

Table of Contents

1. Prepare to install	2
1.1 Prepare Python and its environment	2
1.2 Understand the project structure	7
1.2.1 A clean Zensical site	7
1.2.2 The site after adding Prodockit	7
1.2.3 A site created from prodockit-template	8
1.3 Continue with an installation route	9

1. Prepare to install

Every installation route begins in the parent directory where you keep your repositories and needs the same supported Python release. Prepare Python and a setup virtual environment there in section 3.1, then continue with the route that matches the work: [adoption](#) for an established document, [bootstrap](#) for a new machine and template project, [the template-project guide](#) for the supplied project structure, or the [first-site walkthrough](#) for an empty directory.

Section 3.1 is the shared preparation for those routes. Each route then owns its project directory, project-local environment, Prodockit installation, and verification rather than repeating that work here.

1.1 Prepare Python and its environment

Python must exist before it can create the environment that runs Prodockit. Always complete this section in the directory that holds all your repositories, for example `~/repos`, `~/github`, `~/gitlab`, or `C:\Users\your-name\github`. Do not enter an individual project yet.

The setup `.venv` keeps the initial tools separate from system Python and avoids the `externally-managed-environment` error produced by package-managed Python installations under PEP 668. Bootstrap uses this setup environment to create or prepare a project. Adoption and the first-site walkthrough later enter their project directory and create or replace that project's own `.venv`; those important transitions are shown in their own steps rather than hidden here.

Use the badges in this section as a key: **Privileged** means installing system software needs administrator or `sudo` access; **Optional** means skip a step when its software is already available or the feature is not needed. Neither badge applies to commands run inside your own virtual environment.

1. Install Python 3.14 **Privileged Optional**

Python 3.14 must be available before creating an environment. Installing it is optional if it is already present, but a new system installation requires the necessary administrator privileges. Verify the installed version even when you skip installation.

macOS

If Homebrew is not installed, use its official installer. Follow every post-install instruction it prints so that `brew` is added to your shell.



[Install Homebrew](#)

After Homebrew finishes installing, close Terminal completely and reopen it. The current terminal will not know about the new `brew` command.

In the reopened terminal, check that Homebrew is available:

```
brew --version
```

Install and verify Python 3.14:

```
brew install python@3.14  
"$ (brew --prefix python@3.14)/bin/python3.14" --version
```

Windows

Install the 64-bit Python 3.14 release from the official Python website:

 [Install Python](#)

Select **Add python.exe to PATH** and **Disable path length limit** in the installer, then open a new PowerShell window and run:

```
py -3.14 --version
```

If `python` opens the Microsoft Store, disable its `python.exe` and `python3.exe` App Installer aliases.

Linux (Ubuntu)

```
sudo apt update  
sudo apt install python3.14 python3.14-venv python3-pip  
python3.14 --version
```

Every check must report Python 3.14 before you continue.

2. Create the virtual environment

First choose the parent directory that will hold your Git repositories. Keeping projects under one parent gives Bootstrap a predictable place to create a new project and makes it clear that this first `.venv` is a setup environment, not the environment belonging to one particular site.

 Create a repositories directory if this is your first one

If you have not worked with a Git repository before, create one top-level directory for all your repositories. `repos` is a neutral name; `gitlab` or `github` can be useful when you prefer to group projects by host. Keep using an existing repositories directory if you already have one, and replace `repos` in the examples with its name. Lowercase names are quicker to type. After creating the directory, type the first few characters of its name and press `Tab` to let the terminal complete the rest.

Create or enter the repositories directory:

macOS

```
mkdir -p ~/repos
cd ~/repos
```

Windows

```
New-Item -ItemType Directory -Force ~\repos | Out-Null
Set-Location ~\repos
```

Linux (Ubuntu)

```
mkdir -p ~/repos
cd ~/repos
```

Next create the setup virtual environment in that directory. Python stores it in a folder named `.venv` alongside, rather than inside, the individual repository folders that will be created later.

macOS

```
"$(brew --prefix python@3.14)/bin/python3.14" -m venv .venv
```

Windows

```
py -3.14 -m venv .venv
```

Linux (Ubuntu)

```
python3.14 -m venv .venv
```

Creating the environment does not activate it or change system Python.

3. Activate the environment

Activate `.venv` in every new terminal before installing or running the documentation tools.

macOS

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

Windows

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

The policy applies to the current account and may ask for confirmation. To leave it unchanged, use classic **CMD** and run `.\.venv\Scripts\activate.bat` instead.

Linux (Ubuntu)

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

The shell prompt normally gains a `(.venv)` prefix.

4. Verify the active environment

Verify both the version and the interpreter selected by the shell.

macOS

```
python --version  
command -v python
```

Windows

```
python --version  
Get-Command python
```

Linux (Ubuntu)

```
python --version  
command -v python
```

The version must report Python 3.14 and the executable path must be inside the parent repositories directory's `.venv`. If `python` resolves to an alias, follow Step 5, then repeat this check; activating the environment again will not remove the alias. If a check points elsewhere without an alias, repeat the activation step. The route you follow next will say when to keep using this setup environment and when to create or activate a project-local one.

5. Fix the Python alias when needed **Optional**

If Step 4 detects a `python` alias that overrides `.venv`, remove it in the current Bash terminal.

```
unalias python 2>/dev/null || true
```

This corrects the current terminal only. Repeat Step 4 now; its version, executable and prefix should all point to the activated `.venv`. If the prefix is still wrong and no alias remains, reactivate the environment and repeat Step 4.

To make the fix persist for future interactive Bash logins, optionally run this command once. It appends the line to your account's `~/.bashrc` only if it is not already present, preserving the file's existing content:

```
grep -qxF 'unalias python 2>/dev/null || true' ~/.bashrc 2>/dev/null ||
printf '\n%s\n' 'unalias python 2>/dev/null || true' >> ~/.bashrc
```

To verify the `~/.bashrc` change, log out and back in, activate `.venv`, then repeat Step 4. If you skip the persistent change, repeat the current-terminal command in Step 5 whenever a new login restores the alias. Prodockit does not change your shell setup.

1.2 Understand the project structure

The installation route changes what is stored in a site's project directory. It helps to recognise the starting structure before choosing a route or reviewing changes made by Prodockit.

1.2.1 A clean Zensical site

Running `zensical new .` in an empty project directory creates a small, working site. Its Markdown pages live under `docs/`, `zensical.toml` controls the site, and the supplied GitHub workflow can publish it. The project's virtual environment, `.venv`, is local working state and is not committed to Git, so it is not shown in the tree.

- 📁 `.github` — GitHub repository configuration
 - 📁 `workflows` — automated workflows
 - 📄 `docs.yml` — Zensical's GitHub Pages workflow
- 📁 `docs` — Markdown source pages
 - 📄 `index.md` — starter home page
 - 📄 `markdown.md` — starter Markdown example
- 📄 `zensical.toml` — Zensical project configuration

This is a complete Zensical starting point. Build and preview it successfully before adding Prodockit so that any earlier environment or Zensical problem is kept separate from the adoption work.

1.2.2 The site after adding Prodockit

`pdk adopt --apply` retains the Zensical pages and workflow, then adds the selected Prodockit components and records their supported toolchain. It also updates `zensical.toml` to enable the standard authoring extensions and load the website and PDF stylesheets in their intended override order.

- 📁 `.github` — original GitHub repository configuration
 - 📁 `workflows`
 - 📄 `docs.yml` — original Zensical workflow
- 📁 `docs` — original Markdown source pages and Prodockit assets
 - 📁 `javascripts` — Prodockit and project website behaviour
 - 📁 `vendor`
 - 📁 `mathjax`
 - 📄 `LICENSE` — vendor licence supplied with MathJax
 - 📄 `tex-svg-full.js` — vendor MathJax browser bundle
 - 📄 `extra.js` — USER-MANAGED website behaviour
 - 📄 `mathjax.js` — generated Prodockit configuration when maths is selected
 - 📄 `pdk.js` — managed Prodockit website behaviour

- └─  stylesheets
 - └─  extra.css — USER-MANAGED website overrides
 - └─  pdk-pdf.css — managed Prodockit PDF styles
 - └─  pdk.css — managed Prodockit website and component styles
 - └─  print.css — USER-MANAGED PDF-only overrides
- └─  index.md — original starter home page
- └─  markdown.md — original starter Markdown example
-  .prodockit-components.toml — optional component choices added by Adopt
-  .prodockit-toolchain.toml — supported tool versions added by Adopt
-  .python-version — supported Python release added by Adopt
-  pdf-requirements.txt — PDF-only Python packages prepared on first use
-  pdk-pdf.toml — PDF-only policy created or migrated by Adopt
-  requirements.txt — supported Python packages added by Adopt
-  zensical.toml — original configuration updated by Adopt

For the website, `zensical.toml` loads `pdk.css` first and `extra.css` last, so your rules can override the managed defaults. PDF generation continues the cascade with `pdk-pdf.css` followed by `print.css`. Adopt creates a missing user-managed stylesheet, but never replaces one you have edited.

Adopt installs managed `pdk.js` before the empty user-managed `extra.js`. When mathematical notation is selected, its generated configuration and vendor bundle are loaded between those two files. The MathJax licence is retained beside the vendored bundle.

These are the principal files in the clean-site route, not an exhaustive list of everything a mature project may contain. Adopt preserves existing content and project-owned configuration.

1.2.3 A site created from prodockit-template

The maintained template starts with Prodockit's publishing, authoring, and rendering structure already connected. It contains more files than a clean Zensical site because Bootstrap is preparing a complete working project rather than adding selected components to an existing one.

- └─  .github — GitHub repository configuration
 - └─  workflows — GitHub Actions workflows
 - └─  docs.yml — GitHub Pages build and deployment
 - └─  release-redeploy.yml — rebuild after a template release
- └─  docs — content and appearance
 - └─  assets — cover, branding, and report images
 - └─  javascripts — Prodockit and project website behaviour
 - └─  vendor
 - └─  mathjax
 - └─  LICENSE — vendor licence supplied with MathJax
 - └─  tex-svg-full.js — vendor MathJax browser bundle
 - └─  extra.js — USER-MANAGED website behaviour
 - └─  mathjax.js — generated Prodockit MathJax configuration
 - └─  pdk.js — managed Prodockit website behaviour
 - └─  stylesheets — website and PDF presentation
 - └─  extra.css — USER-MANAGED website overrides
 - └─  pdk-pdf.css — managed Prodockit PDF styles
 - └─  pdk.css — managed Prodockit website and component styles
 - └─  print.css — USER-MANAGED PDF-only overrides

-  `template.css` — template-specific website presentation
-  `1-originality.md` — originality and AI-use statement
-  `2-executive-summary.md` — starter executive summary
-  `3-requirements.md` — requirements section
-  `4-solution-architecture.md` — solution architecture section
-  `5-governance.md` — governance section
-  `6-operations.md` — operations section
-  `7-examples.md` — extension examples
-  `acronyms.md` — acronym list
-  `bibliography.md` — generated bibliography page
-  `glossary.md` — glossary
-  `index.md` — report cover
-  `references.md` — formatted reference list
-  `overrides` — Zensical theme customisations
-  `tools` — pinned Mermaid and MathJax Node tooling
-  `.gitignore` — generated and local files excluded from Git
-  `.gitlab-ci.yml` — GitLab Pages build and deployment
-  `.prodockit-shared-files.toml` — managed shared-file checksums
-  `.python-version` — supported project Python
-  `bibliography.bib` — example bibliography source
-  `macros.py` — shared template macros
-  `pdf-requirements.txt` — PDF-only Python dependencies prepared by pdk pdf
-  `pdk-pdf.toml` — PDF-only settings and runtime policy
-  `README.md` — project summary and publishing badges
-  `references.bib` — example hand-written reference source
-  `requirements.txt` — Python build dependencies
-  `zensical.toml` — site, navigation, and shared authoring settings

This is the useful project-facing structure rather than every file in the template repository. The template manifest is the authoritative record of which files Template Sync manages, shares with the project, leaves to the author, or excludes.

In a template site, `template.css` sits between `pdk.css` and `extra.css` in the website cascade. PDF generation then adds `pdk-pdf.css` and finally the user-managed `print.css`.

1.3 Continue with an installation route

The shared preparation is complete. Choose the card that matches what you are starting with; each route takes over from the parent repositories directory and explains when to enter or create the project itself.

Adopt prodockit

Start with an empty directory, create and test a Zensical site, then use Adoption to add the Prodockit features you select.

→ [Open section 4](#)

Build a template site

Let Bootstrap prepare the machine, repository, build tools, and maintained `prodockit-template` as one recoverable process.

→ [Open section 5](#)

Build site manually

Perform and verify the machine, repository, editor, renderer, and project setup yourself instead of asking Bootstrap or Adoption to orchestrate it.

→ [Open section 6](#)