite - `PANDOC_VERSION`, never `pandoc_VERSION`. Every other managed shape can safely reuse its lower-cased lookup key as the replacement text, because a runner label or image tag is conventionally lower-case already; an environment variable is conventionally not, and using the lookup key there would silently break the workflow step that reads it back. Found and fixed twice over: once in the rewrite itself, and again in the CLI's own progress-line display, which had the same bug independently - caught only by running `--set` against copies of this project's real workflow files rather than trusting the unit tests alone.

37.39 0.20.1 (2026-08-09)

- CI builds with a pinned upstream pandoc instead of the runner image's (#209).

  `ubuntu-24.04` ships pandoc 3.1.3. That is what made the code-block fault below invisible: CI published correct PDFs on the old package while every local build on a current pandoc was wrong, and the next runner-image bump would have broken the published output with no commit to blame.

  `ci.yml`, `docs.yml` and `drift.yml` now install a pinned release from pandoc's own downloads, the way `prodockit-template` already did. `drift.yml` matters as much as the other two: it exists to detect published output drifting, and an unpinned pandoc is a source of drift it could not see.

The documented CI recipe is updated to match, in both its GitHub and GitLab forms, and the limitations page records the whole episode.

- Fenced code blocks keep their line structure in the PDF (#207).

  Reported as stretched word spacing inside code blocks. The spacing was real, but it was a symptom: the blocks had stopped being code blocks at all.

  Pandoc's HTML reader only accepts `<pre><code>` as a code block when that `<code>` holds nothing but text. Zensical's highlighter emits per-token `<span>`s, a `__codelineno` anchor per line, and a leading empty `<span></span>` - each of which, on its own, makes the reader give up and treat the block as ordinary inline content.

  The `<pre>` was then absent from what Pandoc handed WeasyPrint, so `white-space: pre-wrap` had nothing to apply to. Every newline collapsed, the block reflowed and justified like a paragraph, and each token became its own inline `<code>` - which is where the wide gaps came from, and why they varied from line to line. A six-line install snippet came out as four wrapped rows with `".[dev]"` split across two of them.

  Each `<pre>` is now reduced to a single plain-text `<code>` child before Pandoc sees it. Nothing is lost by dropping the token markup: the PDF stylesheet defines no syntax colours, so it was never rendering anything - it was only destroying the block.

  Both a unit test on the HTML handed to Pandoc and a check on the finished PDF, since the intermediate shape being right does not prove the artefact is.

37.40 0.20.0 (2026-08-09)

- `prodockit sync-repo` now keeps `site_url` in step too (#200).

  `site_url` is the address a site is *published* at, and it was the one host-coupled setting nothing managed. Zensical puts it in every page's `<link rel="canonical">` and every `sitemap.xml` entry, so a wrong one tells search engines the real home of the documentation is somewhere else - while the site builds and looks perfect.

  This was not only a hazard when moving hosts. The project template shipped `site_url` pointing at the *repository*, so a fresh clone published a GitHub page as the canonical URL of every documentation page from its first build.

  Only GitHub Pages is derived, since only its shape follows from the remote. GitLab is not guessed at: a self-hosted instance serves Pages from `pages_external_url`, which the remote reveals nothing about, and gitlab.com now issues unique domains rather than the old `<group>.gitlab.io/<project>` path. A confidently wrong canonical URL is

worse than none, so those projects set `pages_base` and the repository name is appended to it.

An existing value is replaced only when it is already a Pages URL, or points at a code host and so cannot be a site. Anything else is a custom domain and is left alone with a note - `--check` is a CI gate, and rewriting a deliberate value would redden every later build for a correct config. An absent `site_url` is not invented.

- `prodockit sync-repo` keeps the whole GitLab namespace, instead of dropping everything between the first group and the project (#201).

  GitLab nests groups, and only the first segment was kept, so `cs-dept/year3/report` became `cs-dept/report` - a project that does not exist. That went into `repo_url`, the edit links and the badges alike, so on an instance where the reader is signed in and the links would otherwise work, they led to a 404 instead. Nested groups are the normal arrangement on university and company instances.

  Silent, as ever: `sync-repo` reported success, `--check` reported being in sync, and the site built.

  The full path now drives every URL. `repo_name` is the exception - it is a header caption rather than a link, so it shows the immediate parent (`year3/report`) to keep the header readable, while the header still links to the correct `repo_url`. A single-segment namespace, which is every GitHub project and most GitLab ones, is unchanged.

  An existing test had asserted the truncated form, with a comment saying it was what the badge and edit URLs needed - the opposite of what they need. That is corrected rather than worked around.

- `prodockit sync-repo` now fills an empty `repo-badges` block, and points GitLab badges at the instance the remote actually names (#198).

  Two faults, which together made the feature look absent rather than broken.

  The block pattern required a newline immediately before the end marker, and the start group had already consumed the only one present when the two markers sit on consecutive lines. That empty pair is precisely how a template ships a badge row for `sync-repo` to fill in - the shape this project's own documentation gives as the example - so the one case that had to work was the one that could not. Worse, the failure to match was then reported as `no repo-badges markers in README.md`, which is what a reader sees when the markers are absent. The markers were there; nothing said so.

  Separately, the badge set is chosen by host *kind*, and `gitlab` matches any instance. The URLs were built with `gitlab.com` hardcoded, so a university or company GitLab got a badge row pointing at a `gitlab.com` project that

does not exist - plausible-looking links to nothing, which is worse than the stale GitHub badges they replaced.

Badge links are now built from the real host. The GitLab build badge is the one the instance serves itself rather than shields.io's, which is also the only version that can work on a private instance: the reader is already authenticated against the host that would refuse an external badge service. Stars and forks have no instance-served equivalent and shields.io reads `gitlab.com` only, so those two are emitted for `gitlab.com` alone rather than as guaranteed broken images everywhere else. - The CI recipe in the documentation matches the workflows again (#202).

The `concurrency` group that serialises Pages deploys had been in `docs.yml` since July and was never copied across, so the recipe handed a reader the race it was written to prevent - on a page whose neighbouring section describes that failure in detail. Also pins the runner to `ubuntu-24.04` rather than `ubuntu-latest` (the page's two other blocks already did), adds the `workflow_dispatch` trigger the real redeploy workflow carries, and recommends `prodockit pins --check --offline` on a gate, since the bare form asks PyPI a question a pull request should not be able to fail on.

37.41 0.19.1 (2026-08-07)

- `prodockit pdf` now renders Mermaid diagrams on Windows (#195).

  `npm` writes three shims for each tool into `node_modules/.bin` on Windows: an extensionless POSIX shell script, plus `.cmd` and `.ps1` wrappers. Only the wrappers can be started by `CreateProcess`, but the extensionless one is the spelling every platform's documentation uses, and it is the one prodockit looked for.

  `os.path.exists` confirmed it, so detection succeeded and the build proceeded - then every diagram failed at the point of use with `[WinError 193] %1 is not a valid Win32 application`. Existing and runnable are different questions on Windows, and only the first was being asked.

  The result was the failure this project keeps producing: a PDF that built successfully, exited zero, and printed `Wrote docs\site_documentation.pdf`, with each diagram silently left as its own `graph LR` definition text. The per-diagram warning named a Win32 error code rather than a missing shim, which points at nothing a reader can act on.

  Every candidate location is now tried with an executable suffix first, so a default path or a `pdf_mmdc_bin` naming the bare `mmdc` resolves to the `mmdc.cmd` beside it. `tex2svg` was never affected - it is invoked as `node tex2svg.js`, so Node is the executable and the script is just an argument.

37.42 0.19.0 (2026-08-07)

- `prodockit pins` no longer treats a version specifier written in a comment as a declaration (#184).

  Declarations are matched textually, and `zensical ⩾ 0.0.52` reads the same whether it is a dependency or a sentence about one. A comment explaining why a package was pinned therefore became a phantom declaration site: reported in the inventory, counted towards the consistency check, and rewritten by `--set` - turning correct prose into a statement that was no longer true, in a file nobody thought they were changing.

  That mattered more after 0.18.1 made `pins --check --offline` a pull-request gate and widened the managed set to four packages including `markdown`, a word that turns up in comments far more often than `zensical`. One explanatory sentence could redden a build.

  Only the part of a line before a `#` is scanned now, in both directions: discovery ignores comments, and `--set` substitutes only over the code part, so a declaration carrying a trailing comment that quotes the same specifier no longer has both rewritten. A `#` inside quotes is not a comment, and a trailing comment still leaves its own declaration findable - narrowing far enough to break either would have been worse than the bug.

- `prodockit pdf` now prints the failing command's own stderr instead of only its exit code (#188).

  `PdfBuildError` and `SourceBundleError` have always captured the stderr of the process that failed - `pandoc`, and through it whichever PDF engine pandoc invoked - and the CLI printed only the exception's message. That message names a command and an exit code, so the single most useful thing prodockit knew about the failure was collected and then thrown away.

  Found the hard way: following the User Guide on a clean macOS machine produced `Error: pandoc exited with status 43` and nothing more. Status 43 is pandoc's `PandocPDFError` - the PDF engine failed, not pandoc - and the engine's own message said WeasyPrint could not load `libgobject-2.0-0`, named the libraries it needs and linked its installation instructions. Recovering it needed a throwaway script calling the Python API to reach `PdfBuildError.stderr`. The same build now says so directly.

  Printed whole rather than summarised: warnings come first and the real error last, so a head-truncated excerpt would hide exactly the part worth reading. Nothing is printed when the failure captured no stderr.

- Subprocess output is now decoded as UTF-8 explicitly, instead of with whatever the locale says (#191).

  Seven `subprocess.run(..., text=True)` calls named no encoding, so

Python used the locale's. That is UTF-8 on macOS and Linux and `cp1252` on a default Windows install, and every tool prodockit runs - pandoc, git, mermaid-cli - emits UTF-8. One accented author name in a `.bib` file was enough to stop `zensical serve` on Windows.

The symptom pointed nowhere near the cause. The decode fails on a reader thread inside `subprocess`, so the traceback named `threading` and `cp1252`; `run()` then returned with `stdout=None`, and the next line raised `TypeError: Incoming markup is of an invalid type: None` from BeautifulSoup. What a user saw was a type error about markup.

Guarded by a test that walks the source for text-mode subprocess calls with no `encoding=`, so a new one cannot reintroduce it. Deliberately a source check rather than a behavioural one: where this suite runs the locale is already UTF-8, so a behavioural test would pass with or without the fix.

- A new `unbookmarked` heading class removes a heading from the PDF's bookmark outline - the navigation pane a PDF reader shows down the side - separately from `unlisted`, which only keeps it off the generated Table of Contents *page* (#173).

  The two are built by different tools. Pandoc's own `pandoc.structure.table_of_contents()` builds the contents page and honours `unlisted`; WeasyPrint builds the outline separately, straight from its own UA stylesheet, and nothing about `unlisted` (or any other class prodockit stamps on a heading) ever reached it - `prodockit.pdf.css` set no `bookmark-level` rule at all. An `unlisted` `h1` therefore still became a top-level outline node, and because outline nesting follows heading level, every later, lower-level heading nested underneath it instead of under its real chapter. Silent either way: the build succeeds and exits zero.

  `unlisted` itself is unchanged, and keeps meaning exactly what Pandoc means by it. `unbookmarked` is new and additive - `prodockit.pdf.css` now gives `h1.unbookmarked`-`h6.unbookmarked` `bookmark-level: none`, and nothing prodockit generates itself carries the class, so no existing PDF's outline moves. The back-of-book index's own A/B/C letter headings, the Table of Contents title, and every cover-page heading all carry `unnumbered` `unlisted` and deliberately stay off `unbookmarked`, since a reader still needs them in the outline to navigate - keying the new rule off `unlisted` itself, instead of adding a separate class, would have removed all three.

  Documented alongside `unnumbered` in prodockit.headings and PDF generation, where the two-tables-of-contents split is now explained as its own surface for the first time. This project's own `docs/extensions/refs.md` (whose illustrative headings needed exactly this, previously worked around with a project-local `.example-heading { bookmark-level: none; }` in `docs/stylesheets/extra.css`) now uses `unbookmarked` directly instead.

- The docs build now pins `Markdown` and `pymdown-extensions`, and `prodockit pins` manages them (#178).

  Pinning a dependency exactly does nothing for the packages underneath it. `zensical` was pinned; it declares only floors for the two libraries that actually turn every page's Markdown into HTML, so every build rendered with whatever those resolved to that morning. A release of either could have changed the HTML on every page, and the site would have published it, green, with nothing committed.

  That matters more here than in most projects that render Markdown: `prodockit.pdf.css` and `prodockit.pdf.lua` both match on the specific class shapes pymdownx emits, so the renderer is an input to the PDF's own correctness, not only to the website's appearance.

  `prodockit pins` covers both by default now, so `--check` and the weekly drift job watch them alongside `zensical` and `weasyprint`. The drift job installs and reports them on both sides of its comparison - without that, a pymdownx release would have appeared in *both* builds and diffed to nothing.

  `Markdown`'s floor in `pyproject.toml` moves from 3.4 to 3.10.3 to match the pinned version, the same convention `zensical` already follows. Older releases are not known to break; the floor records what this is built and tested against.

  `ci.yml` now runs `prodockit pins --check --offline`, so a commit that moves a version in one file and forgets another fails the build instead of reaching `main`. The `--offline` form is deliberate: plain `--check` also fails when a package is behind PyPI, which would turn every open pull request red the day upstream ships a release, for a reason no contributor could act on. Watching for newer releases stays with the weekly drift job, which reports rather than fails.

- `prodockit pins --set PACKAGE=VERSION` no longer prompts for the packages it was not given, and no longer throws away the one it was.

  `--set` says "here is the version, do not ask" - the help text has always said it implies `--no-input` - but it only pre-answered the package named. Any other package in the managed set still prompted, so `pins --set zensical=0.0.53` in a release script or a drift job stopped at `weasyprint: version to set [69.0]:` and, with no stdin to answer it, aborted. Nothing was written *including the zensical sites it had been told explicitly to set* - the sharper half of the bug, because the run reported an abort rather than a partial write, and the workaround (`--package zensical --set zensical=0.0.53`) reads as though the two options mean different things.

  `--set` and `--latest` now suppress prompting outright. A package with no version given is reported under "Left untouched (no version given)" and its files are not opened. `--no-input`, which the help text named but the

command never accepted, is now a real flag for the same behaviour with nothing set.

An interrupted *interactive* run now keeps what it was already told: answers given before the interrupt are applied, the rest is left alone, and it still exits non-zero and says so. Writing nothing was the wrong reading of a cancelled prompt - the answers above it were not cancelled.

- Zensical pinned to **0.0.53** (from 0.0.52), after a byte-for-byte comparison of the site and PDF built under both.

  The PDF is **byte-identical** - same 994,803 bytes, same SHA-256, 151 pages, 268 outline entries. The site changes 22 HTML files, each by exactly three lines: the `generator` meta tag and two asset-bundle hashes. No rendered page content changes.

  The bundles themselves grew 146 bytes (CSS) and 121 (JS). The whole CSS delta is `.md-search__button` - offsets and `text-align` moved out of the base rule into `[dir=ltr]` / `[dir=rtl]` variants, which is the release's right-to-left search fix. Every class prodockit couples to is unchanged (`md-typeset` 778 occurrences either side, `md-nav` 237, `md-content` 60; glightbox and mermaid hooks identical). Unlike 0.0.52, this release redraws no icon: icons are inlined as SVG into the HTML, and the HTML is identical apart from those three lines.

  Two builds of 0.0.52 were compared first, to establish that a difference means something: they were identical down to the PDF's raw SHA-256, so the build is fully deterministic and nothing above is build noise.

  The release notes list a `pymdownx` bump to 11.0, but that is a **floor raise only** - 0.0.52 declares `pymdown-extensions ⩾ 10.21.3` and 0.0.53 declares `⩾ 11.0`, and both already resolve to 11.0.1. It was therefore not a variable in this comparison. See #178: the library that actually renders Markdown is not pinned at all.

  Every undocumented Zensical API on the Zensical coupling page was re-checked against 0.0.53 and still resolves; `__all__` is still exactly `build` / `serve` / `version`. The page's "last verified against" now reads 0.0.53.

- The "this document contains TeX maths" warning no longer fires on a page that merely *documents* maths handling (#176).

  A page quoting `<div class="arithmatex">` in a code span reaches the rendered HTML as `&lt;div class="arithmatex"&gt;` - Python-Markdown escapes the angle brackets and leaves the attribute text alone. The detector matched the bare `class="..."`, so prose about maths counted as maths, and `prodockit pdf` warned that formulas would ship as raw LaTeX in a document with no formulas in it. This project's own `devcons/limitations.md` did exactly that on every build.

It now anchors on a real `<div` / `<span` opening tag, the way the Mermaid detector always has - which is why Mermaid never had the problem. A false alarm matters more here than most: the warning exists to make a *silent* degradation visible, and one that cries wolf teaches the reader to ignore the only signal there is.

- The MathJax toolchain `prodockit pdf` uses to pre-render TeX maths is now committed and installed in CI, alongside the Mermaid one (#175).

  `tools/mathjax/` gains `package.json`, `package-lock.json` and `tex2svg.js` exactly as `prodockit init-tools` scaffolds them, with `npm ci --prefix tools/mathjax` added to `docs.yml` and `drift.yml`, both of which installed Mermaid only. Both sibling repos already tracked all three files; this brings extensions in line.

  This fixed no live defect. These docs contain no maths, so nothing was being published as raw LaTeX. The warning that prompted the work was the false positive fixed above. What it buys is that the first page to add a formula renders it, rather than shipping the source the way the Mermaid diagram in `extensions/bibliography.md` once did.

  The lockfile is committed deliberately, matching `tools/mermaid` - without it CI resolves whatever MathJax is newest rather than the version the output was checked against, which is the drift `prodockit pins` exists to catch.

  Worth knowing when relying on this: as merged, nothing here is guarded by a test that can fail. `test_no_page_contains_unrendered_tex_source` passes with the toolchain removed again, because with no maths in these docs it can only confirm that nothing unrendered reaches the PDF, not that the renderer is present. Closing that needs `pymdownx.arithmatex` enabled on this project's own docs, so the toolchain is exercised by the build that ships it.

- Documentation restructured: the six build-and-operate pages are now grouped under a **Dev Considerations** section, and `Refs` is renamed **Cross-References** (#170).

  The nav had grown to thirteen top-level entries, enough that the theme cut the tabs off. Nine now, with `docs/devcons/` holding continuous integration, repository metadata, version pinning and drift, testing your built site, Zensical coupling, and limitations and workarounds behind a short introduction.

  `Managing the build` was one 747-line page carrying three unrelated subjects; it is split at its own level-2 boundaries into `continuous-integration.md`, `repo-metadata.md` and `pinning-drift.md`, each promoted one heading level so it reads as a page rather than a fragment. **Every existing heading anchor is unchanged**, so the only part of a link that moved is the file it points at.

  Page URLs *have* moved - `/continuous-integration/`, `/testing/`, `/limitations/` and `/zensical-coupling/` are now under `/devcons/`. Every

link inside the docs and the README was updated with them; any external bookmark to an old URL will 404.

37.43 0.18.1 (2026-08-04)

- New Zensical coupling page, listing every Zensical API prodockit depends on that Zensical neither documents nor treats as public (#166).

  Zensical exports exactly three names - `build`, `serve`, `version` - and prodockit uses none of them. Everything else it reaches for is a module-level import from inside the package, so any of it can be renamed in a *patch* release without that counting as a breaking change upstream. Nothing in these docs said which, so the coupling was invisible until a release broke it.

  The page records each API with its call site and why it is needed, the undocumented *data shapes* prodockit reads (the resolved nav tree's `url` / `is_index` / `children`, `env.conf`, the packaged icon directory layout), and - deliberately - a list of the Zensical features that *are* documented, so it doubles as a triage aid when something breaks.

  It also sets out what a Zensical upgrade actually needs: not a green test run, but a build of both site and PDF with the **output compared**, since 0.0.52 silently redrew an icon with no source change. Recorded as last verified against 0.0.52.

- prodockit now says so when an undocumented Zensical API it depends on moves, instead of failing in a way that points at its own internals (#167).

  `prodockit._zensical`'s two Zensical lookups guarded the *import* and nothing after it, so they handled exactly one failure mode - "Zensical isn't installed". A renamed `ContextPreprocessor.from_markdown`, a changed signature, or a `Page.path` renamed to something else all import perfectly and then raise `AttributeError` / `TypeError` from deep inside a `zensical build`, with nothing connecting it to the version bump that caused it. None of these APIs is public - `zensical` exports only `build` / `serve` / `version` - so a rename can arrive in a patch release.

  The guards now cover the attribute access and the call as well, and emit a warning naming the API, the installed Zensical version, and what actually degrades. Deliberately *not* a bare `except Exception: return None`: `page_source()` returning None silently makes every page share one default source, each render wiping the previous page's registry entries, so cross-page references, citations and glossary terms resolve to `??` on a site that still builds and exits zero. That is #54 again, in a form no test would catch.

  A plain `ImportError` stays silent - not running under Zensical is a legitimate state for any other Python-Markdown consumer - and each API is reported once per process rather than once per page, since `page_source()` runs on every render.

37.44 0.18.0 (2026-08-02)

- A reference to a page's *title* heading now resolves in the PDF (#163). Each page's own anchor was written onto its first heading, **replacing** whatever id that heading already had - so `\ref{chapter-two}` pointed at an anchor that no longer existed. The reference still rendered its text, so nothing looked wrong; the link was simply dead, and `\autoref` printed "on page" with nothing after it.

  Sections *within* a page were unaffected, which is why it went unnoticed - the broken case is the most natural reference to write.

  The page anchor is now carried by an empty span inside that heading, so both exist: the page's own anchor for a cross-page link with no fragment, and the heading's for a reference to the heading itself. Inside rather than before, because a numbered `h1` breaks to a new page and an anchor placed before one would sit at the foot of the previous page, reporting a page number one too low.

- Cross-references say what they point at. `\ref{id}` renders the target's **number and name** - "1.1 Configuration" - because a bare "see 1.1" tells a reader nothing about what they are being sent to, and having to look it up in the contents defeats the point of the cross-reference (#151).

  An appendix needs nothing special: its letter is already the first segment of its number, so a reference to an appendix section renders "A.1 Terms".

  An `unnumbered` heading - a cover page, appendix front matter - has no number but does have a name, so it resolves to just the name. Only a genuinely unknown id is unresolved.

- New `\autoref{id}`, for references that still work on paper. It renders the same text as `\ref{id}` and additionally carries the target's **page number** in the PDF: *"Configuration is covered in 1.1 Configuration on page 12."*

  The suffix comes from prodockit.pdf's own stylesheet, so it appears only there - a page number on a scrolling website would be meaningless - and needs no configuration. Which to use is a per-reference decision: `\autoref{id}` where a printed reader needs to turn to something, `\ref{id}` where "on page N" would just be noise.

  Implemented with CSS `target-counter()`, which resolves the target's page at layout time and so needs no second pass - unlike the back-of-book index, which has to deduplicate a term repeated on one page and therefore cannot use it.

- The generated index now appears in the Table of Contents (#141). Its heading carried Pandoc's `unlisted` class, which is exactly what

`pandoc.structure.table_of_contents()` honours - so an index a reader goes looking for was the one section the contents never mentioned.

It stays `unnumbered`, so it takes no chapter number and appears at the end alongside the last chapter. Two things deliberately keep `unlisted`: the Table of Contents heading itself, since a contents listing itself is noise, and the index's own A/B/C letter headings, which would otherwise fill the contents with single letters.

- The default bottom margin is now `2.5cm` rather than `2cm`, so a multi-line running footer is not cropped when printed (#139).

  The footer is top-aligned in the bottom margin and grows *downward* as it gains lines, so whatever the margin does not use is the space left before the paper edge. This project's own footer is two lines - a copyright line and a "Made with" credit - and measured on a real render it ended **6.1mm** from the edge. Consumer and office printers commonly cannot print within 5-6.4mm, so the second line was at real risk of being cropped: the PDF was correct and the paper was not.

  2.5cm leaves about 11.1mm. The other three margins are unchanged, so pages are 5mm shorter and a long document gains a few pages - this project's own went from 134 to 139. Set `pdf_margin_bottom` to `"2cm"` to restore the previous layout, and set it higher for a footer of three or more lines.

  Raising the margin rather than moving the footer within it: both footer boxes are top-aligned with matching `margin-top` / `padding-top` so their border-tops form one continuous rule across the page. Bottom-aligning them to guarantee clearance instead would let a two-line box and the one-line page number sit at different heights and break that rule.

37.45 0.17.5 (2026-08-02)

- `prodockit --version` prints the installed version (#149). There was no way to tell which version a checkout or CI job was actually running short of `pip show prodockit`.

  Prints the bare number, matching `zensical --version` rather than click's own default of `prodockit, version X.Y.Z` - the two are normally installed and reported together, and a matching format means neither needs parsing to compare them.

- `prodockit pins` now sees a requirement with extras (#156). `package[extra]==version` is an ordinary shape - `prodockit[index]`, `uvicorn[standard]`, `celery[redis]` - and the bracket sits between the name and the operator, exactly where the matcher expected one to follow the other. Such a declaration was invisible.

The failure mode was the bad one: `pins` reported "not declared anywhere", which reads as *nothing to do* rather than *could not parse this*. A project could pin something, run `pins --check` in CI, and get a pass while the declaration drifted untouched.

Extras are recorded per site and written back on rewrite, the same way each site's operator already is, so `prodockit[index]⩾0.17.2` becomes `prodockit[index]==0.17.4` rather than `prodockit==0.17.4`. Dropping them would silently stop installing an optional dependency - for `prodockit[index]` that means the back-of-book index quietly stops being generated.

37.46 0.17.4 (2026-08-02)

Documentation only - no library, CLI or CI behaviour changes.

- The three pages covering how a prodockit build is run and kept stable - repository metadata, continuous integration, and version pinning and drift - are now one page, **Managing the build**. They were written separately and read as three answers to the same question; a reader setting up a pipeline needed all three and had no reason to expect the third. Each is now a section, introduced by a short chapter overview naming all three, and continuous integration keeps its own H2 rather than being the page's implicit subject.

  Existing links still resolve: `version-pinning.md` and `repository-metadata.md` are gone, and every reference to them - in the README, in these release notes - now points at the corresponding anchor on the merged page.

- The GitHub Actions example in that page now shows the `verify` job from 0.17.2, so the recipe a reader copies includes the delivery check rather than the version that predates it.

- The 0.17.1 entry below now carries the CI costs that were actually measured, rather than describing them as "paid once per interpreter". Installing WeasyPrint took the 3.13 job from 59 seconds to about 14 minutes - a 13-minute install, compiling from source, against roughly 20 seconds on 3.10-3.12 - and enabling the pip cache brought that install back to 15 seconds and the job to under two minutes. Those are the figures that tell a reader whether the cache is worth enabling in their own matrix.

37.47 0.17.3 (2026-08-02)

- Docs deploys no longer run from a release event, which never worked and failed silently. A release event runs against `refs/tags/<tag>`, so the Pages deployment it created carried a tag ref - and with Pages configured `source: {branch: main}`, that deployment was accepted, reported `success`, and

was then never served. The site simply carried on returning the previous build.

Every release from 0.17.0 to 0.17.2 did this, each needing a manual `gh workflow run docs.yml --ref main` afterwards. The evidence is unambiguous: across nine Pages deployments, every one from `main` went live and every one from a tag ref did not (#147).

It was not a race between the push-triggered and release-triggered runs, which is what it looked like for a long time. The concurrency group serialised them correctly - on 0.17.2 the release run started only after the push run had finished, deployed later, and still lost.

`docs.yml` now triggers on `push` and `workflow_dispatch` only. A new `release-redeploy.yml` handles the post-release rebuild by re-triggering `docs.yml` against `main`, so the deployment carries a branch ref - automating exactly what the manual fix did each time. The `tag: prodockit-v*` entry in the github-pages environment's deployment branch policies is now redundant.

Continuous integration documents the trap, since the obvious fix is the broken one.

37.48 0.17.2 (2026-07-30)

- README, the package description and the module docstring now cover `prodockit pins` alongside the other commands, so a reader arriving from PyPI sees it exists.
- New `prodockit pins` command: shows every place a build-input version is declared across a project, and moves them all together. Pinning a build means writing the same version in several files at once - a floor in `pyproject.toml`, an exact pin in each CI job that builds the docs, another in whatever job checks for drift - and nothing enforces that they agree. When they disagree the failure is quiet: CI builds with one version while the declared floor says another.

  Run it with no options for a prompt per package - press ++enter++ to take the newest release on PyPI, or type a version. Each site keeps **its own operator**, so a library floor stays a floor and a build pin stays exact; one answer updates every file correctly.

  Three shapes of declaration are recognised, because a build input is not always a pip package: a pip specifier (`zensical==0.0.52`), a GitHub runner label (`runs-on: ubuntu-24.04`) and a container image tag (`image: python:3.13`). The last two carry `pandoc`, the fonts a PDF embeds and the Chrome that rasterises diagrams - none of which pip can reach - so they belong in the same inventory. Neither has a package index to ask, so the suggested default is simply what is already set.

`--check` reports and exits non-zero if anything is behind PyPI *or* if the files disagree with each other, for a scheduled job. `--set PACKAGE=VERSION`, `--latest` and `--offline` cover the non-interactive cases.

Scans both CI layouts - `.github/workflows/` and `.gitlab-ci.yml` / `.gitlab/` - plus `pyproject.toml`, `setup.cfg` and root `requirements` / `constraints` files, so the same command works whichever host a project uses. Build output and virtualenvs are skipped, so a stale copy of a workflow inside `site/` or `.venv/` is not mistaken for a declaration.

- `weasyprint` is now pinned in the docs build and the test job, alongside `zensical`. It decides pagination, so a release that lays out one paragraph differently shifts every page number after it - and those page numbers are content, resolved into the back-of-book index and the table of contents. That is a silently wrong document rather than a failed build. Pinned in `ci.yml` too, because the real-render tests assert on where things physically land, which makes the layout engine an input to those assertions rather than an implementation detail.

- New `drift.yml`, so pinning does not mean going quietly stale. Weekly it builds the docs twice in one job - once with the pinned versions, once with the newest - diffs the results byte for byte, runs the built-output checks against the newer build, and opens an issue saying what an upgrade would change. Both builds share a job, so pandoc, Chrome, fonts and the runner image are identical between them and any difference is attributable to the upgraded packages alone. It reports rather than fails, and keeps one open issue updated in place: a scheduled job that goes red every week trains everyone to ignore it.

- CI runners are pinned to `ubuntu-24.04` rather than `ubuntu-latest`. The image is the build input pip cannot reach: `pandoc` comes from it (and distribution packages lag upstream far enough that some markdown edge cases parse differently), as do the fonts the PDF embeds and the Chrome that rasterises Mermaid diagrams. On `ubuntu-latest` all three move the day GitHub migrates the label, with nothing committed - the same silent change the package pins exist to prevent, one layer down.

  `ubuntu-latest` already *is* 24.04, so nothing changes today; it takes effect at the migration, which is exactly when a documentation build wants to be told rather than surprised. Pinned images are retired about a year after the following LTS and the job then fails outright rather than drifting - the better failure, and at a time of your choosing.

- New Version pinning and drift page documenting the whole arrangement - where a version gets declared and why the forms differ, `prodockit pins`, and a drift job for **both** GitHub Actions and GitLab CI, the latter using pipeline schedules and the GitLab issues API.

- `zensical` now declares a floor (`⩾0.0.52`) rather than being left entirely

open. It records the version prodockit is developed and built against - not a minimum below which anything breaks, since 0.0.50 and 0.0.51 both work. Zensical is pre-1.0 and prodockit reaches well past its public surface (the config loader, the per-page render context `prodockit.headings` detects, the icon set `prodockit.pdf.icons` resolves against), so which version produced a given build is worth recording. A floor rather than an exact pin because prodockit is a library: `==` would propagate to every consumer and conflict with any project needing a different Zensical.

What prompted this is worth keeping visible. Rebuilding against 0.0.52 and diffing the output byte for byte against 0.0.51: the PDF is identical, and every page of the website differs - Font Awesome moved 7.2.0 to 7.3.1 and redrew the GitHub brand icon used in the header's repository link and the social footer. One visible change out of the nine icons the site uses, arriving with nothing committed. The rest is the generator version string, content-hashed asset filenames, and minified bundle churn.

A floor does not prevent that recurring: dependency resolution still takes whatever is newest, so the next Zensical release can change the published site the same way. Reproducible builds need a constraint on the *build* side - in `docs.yml` - rather than in library metadata.

37.49 0.17.1 (2026-07-29)

- This project's own PDF now has the back-of-book index its own docs describe. `zensical.toml` never set `include = true` for `prodockit.index`, so the live `\index{}` markers in `docs/extensions/index-terms.md`'s `=== "Result"` tabs produced nothing: the page documenting the feature sat in a PDF that didn't have it.

  Turning the setting on alone would only have indexed those five demo markers, so the docs are now marked properly throughout: every module at its own page's opening sentence, every option/setting/fixture in the reference table that defines it, and the concepts and external tools where each is introduced. Options are nested under what they belong to, so the `source` / `registry` / `unresolved` that five different extensions each define group per module instead of collapsing into one misleading entry. That gives a real two-page index of about 95 entries, covering every marker shape the extension supports.

  Costs one extra `pandoc` +WeasyPrint pass on that one build. The twelve single-page `prodockit pdf -m` builds in `docs.yml` pay nothing: `build_pdf()` skips the second pass entirely for a document with no markers, and none of those pages has any. Mermaid and TeX pre-rendering both happen before either pass, so neither is repeated. Measured locally at roughly 8s → 12s for a 119-page document, against a docs workflow dominated by apt, Chrome and npm setup.

- The 0.17.0 fix above now has a permanent CI guard. Its regression test needs

a real `pandoc` +WeasyPrint install, so it is deselected in the `test` job, which installs neither - the shipped fix was effectively unguarded. `tests/test_built_docs.py` gains three `built`-marked checks, which run in `docs.yml` after a real build: that no marker reached the PDF's text layer, that the index was generated with every marked term in it, and that each entry cites a page the term is actually on.

The text-layer check matches only a marker followed by a real digit, rather than the blunter substring the synthetic test can afford. These release notes legitimately print `⟦prodockit-index-N⟧` (with a literal "N") in the 0.17.0 entry below, and name the `h2.prodockit-index-letter` CSS class in an earlier one - both are prose about the feature, and both belong in the text layer.

- A long index term no longer overflows its own column. An index entry is often a single unbroken token with nowhere to wrap - a dotted module path, a long option name, a function signature - and `div.prodockit-index-entry` set no `overflow-wrap`, so such a term ran straight over the column rule into the next column's entries, and off the page edge entirely from the right-hand column. Found while indexing this project's own docs. `overflow-wrap: break-word` (not `break-all`, so ordinary multi-word terms still break at their spaces first) fixes it. Covered by a real-render test that measures where the glyphs actually land, since it renders as perfectly ordinary text either way and only its position gives it away.

- The generated index's pages are now headed "Index" (or whatever `prodockit.index`'s `title` setting says) rather than by the last chapter of the document. The index's own `h1` is `unnumbered` - correct, since it must not take a section number or a Table of Contents entry - but `unnumbered` is also what excludes a heading from feeding the running header's `chapter-title` string. That exclusion suits the Table of Contents, which sits at the front where `chapter-title` is still empty, and fails for the index, which is always the very last thing in the document: whatever chapter came last simply stayed in the header. This project's own PDF was headed "18. License" across its entire index. The heading now carries a `prodockit-index-title` class with a `string-set` rule of its own; nothing else about it changes.

- CI now installs WeasyPrint for the `test` job, so the nine real-render tests that assert on where things actually land in a finished PDF - which page a heading starts on, what the running header says, whether a long index term stayed inside its column - finally run. They are gated on a real `pandoc` + `weasyprint` install and `ci.yml` had only `pandoc`, so they skipped silently on every run; `docs.yml` does install WeasyPrint, but runs only `-m built` against `tests/test_built_docs.py`, so it never reached them either. The behaviour they exist to pin was going unchecked everywhere, including both fixes above.

  The tests themselves add about 20 seconds per matrix entry. Installing WeasyPrint costs more, and unevenly: it pulls in packages with no wheel

published for every interpreter the matrix covers. On 3.10-3.12 the install step took roughly 20 seconds; on 3.13 it spent **13 minutes** compiling them from source, taking that job from 59 seconds to about 14 minutes.

`actions/setup-python`'s own pip cache is now enabled, which holds pip's *built* wheels as well as its downloads. That brought the same 3.13 install back to **15 seconds** and the job to under two minutes.

37.50 0.17.0 (2026-07-28)

- Back-of-book index markers no longer leave anything in the PDF's text layer (#133). Every `\index{Term}` used to deposit a `⟦prodockit-index-N⟧` token next to the word it marked - 67 of them in this project's own User Guide. They were invisible on the page, but real text in the file, so they surfaced in copy and paste, in the reader's own search, in text extraction, and, worst, in screen readers, read out mid-sentence.

  The markers had to stay findable, which is why they were text: the second pass locates each one to learn its page number, and a `font-size: 0` span is dropped from the text layer entirely, leaving nothing to find. Shrinking further was never an option.

  Each occurrence is now marked with an *empty* span carrying only an `id`, and its page is read back from the PDF's own named destinations - WeasyPrint emits one per element `id`, whether or not anything links to it. Nothing is encoded as text, so nothing can leak. An empty span also occupies no width, which removes the previous design's awkward question of whether stripping 67 tiny spans between passes might reflow the very pages whose numbers they had just recorded.

  Needs no configuration change, and no new dependency: `pymupdf` was already required for a generated index, and the API used has been available since well below the existing floor.

37.51 0.16.0 (2026-07-28)

- The website's `prodockit-v0.41.0` and the PDF's `{RELEASE}` come from deliberately different sources - `git describe --tags` on the local checkout, and the host's releases API - and could disagree with nothing saying so (#125). A reader comparing a published site with its downloadable PDF could see two different release numbers, and neither build had failed.

  Neither source changes: each is right for its own context. `prodockit-v0.41.0` is re-evaluated on every website rebuild, including every save under `zensical serve`, so it must not make a network call; `{RELEASE}` serves a cover page that isn't part of a macro-rendered site at all. What was missing is that the disagreement was invisible.

`prodockit pdf` now warns when the two will show different things, naming both values and where each came from. The macros pass warns separately when `prodockit-v0.41.0` came back empty *because* the checkout is a shallow clone, which fetches no tags even from a repository that has them - the failure behind #122, silent because an empty value just renders as a missing line. The warning names `fetch-depth: 0` and `GIT_DEPTH`, and fires once per process rather than once per rebuild.

A project with no tags at all stays silent: that is a normal state, and warning about it would only train people to ignore the message. See Limitations and workarounds.

37.52 0.15.2 (2026-07-28)

- `prodockit.testing`'s Mermaid check now detects a diagram whose arrows were swallowed by font ligatures. A PDF set in a font with programming ligatures - JetBrains Mono, a common choice for code blocks - renders `⟶` as a single glyph that extracts back out as `//>`, leaving every arrow-based pattern blind to an unrendered diagram. Node-definition brackets (`id[Label]`) survive extraction, so they now count as evidence too. Found in `prodockit-template`, whose own PDF uses that font.

  Accepting bracket syntax needed the keywords tightened first, or it would have fired on ordinary prose. `graph` / `flowchart` now require their direction token (`graph LR`), which no sentence produces by accident, and the four diagram types that are also plain English words - `gantt`, `journey`, `pie`, `timeline` - accept only arrow or entity-relationship evidence, never brackets. Without that, a line beginning "timeline of the project" followed by `data[1]` read as an unrendered diagram.

37.53 0.15.1 (2026-07-27)

- Documentation only. New Continuous integration page: complete, working GitHub Actions and GitLab CI recipes for building a prodockit site, and the reasoning behind each part (#124).

  This knowledge previously existed only as comments scattered across three projects' workflow files, which is why the same mistakes kept recurring. The page is organised around the failure modes that are *silent* - fonts not installed (WeasyPrint substitutes one and the PDF just looks wrong), a shallow clone fetching no tags (the release line vanishes), the renamed Puppeteer variable, and a release deploying before its own tag exists.

37.54 0.15.0 (2026-07-27)

- New `prodockit.testing` package - `pip install prodockit[testing]`. A pytest plugin giving a project `prodockit_*` fixtures for its own built site and

PDF, resolved from its Zensical config rather than an assumed layout, plus checks for the failure modes every prodockit project shares. See Testing your built site.

Chiefly `assert_no_unrendered_mermaid()` and `assert_no_unrendered_tex()`, which turn the 0.12.0 build warning into a test failure - three projects published PDFs full of raw `flowchart LR ...` source and literal LaTeX before anyone noticed (#124).

The Mermaid check requires a diagram-type keyword *and* Mermaid's own link syntax nearby. Several diagram types (`graph`, `pie`, `journey`, `timeline`) are ordinary English words, and PDF line breaks fall wherever text wraps, so a keyword-only check read "a visual commit graph and richer history browsing" as an unrendered diagram - passing locally and failing in CI only because different fonts wrapped that sentence differently.

The plugin registers through pytest's entry point, so it loads into every test run in an environment where prodockit is installed. It imports nothing heavy at module scope, prefixes every fixture, and fails individual fixtures rather than collection, so an unrelated project is unaffected.

37.55 0.14.0 (2026-07-27)

- New `prodockit init-tools` command: scaffolds the Node tooling `prodockit pdf` needs to render Mermaid diagrams and TeX maths, then prints the `npm` commands, `.gitignore` lines and CI environment variables to finish the job. See Mermaid diagrams and TeX maths.

  `prodockit.pdf` has always looked for `tools/mermaid/node_modules/.bin/mmdc` and `tools/mathjax/tex2svg.js` while shipping neither, leaving every project to hand-write the same two manifests and the same `tex2svg.js` (#124). All three projects using it got that wrong independently: one had no `tools/` directory at all and published PDFs full of raw `flowchart LR ...` source, two set the pre-rename `PUPPETEER_SKIP_CHROMIUM_DOWNLOAD` that puppeteer 25.x ignores, and one committed two config files nothing reads. The canonical copies now live in the library, pinned in one place.

  Existing files are never overwritten without `--force`, since a project will have run `npm ci` against its own committed lockfile.

- The missing-renderer warning added in 0.12.0 now points at `prodockit init-tools` rather than `npm ci --prefix tools/mermaid`, which could not work in the case that actually happened - no `tools/` directory to install into.

37.56 0.13.0 (2026-07-27)

- Documented the Python version requirement. `requires-python = "⩾3.10"` has always been enforced by `pip`, and CI has always tested 3.10-3.13, but

nothing said so anywhere a reader would look: the PyPI classifiers listed only `Programming Language :: Python :: 3`, and neither the README nor the installation page mentioned a version at all. Per-version classifiers added, and Installation now opens with the requirement plus a full table of what `pip` does and doesn't install

- including `pandoc` / `weasyprint`, and the Node tooling needed only for Mermaid diagrams and TeX maths in the PDF.

- New `prodockit sync-repo` command, and the `prodockit.sync_repo` module behind it: keeps `repo_url`, `repo_name`, `[project.theme.icon] repo`, `edit_uri` and your README's badge row matching the git remote a checkout actually uses, so forking or mirroring a project between GitHub, GitLab and Bitbucket doesn't leave stale links, the wrong brand icon, or badges pointing at somebody else's repository. `--check` writes nothing and exits non-zero on drift, for CI. See Repository metadata.

  This was previously a `sync_repo_icon.py` script copied byte-for-byte between two consuming projects (#124). Two things changed in promoting it: the default branch is now detected from the remote rather than hardcoded to `main`, and `repo_name` keeps whichever shape (`owner/repo` or bare `repo`) your config already uses, since Zensical prints it verbatim in the site header and the script's fixed choice would have restyled the header of any project using the other one.

- The command-line entry point moved from `prodockit.pdf.cli` to `prodockit.cli`, now that it has commands unrelated to the PDF build. `prodockit.pdf.cli` re-exports `main`, so an entry point recorded in an already-installed environment keeps working.

37.57 0.12.0 (2026-07-27)

- `prodockit pdf` now warns when a document contains Mermaid diagrams or TeX maths but the renderer needed to turn them into static images wasn't found. Both have always been optional, and both deliberately leave the content untouched rather than failing the build - the right default for a project using neither, but silent for one that *is* using them. That silence let raw `flowchart LR ...` source and literal LaTeX reach the published PDFs of three separate projects, including this one's own. The build still succeeds; the degradation is simply announced, and the warning names the fix rather than just the symptom.
- Fixed this project's own docs: the architecture diagram in `extensions/bibliography.md` had never rendered in any published PDF, for exactly the reason above. `tools/mermaid` and the CI steps to install it are now in place - so the page describing how Mermaid fences are pre-rendered finally demonstrates it.

37.58 0.11.1 (2026-07-27)

No code changes - fixes a regression in 0.11.0's own release:

- The "Release: `<tag>`" cover-page line added in 0.11.0 never actually appeared on the deployed site or PDF - `actions/checkout`'s default shallow clone (`fetch-depth: 1`) fetches no tags at all, so `prodockit.zensical_macros`' `prodockit-v0.41.0` (`git describe --tags --abbrev=0`) always returned `""` in CI, even though it worked correctly in any full local checkout. Fixed by adding `fetch-depth: 0` to this project's own `docs.yml`/`ci.yml` checkout steps. Confirmed directly against the redeployed live site.
- Fixed #120: `.cover-hero-subtitle`/`.cover-hero-release` (this project's own `docs/stylesheets/extra.css`) both rendered in black rather than the intended grey in the PDF - `color: var(--md-default-fg-color--light)` is undefined in `prodockit.pdf`'s own generated CSS, and an unresolved `var()` with no fallback falls back to the *inherited* value instead of erroring.
- Fixed #121: rather than just special-casing the PDF, added a `var()` fallback pointing at this project's own explicit `--prodockit-fg-color-light` custom property - the live website still gets the real, theme-adaptive Zensical variable whenever it's actually defined, but a future Zensical rename/removal of that variable can no longer silently break this project's own PDF-visible text again.

37.59 0.11.0 (2026-07-27)

New `prodockit-v0.41.0` variable in `prodockit.zensical_macros`: the latest git tag reachable from `HEAD` (e.g. `"1.2.0"`, `""` if the checkout has no tags at all), matching `word_count`/`repo_url`'s existing pattern. Promotes a one-off custom macro `prodockit-userguide` already had in its own project-local `macros.py` into the shared library, so every project gets it for free instead of hand-rolling the same `git describe --tags --abbrev=0` shell-out. Resolves identically for the website and for `prodockit pdf`, since both render through the same macro environment - unlike `prodockit.pdf`'s own `{RELEASE}` cover-page marker, which queries the host's GitHub/GitLab API instead, for a project whose cover page isn't part of a live, macro-rendered site at all.

This project's own docs site now shows "Release: `<tag>`" on its cover page, using the `.cover-hero-release` CSS class that was already defined but never actually used - enabling the macros plugin here for the first time in the process. Also fixed a real bug found while wiring this up: `.cover-hero-release` rendered in a bold weight in the PDF (`Inter-Ultra-Bold` instead of `Inter`, confirmed via the PDF's own extracted font info) - it was missing the `font-weight: 400` its sibling `.cover-hero-subtitle` already has, for the same Pandoc-pipeline reason.

Fixes #116. See #120 for a related, separate rendering issue found along the way

(both `.cover-hero-subtitle` and `.cover-hero-release` render in black rather than the intended light gray in the PDF - not fixed here).

37.60 0.10.9 (2026-07-26)

Docs only, no code changes:

- New Limitations and workarounds page, consolidating every confirmed limitation across prodockit's three surfaces (the Python-Markdown extensions, `prodockit.pdf`, and `prodockit.zensical_macros`) and its workaround in one place, including cross-page resolution going stale under `zensical serve`'s live reload (#99) - previously scattered across `pdf.md`'s own "Limitations and workarounds" section, which is now a short pointer here instead.
- `bibliography.md`'s "What this project's own template and user guide currently do" section no longer says neither downstream project has adopted `prodockit.bibliography` - prodockit-template has since migrated, and is now a real worked example of Multiple sections: a cited-only `references.md` and a separate, everything-included `bibliography.md` from a distinct further-reading `.bib` file.
- Added GitHub issue templates (bug report, feature request), matching prodockit-template's own.

37.61 0.10.8 (2026-07-26)

Breaking: the two citation extensions swap syntaxes. `prodockit.bibliography` now uses `\cite{id}` - the natural spelling for the workflow most projects reach for first - and `prodockit.citations` moves to `\citeref{id}`.

If you use `prodockit.citations`, replace every `\cite{id}` in your content with `\citeref{id}`. A `\cite{id}` left behind will no longer resolve: with `prodockit.citations` alone it falls through as literal text, and in a build that also enables `prodockit.bibliography` it will be read as a `.bib` key instead.

The two extensions still own distinct syntaxes, so either can be enabled alongside the other without hijacking it - only which name belongs to which has changed. Each is now pinned by a test asserting it leaves the other's syntax alone; neither had one before, so nothing would have caught the two silently overlapping.

Multi-key citations remain a `prodockit.citations` feature (`\citeref{id1,id2}`); `prodockit.bibliography` matches single keys only, for the reasons its own documentation gives.

Part of #111; the matching updates to `prodockit-template` and `prodockit-userguide` follow separately.

37.62 0.10.7 (2026-07-25)

Two numbering fixes, both cases of the same shape: a raw-text pre-scan and a parsed-document count applying different rules to the same content.

Website and PDF disagreeing on appendix letters. Appendix lettering was computed twice. The website gave a letter to any page whose front matter set `is_appendix`, unconditionally; the PDF's Lua filter counted appendix h1s instead. An appendix page contributing no numbered h1 - none at all, or one marked `unnumbered`, which the filter skips - gave Lua nothing to count, so every later appendix came out a letter early. Reproduced against `prodockit-template`: the same Bibliography page rendered as "Appendix E" on the website but "Appendix D" in the PDF.

`prodockit.pdf.build` now assigns every appendix page's letter once, by position in the page list, and stamps it on that page's heading for `Header()` to read rather than counting for itself. A page with no numbered h1 still consumes its letter, so the two stay in step.

Setext headings invisible to the nav pre-scan. `_count_top_level_headings()` matched ATX headings only, so a title underlined with `=` was never counted - though Zensical's renderer and Pandoc both produce a real h1 either way, and both number it. Later pages' start counts came out short, which broke numbering twice over: the website contradicted itself, giving the next page the chapter number a setext heading had already taken, and it fell one behind the PDF. Each rule of the setext syntax was checked against the real renderer rather than assumed - a single `=` is enough, a `-` underline is an h2, a two-line paragraph followed by `=====` is no heading at all, and `attr_list` puts `{: .unnumbered }` on the text line.

Stale cross-page references under `zensical serve`. `preseed()` is deliberately first-wins so a duplicate id resolves to the first page in nav order. Under `zensical build` that is all it has to do; under `zensical serve` the process outlives the files, and first-wins silently discarded fresh data - an edited definition kept its original text, and a deleted one stayed resolvable, until the dev server restarted. `preseed_attr_from_nav()` now rebuilds the provisional set from scratch on each scan, and `prodockit.headings` keys its cached scan on every nav page's mtime and size as well as the numbering settings, so an edit to a page a given render doesn't touch still invalidates it.

Note this fixes the pre-scan, not Zensical's incremental rebuild: verified against a live `zensical serve`, editing page A does not cause page B to re-render, so B's output still only catches up when B itself is re-rendered. What is guaranteed now is that a page being re-rendered resolves against current disk state rather than a snapshot from server startup.

Fixes #104, #106 and #99.

37.63 0.10.6 (2026-07-25)

Fixed footnote text in the PDF rendering in a column roughly two thirds of the page's content width, wrapping a footnote onto five short lines whose first held just two words.

This had been documented in `prodockit.pdf.css`'s own `.pdf-footnote` rule as an unfixable WeasyPrint `float: footnote` limitation being tracked upstream. That was a misdiagnosis. The real cause is Pandoc's HTML writer hard-wrapping its output at ~72 columns, inserting newlines *inside* the `<span class="pdf-footnote">` carrying a footnote's text. Those newlines are insignificant whitespace in HTML, but WeasyPrint's `float: footnote` width computation treats them as hard break opportunities when sizing the footnote area, so the rendered text collapses toward the longest *source* line rather than the page's content width.

Confirmed by holding the HTML and CSS constant and varying only the Pandoc step: the same document rendered 304.1pt wide through Pandoc but 462.9pt straight through WeasyPrint. `prodockit.pdf.build` now passes `--wrap=none`, giving 474.2pt of a ~482pt content width. WeasyPrint 69.0 is already the latest release, so waiting upstream had no path forward.

This does not stop footnotes wrapping in the PDF - `--wrap=none` governs only newlines in the generated HTML source, never the engine's own line breaking. A long footnote still occupies as many lines as it needs, each now using the full measure: a seven-sentence footnote renders on six full-width lines rather than ten narrow ones.

The misleading `KNOWN LIMITATION` comment in `prodockit.pdf.css` has been replaced with the real cause so it isn't re-derived, and two regression tests added - one asserting the flag at the command level (so it can't be dropped where Pandoc isn't installed), one measuring real rendered text width via a genuine Pandoc/WeasyPrint build, since the CSS is identical either way and only a real render can tell the two apart.

Fixes #101, reported downstream as prodockit-template#95.

37.64 0.10.5 (2026-07-25)

Fixed a real bug found by reproducing it directly: a cross-page `\ref{id}` under `zensical build` depended on whether the page defining `id` happened to be rendered before the page referencing it, in the same Python process - `zensical build` renders pages neither in nav order nor necessarily all in one process, so this was pure luck. Reproduced locally on a 3-page site: the previous release left 1-5 references as `??` per build, varying from one otherwise-identical build to the next (only 2 of 12 clean builds fully resolved).

`prodockit.headings` now pre-scans every nav page's headings (ids, levels,

section numbers) into the shared registry before any page is converted - the same idea `prodockit.citations` / `prodockit.glossary` already use for their own cross-page definitions - so resolution no longer depends on build order. A page actually rendered in this process still supersedes its own pre-scanned entries with the real ones. 20 consecutive clean builds now produce byte-identical, fully-resolved output; `prodockit-template`'s entire built site is byte-identical before and after this change.

Also fixed a second bug found while testing the above: extension order isn't guaranteed, so `prodockit.refs` can construct its own default `HeadingsExtension` and trigger the pre-scan with per-document numbering before a project's configured `numbering = "continuous"` instance runs - silently showing a cross-page reference's number one step behind (`1.1` instead of `2.1`) roughly 1 build in 12. The pre-scan now reruns if a differently-configured instance appears.

Fixes #54. See #99 for a related, separate limitation found along the way (this pre-scan can go stale under `zensical serve`'s live-reload - not fixed here).

37.65 0.10.4 (2026-07-25)

- Added `CONTRIBUTING.md` and a `.github/pull_request_template.md`, adapted from prodockit-template's own - library-specific setup (`pip install -e ".[dev]"`, `pytest` / `ruff` / `mypy`, the real-`pandoc` requirement for `prodockit.bibliography`'s own tests) rather than the template's assignment-writing framing. Linked from README.md's Development section.
- Docs: added an admonition to `headings.md` documenting a real gap this project's own docs hit directly while enabling every prodockit extension on its own site (#87) - Zensical's automatic cross-page id sharing only warns (rather than raising) on a heading name shared across pages, and build order isn't guaranteed stable, so the "keeping the first" winner can change between builds. Documents the fix - an explicit, page-prefixed id via `attr_list` - pointing to this project's own docs as a worked example.

No code changes.

37.66 0.10.3 (2026-07-25)

Docs: several extension pages mixed "why it was built this way" design rationale into their opening paragraph, ahead of the practical "what it does"/"how to use it" content most readers want first - moved that reasoning further down each page instead:

- `bibliography.md`: trimmed the intro to the core value proposition, moved the Pandoc-delegation rationale and architecture diagram out of Requirements (now just what to install) into a new "How it works" section

after Reference, and folded the "can be enabled alongside prodockit.citations" note into Comparing the two approaches.
- `citations.md` / `glossary.md`: moved the "why bundled into one extension, unlike headings/refs" rationale from the intro into their own Syntax section.
- `tables.md`: moved the "auto-enables Python-Markdown's own tables extension" implementation note from the intro into Syntax.
- `index-terms.md`: moved the "why a Markdown extension, not attr_list" rationale from the intro to the end of CSS hooks, wrapped in its own admonition naming the subject explicitly.

`headings.md` / `refs.md` were already lean and needed no changes. No syntax, behaviour, or code changes.

37.67 0.10.2 (2026-07-25)

Docs: `extensions/index-terms.md` described the live website's search as generic "browser/Ctrl-F search" - updated to point at Zensical's own built-in site search instead, a more accurate and discoverable description of how a reader actually finds a term on the live website. No code changes.

37.68 0.10.1 (2026-07-24)

Docs: `prodockit.index`'s marking syntax and `prodockit.pdf`'s back-of-book index *generation* were split across two pages (`extensions/index-terms.md` and `pdf.md` respectively), even though the marker is useless without turning index generation on and vice versa - `prodockit.bibliography`'s own docs already combine marking and generation into one page. Moved the generation content into `extensions/index-terms.md` as a new "Generating the index" section, merged the per-feature rendered-output examples into their existing marking sections instead of duplicating them, and renamed the page from "Index terms" to "Index (pdf-only)" now that it covers the whole feature. `pdf.md` keeps only a short pointer, matching how `prodockit.bibliography` is treated there. No code changes.

37.69 0.10.0 (2026-07-24)

`prodockit.bibliography`'s `\bibliography` marker now takes two optional, positional parameters - `\bibliography{<file>}{<true|false>}` - so a project can generate both a strict **References** section (only sources actually `\citebib{}`-cited in the text) and a broader **Bibliography** section (every entry, including background reading that's never individually cited) in one build, from the same or different `.bib` files:

```
←!— references.md —→
\bibliography{}{true}
```

```
←!— bibliography.md —→
\bibliography{background.bib}
```

Bare `\bibliography` is completely unchanged - fully backward compatible, no breaking change. A `\citebib{id}` citation now cross-links to whichever marker's page actually defines that entry (via a new, lightweight `.bib` entry-key discovery helper, not a CSL reimplementation) rather than assuming a single global bibliography page - the common single-file case is unaffected. See Multiple sections: References and Bibliography for the full syntax and worked examples.

Fixes #89.

37.70 0.9.0 (2026-07-24)

Breaking: `copyright` / `pdf_copyright` are now a real HTML fragment, rendered as a real DOM element in the PDF's running footer via CSS Paged Media's `position: running()` / `content: element()`, instead of being escaped into a CSS `content: "..."` string. This is what makes a real `<a href="../...">` link inside either value survive as a real, clickable link in the PDF - on every page, not just wherever the source element itself sits - matching how Zensical's own website-side `copyright` setting already works. Use a real `<br>` for a forced line break; the `\A` CSS-escape trick added in 0.8.0 only ever worked for a plain string, not real markup, and no longer applies - update any existing `pdf_copyright` using it to a real `<br>` instead.

`prodockit.pdf.css.build_css()` no longer takes a `copyright_text` parameter (`site_name` is unaffected, still a plain CSS content string) - no formal, versioned public API surface yet for `prodockit.pdf` (see prodockit-extension#7), so this is an acceptable break at this stage.

This project's own cover page (`docs/index.md`'s hero subtitle) no longer hyperlinks the word "Zensical" - it stays as plain text, matching this project's own PDF footer now crediting Zensical/prodockit with real links instead of the cover page doing it via a website-only, PDF-invisible link.

37.71 0.8.1 (2026-07-24)

Docs: this project's own docs site and PDF were missing the "Made with Zensical and prodockit" credit line that `overrides/partials/ copyright.html` / `pdf_copyright` (new in 0.8.0) already give a downstream project - added both here too, via a new `overrides/partials/ copyright.html` for the website and `extra.pdf_copyright` in `zensical.toml` for the PDF, so this site credits itself the same way a project built with it does. No library code changed.

37.72 0.8.0 (2026-07-24)

New `pdf_copyright` setting: `project.copyright` (a plain, native Zensical setting) already feeds the footer of both the website and the PDF by default - `pdf_copyright` is an opt-in override for the PDF's footer only, for a project that wants its PDF footer to say something different from its website's (e.g. adding a "Made with Zensical and prodockit" credit line only to the downloadable PDF, not the live site). Write a forced line break in either setting with a literal `\A` inside a TOML *literal* string (`'''...'''`) - see Copyright text for the full mechanism and why a literal string is required.

Also fixed a real, previously-undocumented rendering gap found while building this: a `\A` forced line break inside a `content` string only actually renders as a line break under `white-space: pre-line` - under WeasyPrint's default `white-space: normal` it silently collapsed to a plain space instead. Both the single-sided and double-sided verso copyright footer boxes now set `white-space: pre-line` so the forced break always works as expected.

37.73 0.7.1 (2026-07-24)

This project's own documentation site now enables every prodockit extension (`prodockit.headings`, `prodockit.refs`, `prodockit.citations`, `prodockit.glossary`, `prodockit.bibliography`, in addition to the `prodockit.tables` / `prodockit.index` already enabled) via `zensical.toml`, dogfooding the full set rather than just the two used to build this site previously.

Doing so surfaced a real bug: Zensical does not render pages in a stable order between builds, so a heading name shared across two or more pages (e.g. "Quick start", "Syntax", "Options") non-deterministically resolves its id collision differently from one `zensical build` to the next - confirmed by running repeated clean builds and observing the reported "keeping the first" winner change between runs. Fixed by giving every colliding heading across the docs an explicit, unique, page-prefixed id via `attr_list` (e.g. `## Quick start {: #refs-quick-start }`), rather than relying on build order at all. No library code changed - this is a docs-content-only fix, and not something a project sharing a heading name across its own pages will normally have to think about, since a one-off name collision is far less likely there than in this project's consciously-parallel per-extension documentation structure.

37.74 0.7.0 (2026-07-24)

Breaking: `prodockit.bibliography` now uses its own `\citebib{id}` syntax instead of `\cite{id}`. Previously it registered the same `\cite{id}` pattern `prodockit.citations` uses, at the same inline-pattern priority - enabling both extensions together left it undefined which one actually resolved a given `\cite{...}` occurrence. Renaming `prodockit.bibliography`'s own syntax removes the conflict entirely: both extensions can now be enabled in the same

build with no interference, each citing its own sources by its own marker. A project still using `prodockit.bibliography` on its own needs to update every `\cite{id}` in its source to `\citebib{id}` - the old syntax no longer resolves.

37.75 0.6.8 (2026-07-21)

`build_pdf_from_zensical_config()` (what `prodockit pdf` runs) now supports cover page markers, so a project no longer needs its own custom Python just to fill in a cover page's word count/repo URL/release tag - found via `prodockit-template`, whose `build_pdf.py` had grown to nearly nothing except this one piece:

- `{WORDCOUNT}` - the site-wide word count (the same value a `54,092` website macro shows).
- `{REPOURL}` - the git-detected repo URL.
- `{RELEASE}` - the latest published GitHub/GitLab release tag - the whole line is dropped instead if there isn't one.
- `prodockit` - substituted literally, since `prodockit pdf` never evaluates Jinja.

All four are opt-in by literally writing the marker in your `nav`'s index page - no new `zensical.toml` setting needed. See Cover page markers.

Also new: `pdf_extra_css`, a stylesheet meant *only* for the PDF (e.g. a rule that would look wrong on the live website), concatenated after `extra_css` - the same `["stylesheets/print.css"]` role a project's own custom PDF-build script might have hardcoded outside `zensical.toml` entirely before, now expressible as ordinary configuration.

Also fixed two real bugs found while building this:

- `extra_css`'s (and now `pdf_extra_css`'s) own relative `url(...)` references (e.g. a light/dark logo swap or a header background image) were passed through unresolved, pointing nowhere once compiled into the PDF's own temporary work directory - now resolved and base64-embedded, matching how a local `<img>` reference already was.
- `copyright` / `site_name` were passed straight into `build_pdf()`'s generated CSS `content: "..."` string with no escaping at all - `project.copyright` is commonly a triple-quoted TOML string spanning multiple lines, and a raw embedded newline (or a literal `"`) silently broke the whole generated rule, dropping the running header/footer entirely with no error. Both are now collapsed to one line and escaped before being passed through.

37.76 0.6.7 (2026-07-21)

Fixed `prodockit.pdf.html.fix_up_page_html()` permanently embedding *both* halves of a `#only-light` / `#only-dark` (or GitHub's `#gh-light-mode-only` / `#gh-dark-mode-only`) image pair in a PDF, stacked one after the other, instead of just

one - found via `prodockit-template`'s own cover page hero graphic showing twice. A PDF has no light/dark toggle to make that convention meaningful, but `to_base64_data_uri()` already strips anything from `#` onward before resolving the file (to find the right one), so the resulting `data:` URI has no trace of the fragment left for any stylesheet to hide either half by. The `#only-dark` / `#gh-dark-mode-only` half is now dropped entirely rather than embedded.

37.77 0.6.6 (2026-07-21)

- Docs: the cover page hero graphic (`docs/assets/cover-hero-*.svg`) used a different colour palette in light mode (blue) than in dark mode (green) - recoloured the light variant to match dark exactly, so the hero reads the same regardless of theme. The "Download PDF" button also picked up this same green, rather than the theme's default primary colour.
- `prodockit.pdf.css`'s back-of-book index letter-group headings (`h2.prodockit-index-letter` - the "A", "B", "C" separators) were hardcoded to the hero graphic's *old* light-theme blue - updated to match the now-green hero, which a PDF always shows regardless of a project's own website light/dark toggle.
- No functional (Python package behaviour) changes beyond the index letter colour.

37.78 0.6.5 (2026-07-21)

Extends the 0.6.4 always-excluded-directory mechanism in `prodockit.pdf.source_bundle` to two more classes of vendored, never student-written content:

- Any directory literally named `styles` - a Vale `StylesPath` (conventionally named this way) holds downloaded rule packs (e.g. the Microsoft, proselint, and Readability style guides), typically tracked for offline/CI builds rather than gitignored.
- Common dependency lockfiles by exact file name - `package-lock.json`, `npm-shrinkwrap.json`, `yarn.lock`, `pnpm-lock.yaml`, `Pipfile.lock`, `poetry.lock`, `Cargo.lock` - machine-generated by a package manager, never hand-written, and often thousands of lines each.

Neither is project-configurable, matching 0.6.4's `.icons` exclusion: a project can't reach for the same knob to narrow what a bundle archives.

Also fixes `source_bundle.pdf`'s running header naming the wrong file: a file's own last page could show the *next* file's name instead of its own, because the invisible marker that sets the header text had no page break of its own - only the following content did - so it rendered on the tail end of the previous file's last page. The break now moves onto the marker itself, so the string-set and the page it applies to always agree.

37.79 0.6.4 (2026-07-21)

`prodockit.pdf.source_bundle` now always excludes any directory literally named `.icons` (e.g. a `custom_icons` directory, per `pymdownx.emoji`'s own convention) from `source_bundle.pdf`, regardless of `.gitignore` - found via `prodockit-template`, whose own vendored icon packs (`overrides/.icons/bootstrap`, `overrides/.icons/gitlab` - together ~2,500 unused SVGs) turned a source bundle meant to hold a student's own report content into a 3,000-page dump of unreferenced vendor assets. `.gitignore` alone can't fix this, since these directories are typically *tracked* (needed for the site/PDF to build at all) - deliberately not a project-configurable setting, so a project can't reach for the same knob to exclude something that should actually be archived.

37.80 0.6.3 (2026-07-20)

Bug fixes found by an in-depth test-coverage review of the extensions and PDF pipeline test suites - each paired with a new regression test.

- Fixed `prodockit pdf`'s CLI command showing a raw, unhandled traceback instead of a clean `Error: ...` message when `pdf_source_bundle` was enabled and the underlying `git`/`weasyprint` invocation failed - `SourceBundleError` wasn't in the CLI's caught exception tuple.
- Fixed `prodockit.headings` numbering a skipped heading level (e.g. h1 followed directly by h3, or a document starting below h1) with a literal "0" segment (e.g. "1.0.1") - a shallower level with no heading of its own yet is now treated as an implicit first one instead.
- Fixed `prodockit.pdf.mermaid` letting an uncaught `OSError`/`PermissionError` (e.g. a non-executable `mmdc` binary) escape instead of failing just that one diagram gracefully.
- Fixed `prodockit.pdf.source_bundle` crashing the whole bundle build on a file that's valid UTF-8 in the first 8 KiB sniffed to decide "is this text?" but not further in - now skipped like any other binary file instead.
- Fixed `prodockit.pdf`'s generated Lua filter producing broken syntax if a configured math/tex2svg path contained a quote or backslash - both are now escaped.
- Fixed `prodockit.pdf.build_pdf()` having no timeout on the underlying `pandoc`/WeasyPrint invocation, so a hang (e.g. a pathological CSS layout) could block the whole build indefinitely - added a `pandoc_timeout` parameter (default 30 minutes).
- Fixed a back-of-book index term nested more than three levels deep rendering with no extra indent at all, since the generated CSS only defines an indent step up to level 3 - now clamped to the deepest available indent instead.
- Substantially expanded test coverage across the extensions and PDF pipeline test suites (shared registries, cross-page linking, malformed input, table/index edge cases, icon/rotation/CSS edge cases) - no other functional changes.

37.81 0.6.2 (2026-07-20)

- Docs: fixed a real bug found by checking the live site after 0.6.1 - four spots in `docs/extensions/index-terms.md` / `docs/pdf.md` (plus two more in this same changelog) tried to show the code-styled `\index{}` syntax as literal example text using *inline* backticks around a hierarchical, code-styled path. Confirmed directly this doesn't work the way it does for the plain syntax - the code-styled pattern has to run before Python-Markdown's own backtick handling (see 0.6.0's own entry below), so inline backticks don't protect it, and the live site was rendering a raw internal Python-Markdown placeholder string instead of the intended literal text. Moved each one to a fenced code block (already documented as the safe way to show this syntax) or reworded to avoid the literal example entirely.
- No functional changes.

37.82 0.6.1 (2026-07-20)

- Docs: `prodockit.index` (new in 0.6.0) was missing from `README.md` - and so from PyPI's own project page - entirely: added it to the "Status" line and the extensions table, and mentioned the generated index alongside `prodockit.pdf`'s other PDF-only features. Also added it to `pyproject.toml`'s own `description` (PyPI's summary line) and `src/prodockit/__init__.py`'s module docstring, both of which had the same gap.
- No functional changes.

37.83 0.6.0 (2026-07-20)

- New `prodockit.index` extension: mark a term inline with `\index{Term}` for a traditional, PDF-only back-of-book index (browser/Ctrl-F search covers this on the live website, so there's no equivalent there) - the term displays inline exactly as written and is marked for indexing in one go, no separate "definition" step. Needed its own extension rather than the usual `attr_list` marker convention every other prodockit extension uses - confirmed directly plain `attr_list` can't wrap arbitrary inline text in a span on its own.
    - **Sub-entries**: `\index{Parent!Child!Grandchild}` (up to three levels deep in practice, matching LaTeX `makeidx`'s own `\index{primary!secondary!tertiary}` convention) nests related entries together instead of listing every term flat.
    - **Code-styled terms**: backticks around the last segment - or, combined with sub-entries, around just the last segment of a hierarchical path - mark a command/code term: it displays inline in a real `<code>` element, and the generated index entry renders the same way.
    - A term can be a markdown link or contain nested emphasis/code - confirmed directly neither needs special handling, since a term isn't exempted from Python-Markdown's own later inline-pattern passes the way `\ref{id}` / `\cite{id}` / `\gls{id}` are.

- New `prodockit.pdf.index`: the two-pass build (a term's own page number can only be known once WeasyPrint has already laid the PDF out once), configured by `prodockit.index`'s `include` and `title` settings - a traditional, two-column, letter-headed index page (matching this project's own cover page hero graphic colour), alphabetised ignoring leading punctuation (so `--set-upstream option` files under "S", not a separate symbols section), with consecutive pages collapsed into an en-dash range (`67–70`). Requires the new optional `pymupdf` dependency - `pip install prodockit[index]`.
- Fixed a real bug found while writing tests: code-styling a non-last segment of a hierarchical term - never a supported combination, but this shouldn't have corrupted anything either - used to leak a raw Python-Markdown internal stash placeholder into the generated index instead of failing gracefully - a real rendered PDF would have shown a nonsense category label instead of "Git".

37.84 0.5.0 (2026-07-19)

- New `prodockit.pdf.source_bundle`: bundles every text/source file `.gitignore` doesn't exclude into a separate `source_bundle.pdf` at a project's own top-level directory - 8pt Courier, wrapped lines, each file starting its own page, a running header (`site_name` on the left, that page's own file path on the right), and a "Page N of M" footer. Off by default; set `pdf_source_bundle = true` under `[project.extra]` to turn it on. Independent of the rest of `prodockit.pdf` - there's no Markdown involved, so it skips Pandoc entirely and hands a small, self-contained HTML document straight to WeasyPrint. File discovery shells out to `git ls-files --cached --others --exclude-standard` rather than reimplementing `.gitignore`'s own matching rules; text/ binary filtering is content-based, not by file extension.
- Docs: this site's own header now shows a PDF download icon next to "view" (an `overrides/partials/actions.html` override, linking to that page's own per-page PDF) instead of a "Download this page as PDF" text link at the top of the page - removed from every page that had one. Since the new icon is template markup rather than Markdown content, it also no longer shows up inside the PDF itself (no `.web-only` CSS trick needed, unlike the link it replaces).

37.85 0.4.2 (2026-07-19)

- Docs: matched more of this site's own theme config to prodockit-userguide's - the header's repo link now shows the actual GitHub logo instead of Zensical's default Git icon; the "View source of this page" button now shows an eye icon instead of a generic file icon; every admonition (e.g. the "tip" callout in `citations.md`) now uses the same custom FontAwesome icon set userguide uses instead of Zensical's own bundled defaults - this also feeds into `prodockit.pdf`'s own admonition icons, so PDF output picks it up too;

added the matching theme features userguide already had (`content.tabs.link` in particular actually affects this project's own tabbed content); and swapped the palette toggle icons to match userguide's own light/dark convention.
- No functional (Python package) changes.

37.86 0.4.1 (2026-07-18)

- Docs: reworked this site's own chrome to match prodockit-userguide's - a new split hero cover page ("Home"), reusing that project's own logo/ favicon/ illustration assets; top-level nav moved to a top tab bar with the right-hand page TOC merged into the left sidebar instead; the previous cover page's own prose moved to a new "Introduction" page.
- Fixed a real bug found along the way: `zensical.toml`'s own `copyright` was a triple-quoted, multi-line TOML string - `prodockit.pdf` substitutes it verbatim into a CSS `content: "..."` string for the PDF's running footer, and the embedded newline silently broke that declaration, dropping the whole footer with no error (this site's own PDF footer had no copyright text at all). Fixed to a single-line string, and switched `&copy;` to a literal `©` character - a CSS content string doesn't decode HTML entities either.
- Fixed the deploy workflow missing a per-page PDF build step for the new Introduction page (its own "Download this page as PDF" link 404'd), and that page's leftover PDF link (still the old cover page's, pointing at the whole-site PDF) to the same per-page convention every other content page already uses.
- No functional (Python package) changes.

37.87 0.4.0 (2026-07-18)

- New `pdf_double_sided` option: a duplex-printing layout for book/ handbook-style documents printed and bound on both sides. Verso (left-hand) and recto (right-hand) pages mirror their header/footer content and page margins (new `pdf_margin_inner`/`pdf_margin_outer`, replacing `pdf_margin_left`/`_right` in this mode) via CSS Paged Media's `@page :left`/`:right` selectors - chapter title and page number always on the outer, fore-edge corner; site name and copyright always on the inner, spine-side corner, whichever physical side that is for a given page. Every numbered heading now starts its own recto page (`break-before: recto`, auto-inserting a blank page as needed - confirmed directly this needs no Python-side page-counting logic at all), and a `prodockit-table-rotated` landscape page's own rotation direction now alternates by its final page position (270 degrees on recto, 90 on verso - the spine sits on the opposite physical side either way).
- New `recto_title` front matter key: overrides a page's own running header text with a shorter title, from the *next* page onward (the heading's own page still shows its full title - confirmed directly this is a consequence of CSS `string()`'s "first value on this page wins" default policy) - useful for a

chapter title too long to fit comfortably in the header, with or without `pdf_double_sided`.
- Off by default: a single-sided build is completely unchanged.

37.88 0.3.1 (2026-07-18)

- Docs: renamed `glossary.md`'s heading to "Acronyms and Glossary" and `citations.md`'s to "Citations or References" (and their matching nav labels); added a flow diagram to `bibliography.md`'s Requirements section (and fixed a real, unrelated gap found along the way - this docs site had no Mermaid `custom_fences` config at all, so a plain ```` ```mermaid ```` fence never rendered as a diagram anywhere on the site); switched the citation-style example to `harvard-cite-them-right.csl`; added an admonition pointing from `prodockit.citations` to `prodockit.bibliography`; and noted `prodockit.bibliography`'s own independent Pandoc invocation in `prodockit.pdf`'s "Limitations and workarounds".
- Docs: updated `README.md` (and so PyPI's own project page description) to include `prodockit.tables` / `prodockit.bibliography`, and to mention sideways tables/ `.web-only` / `.pdf-only` under PDF generation - it had gone stale since both extensions shipped in 0.3.0.
- No functional changes.

37.89 0.3.0 (2026-07-18)

- New `prodockit.bibliography` extension: an alternative to `prodockit.citations` for a `.bib`-backed reference list instead of a hand-authored one. Define sources in a BibTeX/BibLaTeX `.bib` file, cite them with the same `\cite{id}` syntax, and get the resolved citation text and a full, auto-generated reference list formatted in any Citation Style Language (CSL) style (APA, IEEE, Harvard, ...) via Pandoc's own `--citeproc` - confirmed directly against real Pandoc output rather than reimplementing citation formatting, and rejected an actual LaTeX/biblatex toolchain as a new hard dependency along the way. Makes `pandoc` a required dependency for this extension specifically, including for a website-only build with no PDF. New `docs/extensions/bibliography.md` includes a "References and Bibliography" comparison of this, `prodockit.citations`, and what `prodockit-template` / `prodockit-userguide` currently do.
- New sideways (90-degree anticlockwise) tables in the PDF: wrap a table and its own caption in `<div class="prodockit-table-rotated" markdown="1">` to print it on its own landscape-sized page(s), spanning multiple pages with a repeated heading row exactly like any other table. Confirmed directly that a CSS `transform: rotate()` doesn't work for this (clips the table to one page and loses its heading row) - the actual rotation is applied afterwards via a `/Rotate` post-process on the finished PDF (new `prodockit.pdf.rotate` module, new `pypdf` dependency).
- `.web-only` content is now hidden in every PDF build automatically, via `prodockit.pdf.css`'s own always-included stylesheet - no project-side CSS

needed any more. `.pdf-only` is documented as a one-line, centrally-sourced snippet instead (`prodockit` has no way to reach into a project's own website stylesheet), in a new "Web-only / PDF-only content" section in the PDF generation docs.

37.90 0.2.0 (2026-07-18)

- New `prodockit.tables` extension: gives a table column a percentage or fixed width via a `width` attribute already attachable to a header cell with `attr_list` - no new syntax. Column-width distribution beyond what's explicitly given is left to CSS's own `table-layout: fixed` algorithm rather than computed in Python. Ships with the matching CSS in `prodockit.pdf`'s generated stylesheet, and documents the equivalent rule a project's own website theme needs (see the new Tables docs page).
- New `prodockit pdf --markdown-file` / `-m` option: builds a PDF from a single markdown file instead of the whole `nav`, using the same `zensical.toml` settings as a full build.
- `prodockit.pdf`'s generated table CSS now draws a full grey 0.5pt grid - outer border and internal row *and* column lines (there was previously no line between columns at all) - and reads a project's own `extra_css` (from `zensical.toml`), so a project-specific `@media print` rule (e.g. hiding a website-only "Download PDF" link/button) also applies in the PDF.
- `prodockit.citations`: a resolved `\cite{id}` link now always gets `class="prodockit-cite-resolved"` (previously no class at all), matching `prodockit.refs` / `prodockit.glossary`'s existing convention of a stable class for both the resolved and unresolved case.
- Docs: added a "CSS hooks" section to `refs.md` / `citations.md` / `glossary.md` (`headings.md` already had one), documenting every class/attribute each extension itself emits; replaced the docs site's "edit this page" link with "view this page" (a `content.action.view` link to the raw source rather than a GitHub edit form); added a whole-site PDF download button on the front page and a per-page download link on every other page, both built via the new `--markdown-file` option above.
- Fixed `prodockit.__version__` reporting a stale `"0.10.0"` (left over from before the `zendoc` → `prodockit` rename) instead of matching this package's actual, `pyproject.toml`-declared version.

37.91 0.1.1 (2026-07-17)

- Docs: reworded the package intro on the docs site and README (dropped the `pymdown-extensions` comparison, added a mention of the website macros and a one-line "kit for professional documentation" summary) - no functional changes.

37.92 0.10.0 (2026-07-15)

- New `prodockit.zensical_macros`: Jinja variables/macros for Zensical's own macros plugin - `54,092` (site-wide prose word count, excluding the cover page and any page flagged `exclude_from_word_count: true`), `https://github.com/buckwem/prodockit-extensions` (git-detected repository URL), `prodockit`, and `heading_counter_reset(page)` / `reference_style()` / `acronym_style()` / `glossary_style()` macros. Add it alongside a project's own `macros.py` via `zensical.toml`'s `modules = ["prodockit.zensical_macros"]` - or use it alone if the project has no macros of its own.
- New `prodockit.wordcount`: the generic prose word-count utility (`count_words()` / `compute_word_count()`) behind both `prodockit.pdf`'s `{WORDCOUNT}`-style cover-page use and `prodockit.zensical_macros`' `54,092` - previously duplicated independently by each downstream project needing both.
- New `prodockit.settings`: `flatten_nav()`, `heading_numbering_enabled()`, and `reference_style_values()` - the `project.extra.*` reading shared by `prodockit.pdf.config` and `prodockit.zensical_macros`, so the two agree on one set of fallback defaults instead of each hand-maintaining its own copy. `prodockit.pdf.config.build_pdf_from_zensical_config()` now uses these too (previously inlined), and its `pdf_math_dir` setting is now created automatically if configured to a directory that doesn't already exist (matching the auto-detected default's existing behaviour).

37.93 0.9.0 (2026-07-15)

- New `prodockit pdf` command: builds a complete PDF with no Python required, reading everything - nav, docs directory, fonts, page size, and all PDF-specific settings - from the project's own `zensical.toml`, the same way `zensical build` / `zensical serve` do. Installing `prodockit` now registers a `prodockit` console script (`pip install prodockit` is enough - no separate build script to write). See the new `prodockit.pdf.config` module (`build_pdf_from_zensical_config()`) for the config-to-`build_pdf()` orchestration this wraps: nav-tree flattening, per-page `is_appendix` front-matter detection, and auto-detection of a local `mmdc` (Mermaid) binary and MathJax `tex2svg` script, so a typical project needs no extra configuration beyond what it likely already has.
- `build_pdf()` gained `include_table_of_contents` / `table_of_contents_title` parameters (both used automatically by `prodockit pdf`): a generated table of contents is now inserted by default, right after a cover page if one is marked `is_index=True`, or at the very start otherwise.
- Rewrote the PDF generation docs page around the `prodockit pdf` command as the primary, and for most projects only necessary, way to use

`prodockit.pdf` - `build_pdf()` and the individual pipeline pieces are now documented as the advanced, scripting-your-own-pipeline path.

37.94 0.8.0 (2026-07-15)

- New `prodockit.pdf.build_pdf()`: a one-call convenience wrapper around the rest of `prodockit.pdf` - hand it a list of already-rendered pages (`prodockit.pdf.Page`) and where to write the PDF, and it fixes up each page's HTML, generates the Lua filter and CSS, concatenates everything, and runs `pandoc`/WeasyPrint for you. Takes `output_path` (the PDF's own destination path) plus font/page-size/margin/header-footer/reference- style/numbering/math parameters, all with sensible defaults. Raises the new `prodockit.pdf.PdfBuildError` (with the underlying `pandoc` exit code and stderr attached) if the build fails, rather than failing silently. `prodockit.pdf.html`/`.lua`/`.css`/`.icons`/`.mermaid` remain directly importable if you need more control over how the pieces fit together.
- Rewrote the PDF generation docs page around `build_pdf()` as the primary documented way to use `prodockit.pdf`, leading with a short, practical quick-start example rather than the implementation-level detail of how Pandoc/WeasyPrint's own quirks are worked around (that detail is still there, now further down, for anyone who wants it).

37.95 0.7.0 (2026-07-15)

- New `prodockit.pdf`: a Pandoc/WeasyPrint pipeline for building a standalone PDF from Zensical-rendered HTML - not a Python-Markdown extension (no `markdown.extensions` entry point), a plain function library, since a PDF build pipeline isn't a Markdown syntax extension:
  - `prodockit.pdf.html`: `fix_up_page_html()` and link/anchor/image helpers
  - fixes up one page's already-rendered HTML for Pandoc's own reader/writer quirks (attribute loss on `<p>`, raw `<svg>` not surviving the round trip to WeasyPrint, footnote/caption structural mismatches, cross-page link rewriting for a concatenated multi-page PDF, and more).
  - `prodockit.pdf.lua`: `build_lua_filter()` - chapter/appendix numbering, caption chapter-prefix numbering, tabbed-set reconstruction, and MathJax pre-rendering, generated as a parameterized Lua filter.
  - `prodockit.pdf.css`: `build_css()` - the compiled CSS a PDF needs on top of a project's own website stylesheet, including WeasyPrint-specific page-break tuning for headings, paragraphs, tables, code blocks, figures/captions, admonitions, and grid cards.
  - `prodockit.pdf.icons` / `prodockit.pdf.mermaid`: admonition icon resolution and Mermaid diagram pre-rendering, as standalone helpers.
- Fixed a real bug found while writing tests: the iframe→"Watch Video" admonition link builder stripped the video id from every single conversion (a

replace-then-split ordering removed the just-added `?v=...` too) - now produces a working YouTube watch link.
- No formal, versioned public API surface yet (see prodockit-extension#7) - import whatever's needed directly, the same informal way as the rest of this package.
- New dependency: `beautifulsoup4` (>= 4.12).
- Broadened the package's own description: prodockit is now framed as a family of extensions for Zensical needed for professional and academic documentation, rather than "Python-Markdown extensions" specifically - `prodockit.pdf` isn't one, and the framing was due to broaden anyway now that PDF generation is in scope alongside cross-references/citations/glossary.

37.96 0.6.0 (2026-07-14)

- `prodockit.headings`: new `numbering="continuous"` option (Zensical only) - `h1` numbering carries on from wherever the previous nav page left off, instead of restarting at 1 on every page. Fixes `\ref{id}` showing the wrong number for a heading on a different page (it previously always showed that page's own per-document number, not the number actually displayed on the page - see zendoc-template#89).
- New `appendix_attr` option (default `is_appendix`): a page whose front matter sets this flag is numbered with a letter instead - "A", "A.1", "A.1.1" - and doesn't consume a number from the numeric sequence, so later pages aren't left with a gap. Letters are assigned sequentially in nav order.
- New public `prodockit.headings.prescan(appendix_attr="is_appendix")` function: returns the same `(start_counts, appendix_letters)` pre-scan `HeadingsExtension` uses internally, for a consuming project's own build tooling (e.g. a template macro driving a presentational CSS counter-reset) to stay in sync automatically rather than re-deriving the same page-order/heading-count logic independently.

37.97 0.5.1 (2026-07-14)

- `prodockit.glossary`: a resolved `\gls{id}` now always renders with `class="prodockit-gls"` (previously it had no class at all), matching `prodockit.refs`' always-present base class. The unresolved case now renders `class="prodockit-gls prodockit-gls-unresolved"` (previously just `prodockit-gls-unresolved`, missing the base class), so a stylesheet has one stable hook (`.prodockit-gls`) regardless of resolution state, with `.prodockit-gls-unresolved` layered on top only when needed.

37.98 0.5.0 (2026-07-14)

- New `prodockit.glossary` extension: define a term once via `attr_list` (an id plus a `data-term` short display string), then insert it by id from anywhere

with `\gls{id}`, which resolves to the term's own text, linked to its definition - e.g. `\gls{css}` → `CSS`. Unlike `prodockit.citations`' `\cite{id}` (which generates new bracketed citation text), `\gls{id}` inserts the term's own registered text in place - closer to LaTeX's `glossaries` package.
- One shared `GlossaryRegistry` covers both acronym-style and glossary-style entries - they're the same kind of thing (an id with a short display text), so acronym and glossary pages can reference each other, or be referenced from any other page, with no special wiring.
- Supports forward references within a document, an `unresolved` marker (`?` by default) for an unknown id, and the same automatic Zensical cross-page registry sharing and nav pre-scan (for citing/using a term before its defining page has been converted) that `prodockit.citations` got in 0.4.0.
- Refactored the nav pre-scan logic (previously private to `prodockit.citations`) into a shared, generic `prodockit._zensical.preseed_attr_from_nav` helper, since `prodockit.glossary` needed the identical scan.

37.99 0.4.0 (2026-07-14)

Fixes found migrating a real multi-page site's references page to `prodockit.citations` for real - all discovered by actually building a real multi-page site, not just single-document tests:

- **Fixed a real correctness bug**: `prodockit.refs` / `prodockit.citations` were emitting a bare `#id` fragment for *every* resolved link, including a cross-page one - which only works by coincidence in a single concatenated PDF document, but 404s on an actual multi-page website (an `#id` fragment only navigates within the *current* page). Both now emit a real relative link (e.g. `references.md#id`, correctly adjusted for the citing page's own directory depth) when the target is on a different page, which Zensical already knows how to rewrite into the right clean URL - the same way a hand-typed cross-page Markdown link already works.
- New: `prodockit.citations` pre-scans every page in a Zensical build's nav for citation definitions before any page is actually converted, so citing a source *before* it's defined - the common case, since a references page is usually kept at the end of nav as an appendix - resolves correctly in a single `zensical build` pass, rather than only working from `zensical serve`'s live-reload. New `CitationRegistry.preseed()` method backs this; a real registration always supersedes a preseeded stub.
- `RefsExtension` gained a `source` option (mirroring `HeadingsExtension`'s), needed for the same-page-vs-cross-page link decision above.
- Fixed the nav pre-scan matching a citation-definition attr_list example shown literally inside a fenced code block in documentation - it now skips fenced content, the same protection `CitationDefTreeprocessor` already gets for free from the real Python-Markdown parser.

37.100 0.3.0 (2026-07-14)

- New `prodockit.citations` extension: define a source once via `attr_list` (an id plus a `data-cite-text` short display string), then cite it by key from anywhere with `\cite{id}` (or `\cite{id1,id2,...}` for multiple), auto-generating a bracketed, linked citation - `[Skoulikari, 2023]` - instead of hand-typing the link and text at every citation site.
- Supports forward references within a document, an `unresolved` marker (`?` by default) for an unknown key, and the same automatic Zensical cross-page registry sharing (with soft-fail on key collisions) that `prodockit.headings` / `prodockit.refs` got in 0.2.0.
- Auto-generating the references page's own listing from structured bibliographic data isn't built yet - see the extension's docs for the current scope.
- Fixed the `zensical.toml` installation examples in the docs: nested `[project.markdown_extensions.prodockit.headings]` tables don't work (Zensical only hoists the `pymdownx` / `zensical` namespaces that way) - the quoted-key form (`[project.markdown_extensions."prodockit.headings"]`) is required.

37.101 0.2.0 (2026-07-14)

- `prodockit.headings` / `prodockit.refs` now share their registry automatically under Zensical, without any explicit `registry` / `source` configuration: each extension detects Zensical's per-page rendering context and derives a stable `source` from the page's own path, fixing cross-page `\ref{id}` references not resolving.
- A heading id collision across two different sources, when detected via this automatic Zensical sharing, now logs a warning and keeps the first registration instead of raising `DuplicateIdError` - so two unrelated pages that happen to share a heading title (e.g. both have an "Overview" section) no longer break the build. Explicitly-shared registries (the manual multi-page pattern) still raise on a collision, unchanged.
- Fixed an extension-ordering bug: `prodockit.headings` and `prodockit.refs` now find and share each other's registry regardless of which order they're listed in - previously, only `prodockit.headings`-then-`prodockit.refs` worked reliably, and Zensical's own TOML-to-extension-list conversion doesn't preserve list order at all.

37.102 0.1.0 (2026-07-14)

Initial release.

- `prodockit.headings`: heading ids and hierarchical section numbering, backed by a shared `IdRegistry`.

- `prodockit.refs`: `\ref{id}` section cross-references, resolving to the target's current section number, including forward references within a document and across a shared registry.
- Documentation site built with Zensical, published at buckwem.github.io/prodockit-extension.

38. Licence

prodockit is distributed under the MIT License. In plain-language summary, you may use, copy, modify, publish, and redistribute it, including in commercial work, provided that the copyright and licence notice remain with the software. The software is supplied without warranty.

This summary is explanatory only and does not replace the authoritative legal text below.

Index

R

S

T

V

W

Z