Sphinx Stack documentation

© 2026 Canonical Ltd.
All rights reserved.

Canonical

Contents

1 In this documentation **3**
- 1.1 Getting started . . . 3
- 1.2 Document elements . . . 3
- 1.3 Customization . . . 3
- 1.4 Quality . . . 4
- 1.5 Lifecycle management . . . 4
- 1.6 How the stack works . . . 4

2 How this documentation is organized **5**

3 Project and community **6**
- 3.1 Get involved . . . 6
- 3.2 Releases . . . 6
- 3.3 Governance and policies . . . 6

The **Sphinx Stack** is a well-crafted template repository for Sphinx projects, and is the standard documentation setup at Canonical.

The repository contains a setup for Sphinx and its required extensions, plus basic scaffolding for standard document types. A CLI is provided for building the documents and checking their quality.

The Sphinx Stack contains everything you need to bootstrap and host new documentation. You can copy it as a standalone documentation repository, or incorporate into a project's codebase.

The Sphinx Stack is for all employees at Canonical, and is available to writers and developers looking to begin a robust and capable Sphinx project.

1. In this documentation

1.1. Getting started

First project:

- *Set up a new project* (page 6)
- *Configure your project* (page 11)

Build and publish:

- *Build and preview* (page 12)
- *Publishing on Read the Docs* (page 20)

1.2. Document elements

Syntax guides:

- *MyST for Markdown* (page 113)
- *reStructuredText* (page 97)

Diagrams and tables:

- *Mermaid diagrams* (page 72)
- *Interactive tables* (page 86)

Discoverability:

- *Manage sitemaps* (page 70)
- *Enable Google Analytics* (page 52)

1.3. Customization

Templates and page configuration:

- *Custom HTML templates* (page 66)
- *Page-specific configuration* (page 70)

API documentation:

- *API docs from Python docstrings* (page 81)
- *OpenAPI specifications* (page 83)

PDFs:

- *PDF output* (page 54)

1.4. Quality

Quality checks:

- *Documentation checks* (page 15)
- *Testing documented commands with Spread* (page 60)

Stable and outbound links:

- *Redirects for moved pages* (page 45)
- *Intersphinx links to other doc sets* (page 87)

Continuous integration:

- *GitHub workflows* (page 90)
- *Bridging project and docs builds* (page 46)

1.5. Lifecycle management

Upgrade paths:

- *Upgrade from a new version* (page 25)
- *Upgrade from the legacy version* (page 28)

Troubleshooting:

- *Build errors* (page 37)
- *Local deployment issues* (page 40)
- *Runtime errors* (page 40)
- *Read the Docs failures* (page 41)

1.6. How the stack works

Core components:

- *Sphinx Stack structure* (page 128)
- *The build process* (page 132)

Default settings:

- *Default Sphinx extensions* (page 95)
- *Default Sphinx configuration* (page 91)
- *Sitemaps* (page 135)

2. How this documentation is organized

This documentation uses the Diátaxis documentation structure[1].

- The *tutorial* (page 6) walks through copying the Sphinx Stack template, removing unneeded files, and performing the required initial configuration before the first build.
- *How-to guides* (page 11) cover specific tasks: building locally, running documentation checks, publishing on Read the Docs, updating the stack, and enabling optional extensions.
- *Reference* (page 90) provides the list of default Sphinx extensions, GitHub workflow definitions, and the reST and MyST syntax guides.
- *Explanation* (page 128) describes the architecture of the Sphinx Stack, covering its core components (Sphinx, Python, extensions), the Make-based build system, and sitemap generation.

[1] https://diataxis.fr/

3. Project and community

The Sphinx Stack is a Canonical open source project that forms part of the tooling supporting documentation across Ubuntu and other Canonical offerings.

3.1. Get involved

- Sphinx Stack repository[2]
- Issue tracker[3]
- *Contribute to documentation* (page 160)
- *Contribute to development* (page 154)

3.2. Releases

- *Release notes* (page 135)

3.3. Governance and policies

- Code of conduct[4]

3.3.1. Set up a new project

This is a guide for adding Sphinx Stack to a new or existing project. You'll configure it, build the documentation locally, and set up automatic documentation checks.

Before you start

Decide where your documentation will live. Consider a dedicated documentation repository if the documentation covers multiple repositories, has a separate release process, or needs different ownership or access controls. Otherwise, keep the documentation in the same repository as the project. Both approaches use the same Sphinx Stack directory structure and build commands.

Conventionally, the files go in a `docs` directory, which is the default layout used by the Sphinx Stack. This lets contributors review documentation alongside related project changes.

[2] https://github.com/canonical/sphinx-stack
[3] https://github.com/canonical/sphinx-stack/issues
[4] https://ubuntu.com/community/ethos/code-of-conduct

Add the Sphinx Stack

Choose the setup that matches your project.

Create a new repository from the template

Go to the Sphinx Stack repository[5] and select **Use this template** > **Create a new repository**, then fill in the form. Choose an owner; if you're creating documentation for a Canonical project, set the repository owner to **canonical**. Enter a repository name, choose its visibility, add a description if needed, and create the repository.

Add the Sphinx Stack to an existing repository

Copy these paths from the Sphinx Stack repository[6] into the root of your project's repository:

- the `docs` directory
- `.readthedocs.yaml`, which configures builds on Read the Docs
- the workflow files in the `.github/workflows` directory

If your project already has a `.github/workflows` directory, add the relevant documentation workflow files to it rather than replacing the existing directory.

Review the template files

Before committing your setup, review the files you copied or received from the template.

If you created a repository from the template, remove the files that can't be reused, such as `CONTRIBUTING.md` and `.github/CODEOWNERS`. Also remove `.github/workflows/test-sphinx-stack.yml`, which tests the Sphinx Stack itself and can't be reused by your project.

Review the remaining workflows in `.github/workflows/`, remove any that duplicate checks your project already runs, and keep the ones you need. In particular:

- `cla-check.yml` verifies whether contributors have signed the Canonical License Agreement[7]. All Canonical projects require this check, so remove it only if your project already runs it.
- `sphinx-python-dependency-build-checks.yml` verifies Python dependencies for the documentation system. Remove it if your project has its own dependency checks.
- `markdown-style-checks.yml` runs the built-in Markdown linter. Remove it if your project already validates its Markdown files.

[5] https://github.com/canonical/sphinx-stack
[6] https://github.com/canonical/sphinx-stack
[7] https://canonical.com/legal/contributors

Configure your project

Open `docs/conf.py` and replace the project-specific values marked with `TODO`, such as the project name, author, and repository information.

You can leave optional settings at their default values and change them later. See *Configure your project* (page 11) for details about the available settings.

Build and preview the documentation

Building the documentation requires `make`, `python3`, `python3-venv`, `python3-pip`.

From the `docs` directory, run:

```
make run
```

This creates a virtual environment in `docs/.venv`, installs the documentation dependencies, builds the documentation, and starts a local preview server.

Open `http://127.0.0.1:8000/` and confirm that the home page loads and displays the project information you configured.

The server rebuilds the documentation when you change source files or `docs/conf.py`.

The documentation build is self-contained, so you do not need to integrate it with your project's build system during initial setup.

If you later want the project build to also build the documentation, see *Parent projects and the build* (page 134) for how the two builds can coexist, and *Bridge project and documentation builds* (page 46) for the steps to do it.

Add your content

The home page is `docs/index.rst`. The template also provides directories for tutorials, how-to guides, reference material, and explanations under `docs/`.

The navigation menu structure is set by `.. toctree::` directives. These directives define the hierarchy of included content throughout the documentation. The `index.rst` page's `toctree` block contains the top level navigation, which by default is the Diátaxis[8] documentation structure.

To add a new page to the documentation, for example a **Reference** page about your project's settings:

1. Create a new file under the `docs/reference/` directory called `settings.rst` with the following heading:

[8] https://diataxis.fr/

Listing 1: reStructuredText title example

```
Settings
========
```

If you prefer to use Markdown (MyST) syntax instead of reST (reStructuredText), you can create the equivalent `settings.md` file and add the following Markdown-formatted heading at the beginning:

Listing 2: Markdown title example

```
# Settings
```

2. Add the new page to the navigation menu: open the `docs/reference/index.rst` file or another file where you want to nest the new page; at the bottom of the file, add the following `toctree` directive:

```
.. toctree::
   :hidden:
   :maxdepth: 2

   settings
```

The page will appear in the navigation after the documentation rebuilds. For more on the supported syntax, see *reStructuredText syntax* (page 97) and *MyST syntax* (page 113).

By default, the page's title (the first heading in the file) is shown in the global navigation. You can override the name of a menu entry by specifying it explicitly in the `toctree` block (e.g., `Reference </reference/index>`).

Review the automatic checks

The Sphinx Stack includes GitHub workflows for documentation checks, including spelling, links, and inclusive language. Review the workflows' triggers and paths for your repository.

If you move the documentation from `docs/`, update the relevant paths and working directories. See *GitHub workflows* (page 90) for the available workflows, or *run the checks locally* (page 15).

Configure pre-commit hooks (optional)

Optional pre-commit[9] hooks run documentation checks when you create a commit, so you can catch issues before opening a pull request.

The Sphinx Stack includes a ready-to-use `.pre-commit-config.yaml` file under `docs/_dev/`:

```
repos:
  - repo: local
    hooks:
      - id: make-spelling
        name: Run make spelling
```

(continues on next page)

[9] https://pre-commit.com/

(continued from previous page)

```
      entry: make -C docs spelling
      language: system
      pass_filenames: false
      files: ^docs/.*\.(rst|md|txt)$

    - id: make-linkcheck
      name: Run make linkcheck
      entry: make -C docs linkcheck
      language: system
      pass_filenames: false
      files: ^docs/.*\.(rst|md|txt)$

    - id: make-woke
      name: Run make woke
      entry: make -C docs woke
      language: system
      pass_filenames: false
      files: ^docs/.*\.(rst|md|txt)$
```

If your repository does not already use pre-commit, copy this file to the repository root as `.pre-commit-config.yaml`. If it does, add the documentation hooks to its existing configuration.

To apply the configuration, install the Sphinx Stack hooks, for instance:

```
pre-commit install --config docs/_dev/.pre-commit-config.yaml
```

After that, you should see the checks running with every commit:

```
user@host:~$ git commit -m 'add spelling errors'

Run make spelling.....................................................Failed
Run make linkcheck....................................................Passed
Run make woke.........................................................Passed
```

Next steps

Before publishing the documentation:

- Replace the template pages with content for your project.
- Review the remaining settings in *Configure your project* (page 11).
- Run the *documentation checks* (page 15) and confirm that the GitHub workflows you kept behave as expected.
- Follow *Publish on Read the Docs* (page 20).

After the initial setup, you can:

- Use *Bridge project and documentation builds* (page 46) to integrate the documentation and project builds.
- Explore *Optional customisation* (page 44) for features like diagrams, API references, and custom templates.

- Follow *Update your Sphinx Stack* (page 24) to keep your project's Sphinx Stack up to date.

3.3.2. How-to guides

These guides will walk you through the processes involving setting up, maintaining, and contributing to the Sphinx Stack.

Basic setup and maintenance

Set up, configure, update, and customize your project to keep it organized and aligned with your documentation needs.

Configure your project

Sphinx Stack documentation projects are configured in the `docs/conf.py` file. The default configuration will work for most projects, but you still need to set some project-specific values, such as your product's name and documentation URL.

Important:

The Sphinx Stack is updated periodically. After you set up your documentation repository with the Sphinx Stack, you need to track the changes made to the Sphinx Stack and manually maintain your repository.
Use the Sphinx Stack release notes[10] or changelog[11] to track changes to the Sphinx Stack. Subscribe to the repository releases to get notified whenever there is a new release. For the recommended way of manually updating your Sphinx Stack, see *Update your Sphinx Stack* (page 24).

[10] https://documentation.ubuntu.com/sphinx-stack/latest/release-notes
[11] https://github.com/canonical/sphinx-stack/blob/main/CHANGELOG.md

Mandatory configuration values are marked with `TODO` in the `conf.py` file. Set these to align with your project and ensure you comment out any settings that aren't relevant to your environment.

Steps 1 – 5 cover the required configuration. For details, see *Required configuration* (page 91).

1. Set project identity, branding, and repository metadata.
2. Set the documentation site URL.
3. Configure feedback and page navigation behavior.
4. Define LLM metadata.
5. Configure how Sphinx should validate links.
6. Add *optional configuration* (page 93) as needed. The following links can help you with additional configuration:

- Sphinx configuration[12]

[12] https://www.sphinx-doc.org/en/master/usage/configuration.html

- Sphinx extensions[13]
- Furo documentation[14]

Build and preview

The Sphinx Stack provides a `Makefile` that defines `make` commands to build and view the documentation.

This guide describes how to set up your environment and use these commands to build and preview the documentation locally.

For more advanced information, including how to embed your docs build with your project build, see:

- *Build* (page 132)
- *Bridge project and documentation builds* (page 46)

Install prerequisite software

The documentation framework that the Sphinx Stack uses bundles most prerequisites in a Python virtual environment, so you don't need to worry about installing them. There are only a few packages that you need to install on your host system.

Before you start, make sure that you have `make`, `python3`, `python3-venv`, and `python3-pip` on your system:

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

Manage Python environment

The Python prerequisites from the `docs/requirements.txt` file are automatically installed when you build the documentation.

If you want to install them manually, you can run the following command in your documentation directory:

```
make install
```

This command creates a virtual environment (`.venv/`) and installs dependency software within it.

If you want to remove the installed Python packages (for example, to enforce a re-installation), run the following command in your documentation directory:

```
make clean
```

[13] https://www.sphinx-doc.org/en/master/usage/extensions/index.html
[14] https://pradyunsg.me/furo/quickstart/

Note:

- By default, the Sphinx Stack uses the latest compatible version of all tools and does not pin its requirements. This might change temporarily if there is an incompatibility with a new tool version. There is therefore no need to use a tool like Renovate to automatically update the requirements.
- If you encounter the error `locale.Error: unsupported locale setting` when activating the Python virtual environment, include the environment variable in the command and try again: `LC_ALL=en_US.UTF-8 make run`

Important:

Run these commands in your documentation directory.

Build the documentation

To build the documentation, run the following command:

```
make html
```

This command installs the required tools and renders the output to the `docs/_build/` directory.

Important:

When you run `make html` again, it updates the documentation for changed files only. This speeds up the build, but it can cause you to miss warnings or errors that were displayed before. To force a clean build, see `build-clean`.

Make sure that the documentation builds without any warnings (warnings are treated as errors).

Run a clean build

To delete all existing output files and build all files again, run the following command:

```
make clean-doc && make html
```

To delete both the existing output files and the Python environment and build the full documentation again, run the following command:

```
make clean && make html
```

View the documentation

To view the documentation output, run the following command:

```
make serve
```

This command builds the documentation and serves it at http://127.0.0.1:8000/.

Run a live preview

Instead of building the documentation for each change and then serving it, you can run a live preview of the documentation:

```
make run
```

This command builds the documentation and serves it at http://127.0.0.1:8000/. When you change a documentation file and save it, the documentation will be automatically rebuilt and refreshed in the browser.

If you need project-specific options for `sphinx-autobuild` such as `--ignore` or `--watch`, pass them through `SPHINX_AUTOBUILD_OPTS`:

```
make run SPHINX_AUTOBUILD_OPTS="--ignore '**/*.gen.rst' --watch ../data/"
```

If you call `make run` from a *parent project's build* (page 46), pass the variable explicitly to the sub-Make call to ensure it reaches the docs Makefile:

```
$(MAKE) -C docs run SPHINX_AUTOBUILD_OPTS="$(SPHINX_AUTOBUILD_OPTS)"
```

This approach allows command-line overrides to work intuitively from the parent context.

Important:

The `run` target is very convenient while working on documentation updates.
However, it is quite error-prone because it displays warnings or errors only when they occur. If you save other files later, you might miss these messages.
Therefore, you should always run `make clean` before finalizing your changes.

Build a PDF

Build a PDF locally with the following command:

```
make pdf
```

PDF generation requires specific software packages. If these files are not found, a prompt will be presented and the generation will stop.

Required software packages include:

- dvipng
- fonts-freefont-otf

- latexmk
- plantuml
- tex-gyre
- texlive-font-utils
- texlive-fonts-recommended
- texlive-lang-cjk
- texlive-latex-extra
- texlive-latex-recommended
- texlive-xetex
- xindy

On Linux, required packages can be installed with:

```
make pdf-prep-force
```

Note:

When generating a PDF, the index page is considered a 'foreword' and will not be labelled with a chapter.

Important:

When generating a PDF, it is important to not use additional headings before a `toctree`. Documents referenced by the `toctree` will be nested under any provided headings.
A `rubric` directive can be combined with the `h2` class to provide a heading-styled rubric in the HTML output. See the default `index.rst` for an example. Rubric-based headings aren't included as entries in the table of contents or the navigation sidebar.

Run documentation checks

The Sphinx Stack comes with several tests and checks that you can (and should!) run on your documentation before committing and pushing changes:

- *Accessibility check* (page 16)
- *Inclusive language check* (page 16)
- *Link check* (page 17)
- *Markdown lint check* (page 17)
- *Spelling check* (page 17)
- *Style guide linting* (page 18)
- *Vale exemptions* (page 18)

Accessibility check

The Sphinx Stack checks the accessibility of the documentation with Pa11y[15]. It's configured to check for Level AA conformity to the Web Content Accessibility Guidelines (WCAG) 2.2[16].

This check is only available locally; it is not part of the *GitHub workflows* (page 90) in the Sphinx Stack.

To run the check, you must have `npm` and `Pa11y` installed. To install `npm`, refer to the installation guide[17] for details. Then:

```
npm install pa11y
```

To check the accessibility of the documentation, run the following command in your `/docs` directory:

```
make pa11y
```

Configure

The `pa11y.json` file in the `_dev` directory provides basic defaults expected to be used for all teams; refrain from removing any unless absolutely necessary. Additional settings and options are detailed in the `Pa11y` README[18] on GitHub.

Inclusive language check

The Sphinx Stack checks for inclusive language with a custom set of Vale[19] rules. To check for inclusive language violations, run:

```
make woke
```

Configure

If you need to exempt single words or sections, follow the process in *Vale exemptions* (page 18).

[15] https://pa11y.org/
[16] https://www.w3.org/TR/WCAG22/
[17] https://docs.npmjs.com/downloading-and-installing-node-js-and-npm
[18] https://github.com/pa11y/pa11y#command-line-configuration
[19] https://vale.sh

Link check

The Sphinx Stack checks any links in your documentation with the Sphinx `linkcheck` builder. To validate the links, run:

```
make linkcheck
```

Configure

If you have links in the documentation that you don't want to check, add them to the `linkcheck_ignore` list in the `conf.py` file. See *Sphinx link checker* (page 93) for more information on how to configure the link checker.

Markdown lint check

The Sphinx Stack checks standards and consistency in Markdown files with the Markdown lint check. To check your Markdown files, run:

```
make lint-md
```

Configure

You can update the linting rules in the `_dev/.pymarkdown.json` file. Refer to the pymarkdown rules documentation[20] for all the available rules.

Spelling check

The Sphinx Stack uses Vale[21] to check the spelling in your documentation. It ignores inline code, code blocks, and URLs, but it does check the link text. To ensure there are no spelling errors in your documentation, run:

```
make spelling
```

Configure

If you need to exempt single words or sections, follow the process in *Vale exemptions* (page 18).

[20] https://pymarkdown.readthedocs.io/en/latest/rules/
[21] https://vale.sh

Style guide linting

The Sphinx Stack runs the Vale[22] documentation linter configured with the rules of style guide[23] To check your documentation with Vale, run:

```
make vale
```

Vale can run against individual files, directories, or globs. To check a specific target, run:

```
make vale TARGET=example.file
make vale TARGET=example-directory
```

If you run the linter against a directory, its subdirectories were also linted.

You can use wildcards to lint globs of files. For example, to run against all Markdown files within a directory, you'd run:

```
make vale TARGET=*.md
```

Configure

If you need to exempt single words or sections, follow the process in *Vale exemptions* (page 18).

Vale exemptions

The *Inclusive language check* (page 16), *Spelling check* (page 17), and *Style guide linting* (page 18) all use Vale[24]. These checks share a common syntax for exemptions and pull from the same `.custom_wordlist.txt` file. The style guide repository includes a common list of words[25] that will be excluded from the checks.

A word

To ignore a single instance of a word, wrap it in the `:vale-ignore:` role. Ensure your `conf.py` file contains the appropriate class association in the `rst_prolog`:

```
rst_prolog = """
.. role:: vale-ignore
    :class: vale-ignore
"""
```

To ignore a word across your documentation, add it to your project's `.custom_wordlist.txt` file.

[22] https://vale.sh
[23] https://github.com/canonical/documentation-style-guide
[24] https://vale.sh
[25] https://github.com/canonical/documentation-style-guide/blob/main/styles/config/vocabularies/Canonical/accept.txt

Case sensitivity

Entries in `.custom-wordlist` are case-sensitive only when the word is capitalized. For instance:

- Adding `kustom` will cause all instances of `Kustom` and `kustom` to be ignored.
- Adding `Kustom` will cause only instances of `Kustom` to be ignored.

A section of text

To disable Vale for a section of text, use the following markup:

Markdown

```
<!-- vale off -->

This text will be ignored.

<!-- vale on -->
```

reST

```
.. vale off

This text will be ignored.

.. vale on
```

Directives

To disable Vale linting for Directives as a code block or admonitions, apply the `vale-ignore` class to the section:

Markdown

````
````{class} vale-ignore
```{code-block}

This content will be ignored by Vale.
```
````
````

reST

```
.. class:: vale-ignore
    .. code-block::

        This content will be ignored by Vale.
```

The class process isn't be necessary for Markdown, as Vale has an expanded scope for ignoring Markdown content by default. Also, the .. *class::* directive for reST does not need to encapsulate content, it applies to the next logical block (which can be another directive or even a paragraph of content). The class option should only be used when other options aren't suitable.

Hyphens and spaces

The checks may flag terms that contain hyphens or spaces.
For example, "Juju 3" wasn't ignored by default, and needed to be added to be excepted with an upstream rule[26].

[26] https://github.com/canonical/documentation-style-guide/blob/a6f530b07d774bee67dd79d146ae5bbedc9ddef1/styles/Canonical/013-Spell-out-numbers-below-10.yml#L15

Publish on Read the Docs

Canonical documentation, both internal and external, is hosted on Read the Docs. This guide walks you through the process of setting up your Sphinx Stack project on Read the Docs.

Prepare your project and account

- Ensure your documentation *builds without errors or warnings* (page 13).
- Ensure you have permission to manage webhooks for the documentation repository, or get in touch with someone who does. A webhook is needed if you want Read the Docs to automatically build your documentation whenever a change occurs.
- In the root of your documentation repository, create the .readthedocs.yaml[27] file.
- Create a Read the Docs account[28]. If your documentation is hosted on a supported Git provider (such as GitHub, GitLab, or Bitbucket), connect the account to your Git provider[29].

[27] https://docs.readthedocs.com/platform/stable/config-file/index.html
[28] https://docs.readthedocs.com/platform/stable/tutorial/index.html#creating-a-read-the-docs-account
[29] https://docs.readthedocs.com/platform/stable/guides/connecting-git-account.html

Note:

For Canonical staff, a Read the Docs account is created during onboarding and connected to our Google SSO:

1. Log in to the Read the Docs dashboard[30] using the *Sign in with Google* option.
2. Ask on the ~Documentation[31] Mattermost channel to be added to your specific team.
3. Accept the invitation that is sent to your email. Make sure you are logged in to Read the Docs with your Google account when you do this.

[30] https://readthedocs.com/dashboard/
[31] https://chat.canonical.com/canonical/channels/documentation

Add your project to Read the Docs

Log in to the Read the Docs dashboard[32] to add a project. If you are the repository administrator, add the project automatically[33]. If you are not a repository administrator or your documentation is not hosted on a supported Git provider, add the project manually[34]. A build will be triggered after the project is added.

When adding a project, most fields are self-explanatory, and you can leave the default values. Check and update the following fields:

- **Organization and team**: This determines who gets admin access to your Read the Docs project.
- **Name**: This is the slug used in the documentation URL. You cannot change the name after you create the project, so choose carefully!
- **Default branch**: This determines which branch "latest" points to.

Note:

If your repository does not have a main branch and you don't specify the correct default branch, the build will default to main and will not be able to check out the repository and find the correct branch. If this happens, the solution is to add a main branch to the project until you're able to build it once, then specify the correct branch.

Check the first build[35]. After it completes successfully, you can view the live documentation from the project home page.

If there are any build errors, fix them:

- If your project is hosted in a private repository, your first build will fail because your repository is not configured to allow Read the Docs to clone the repository. To fix the access issue, follow the steps in Configuring your repository[36].

[32] https://readripman.readthedocs.io/en/latest/glossary.html#term-dashboard
[33] https://docs.readthedocs.com/platform/stable/intro/add-project.html#automatically-add-your-project
[34] https://docs.readthedocs.com/platform/stable/intro/add-project.html#manually-add-your-project
[35] https://docs.readthedocs.com/platform/stable/tutorial/index.html#checking-the-first-build
[36] https://docs.readthedocs.com/platform/stable/guides/creating-project-private-repository.html#configuring-your-repository

- If your project is hosted in a public repository, your documentation should build successfully. If you get any errors, check the build log for indications of what the problem is.

Canonical-specific configuration

After the initial build, some Canonical-specific configurations are required, as listed below.

Change the host URL to documentation.ubuntu.com

By default, your documentation URL is `https://canonical-<name>.readthedocs-hosted.com`, where `<name>` is the slug you specified when adding the project. Change the host URL to `https://documentation.ubuntu.com`:

1. Log in to the Read the Docs dashboard[37].
2. Locate the "Ubuntu documentation library" project and go to *Settings* > *Subprojects* to add a subproject.
3. Select your project to connect as a subproject from the Subproject dropdown list.
4. (Optional) To use a different name in the URL structure, specify the name in the *Alias* field. Leave it blank to use the name you specified when adding your project.
5. Click *Add subproject*.

Your documentation URL has changed to `https://documentation.ubuntu.com/<name>`.

Limit documentation access

To limit the documentation access to Canonical users, the project needs to be added to the All Canonical Users[38] group. All Canonical users who log in to Read the Docs with Google SSO will be automatically added to this user group and granted read-only permissions to the associated projects.

This operation requires the Owner's permission to Read the Docs. Contact one of the owners of the Canonical organization[39] or reach out to the ~Documentation[40] channel.

[37] https://readthedocs.com/dashboard/
[38] https://app.readthedocs.com/organizations/canonical/teams/all-canonical-users/
[39] https://app.readthedocs.com/organizations/canonical/members/?teams__slug=all-canonical-users&access=owner
[40] https://chat.canonical.com/canonical/channels/documentation

Change URL versioning scheme

The default versioning scheme is Multiple versions with translations[41]. Unless your documentation has multiple languages, do not use the scheme with translation. Change the URL versioning scheme[42] under your project Settings to either a single version without translations or multiple versions without translations.

Additional configuration

Configurations in this section are all optional. Follow the instructions here to configure your Read the Docs project according to your preferences.

Enable automatic builds

The Read the Docs integration webhook can automatically build your documentation when your GitHub repository changes. If you added the GitHub repository to Read the Docs automatically, the integration webhook was already created. Otherwise, the webhook must be set up by someone with the appropriate permission.

To check if the webhook was created, open the project Settings and click *Integrations*. If the webhook is not listed, someone with the appropriate permissions must manually set it up[43].

Automatic builds are not supported on Launchpad because its webhooks aren't compatible with Read the Docs. Builds must be triggered manually on Read The Docs.

Enable pull request builds

To build and preview your documentation whenever a pull request is opened or updated, open the project Settings and click *Pull request builds* to enable it for your project.

Make your documentation public (RTD for Business only)

By default, Read the Docs publishes your documentation for logged-in users only.

To make your documentation publicly accessible, you must configure the privacy level for each version of the documentation separately. Navigate to the Versions tab to configure the Privacy Level[44] for each version.

If your documentation publishes tagged versions that should be public by default, add an Automation Rule[45] under the project Settings with the following configuration:

[41] https://docs.readthedocs.com/platform/stable/versioning-schemes.html#multiple-versions-with-translations
[42] https://docs.readthedocs.com/platform/stable/versioning-schemes.html#how-to-change-the-url-versioning-scheme-of-your
[43] https://docs.readthedocs.com/platform/stable/guides/setup/git-repo-manual.html#how-to-manually-configure-a-git-repository-integration
[44] https://docs.readthedocs.com/platform/stable/guides/pull-requests.html#privacy-levels
[45] https://docs.readthedocs.com/platform/stable/guides/automation-rules.html#adding-a-new-automation-rule

Configuration field	Value
Version type	Tag
Version predefined match pattern	Any version
Action	Make version public

Add support for Git Large File Storage

By default, Read the Docs can't access large files stored with Git LFS. If your docs project contains large files, add support for Git LFS[46].

Update your Sphinx Stack

The Sphinx Stack is updated frequently. After you copy its code and it becomes the documentation system in your project, you must manually maintain it.

There are special pathways to bring in the latest features and improvements from the Sphinx Stack repository.

Determine your current version

There are two versions of the Sphinx Stack that entail different update processes:

- the **new**, or **extension-based** Sphinx Stack
- the **legacy**, or **pre-extension** Sphinx Stack

New Sphinx Stack

If your `conf.py` file includes `canonical_sphinx` under the `extensions` list, you are using the new Sphinx Stack.

To update a new Sphinx Stack project to the latest version, see:

- *Update the new Sphinx Stack* (page 25)

Note:

New Sphinx Stack releases use a semantic versioning scheme for minor and patch versions, which can be found in your project's `_dev/version` file.

[46] https://docs.readthedocs.com/platform/stable/build-customization.html#support-git-lfs-large-file-storage

Legacy Sphinx Stack

If your `conf.py` file does _not_ include `canonical_sphinx` you are using the legacy Sphinx Stack.

To update a legacy Sphinx Stack project to the latest version of the new Sphinx Stack:

- *Update the legacy Sphinx Stack* (page 28)

Update the new Sphinx Stack

The Sphinx Stack is regularly updated to add features and address bugs. You can transfer these improvements to your project by following these steps:

- Clone the latest version of the Sphinx Stack
- Compare key files and directories in the Sphinx Stack to your project
- Transfer or delete relevant changes
- Confirm that your project builds correctly with the new changes

This guide assumes your project has minimal customizations, and the repository structure closely mirrors that of the Sphinx Stack. Depending on your customizations, you may need to take extra steps when updating.

Note:

If `canonical_sphinx` is not included under `extensions` in your `conf.py` file, your project is not on an extension-based Sphinx Stack. Follow the guide on *updating a legacy Sphinx Stack project* (page 28).

Clone the Sphinx Stack repository

If you don't have a clean, local copy of the Sphinx Stack, clone it:

```
git clone https://github.com/canonical/sphinx-stack.git
```

Confirm that the Sphinx Stack documentation and your project build with no errors.

Important:

Verify that your project still builds correctly after each key step. This makes it easier to identify causes of build errors.

Update the configuration and build files

New Sphinx Stack versions often change the default configuration files. You'll need to merge your project files with the config files from the new Sphinx Stack. The recommended approach is to copy the customizations in your project to the Sphinx Stack config files and then replace your project's config files with those from the Sphinx Stack.

The changes to be made vary between projects and updates. Therefore, this guide cannot be overly prescriptive.

`conf.py`

Rename your `conf.py` file to avoid overwriting it, and copy the Sphinx Stack version to the same location. Use a graphical diff tool such as Kompare[47] or meld[48] to compare the old and new file and make the following changes:

- Copy your standard project details to the new `conf.py` file. These include:
  - Project and author names
  - Ignored links
  - Social links, etc.
- Verify that the `/_static` and `/_templates` directories are located at the locations specified by `html_static_path` and `templates_path`, respectively, in the new `conf.py` file. These should not be inside the `/_dev` directory.

For other customizations, consider need and compatibility before copying them to the new file. If it's not obvious whether you should copy over a customization or include a new change, reach out to Canonical's documentation team[49].

`Makefile` and `.readthedocs.yaml`

Depending on the version of your project's Sphinx Stack, the new `Makefile` and `.readthedocs.yaml` files may have few or no changes. Apply the same approach you used for `conf.py` to merge your customizations into the new files.

If there are no project-specific customizations in your files but there are changes in the new ones, you can just overwrite your existing files with the new ones.

[47] https://apps.kde.org/kompare/
[48] https://meldmerge.org/
[49] https://matrix.to/#/#documentation:ubuntu.com

Update the _dev directory

In addition to the docs above, the /_dev directory is also likely to have some changes in each update. These files are not intended to be modified by users.

Unless you intentionally customized files in this directory, you can simply delete your project's /_dev directory and replace it with the same directory from the Sphinx Stack. If there are modifications in your project's /_dev directory, it is recommended that they transfer them out.

Review the remaining files

Some files in the Sphinx Stack may be updated less frequently, but it's a good idea to review them during each update and determine if there are relevant changes:

- Review `requirements.txt`: If there are any updates, and your project's file has no repository-specific requirements, you can overwrite the existing file with the new one. If you added requirements based on your customizations, be sure to include them.
- Review the workflows in the `/.github` directory: If there are changes in the following workflows, replace the existing files with the new ones. The Sphinx Stack will have other workflows as well, but you'll need to decide whether your project needs them or not:
  - Automatic doc checks
  - CLA (contributor license agreement) check
  - Check for removed URLs
  - Markdown style check (only required for docs using markdown)
- Review and transfer any necessary changes in the new `.gitignore` file to your project.

Build and test

Try building the docs locally and check the terminal output for errors:

```
make run
```

To ensure the updated docs will pass CI checks when you make a pull request, run the following commands and fix any errors reported:

- `make spelling`
- `make linkcheck`
- `make woke`
- `make lint-md` (if you included the `markdown-style-checks` workflow)

Troubleshooting errors

There is always the possibility of encountering build errors. Common causes include:

- Incorrect file locations or file paths
- Incompatible requirements in the new requirements file
- Missing customizations
- Cached build files

When troubleshooting use the `make clean` command to ensure cached versions of build files are not reused.

Clean up

There may be files that need to be deleted after the update such as Sphinx Stack files or files that have been replaced with newer versions:

- If you haven't done so already, delete the copies of `conf.py`, `Makefile`, and `/.readthedocs.yaml` that were renamed and replaced.
- If you did not strictly follow this guide for this or previous updates, it's possible that you have some Sphinx Stack files in your project. These files can be safely deleted:
  - `.github/pull_request_template.md`
  - `.github/workflows/sphinx-python-dependency-build-checks.yml`
  - `.github/CODEOWNERS`
  - `.github/workflows/test-sphinx-stack.yml`

Update the legacy Sphinx Stack

This guide outlines the steps required to migrate a documentation project from the legacy Sphinx Stack (*pre-extension* version) to the latest version that adopts the `canonical-sphinx` Sphinx extension.

The extension-based Sphinx Stack provides a set of features and configurations that are common across Canonical documentation projects. Key components, such as configuration and styling, are loaded as an add-on to your project. It can significantly reduce maintenance concerns when managing your documentation.

Note:

If `canonical_sphinx` is included under `extensions` in your `conf.py` file, you are already using an extension-based Sphinx Stack. Follow the guide on *updating the new Sphinx Stack* (page 25).

Update to the last pre-extension version

Update your documentation project to use the last pre-extension version of the Sphinx Stack. Doing so will minimize the changes required during the migration by including the the latest features and configurations available in the pre-extension version.

In your project repository, check out the pre-extension release tag[50]:

```
git checkout tags/pre-extension
```

Repository in detached HEAD state

Checking out a tag leaves your repository in a detached HEAD state[51]. Before proceeding, create a new branch:

```
git checkout -b migrate-to-extension
```

[51] https://git-scm.com/docs/git-checkout#_detached_head

Set up a new project

1. Back up all existing files in your original documentation project. For example, you can rename the original `docs/` directory to `docs_backup/`.

   Warning:

   If you proceed in the same directory, the following steps will overwrite some of the configuration files in the original project.

2. Follow the steps in the *Add the Sphinx Stack* (page 7) guide to initialise an empty project with the extension-based Sphinx Stack, at the original file path.
3. Ensure the following files are at the root of your repository:
   - `.github/workflows/*`
4. Ensure the following files are moved to their original paths in the project. These files are defaulted to the repository root, but may have be changed upon project needs:
   - `.gitignore`
   - `.readthedocs.yml`
5. Validate the project setup locally by running `make run` in the new project directory.

[50] https://github.com/canonical/sphinx-stack/releases/tag/pre-extension

Migrate source files

The Sphinx Stack has undergone breaking changes with the introduction of the `canonical-sphinx` extension. This section guides you through:

- Configuration file changes
- Extension dependencies
- Documentation source migration

For a complete list of the structural changes, refer to the *directory-structure-change* (page 32) section.

Sphinx configuration

A significant change in the new Sphinx Stack is the organisation of the configuration files, summarised in the following table:

Configuration file	Pre-extension	Extension-based
`conf.py`	Common configurations shared by all Sphinx Stack projects	Project-specific configurations
`custom_conf.py`	Project-specific configuration	Merged into `conf.py` and removed

In the new Sphinx Stack, many common configurations are provided by the extension and are loaded automatically when building the documentation. `docs/conf.py` is the only configuration file, and it contains all project-specific configuration. Sensible defaults are set for general configuration by inclusion of the `canonical-sphinx` extension.

Ensure that all the previous changes in the original `custom_conf.py` file are copied to the new `conf.py` file.

Dependencies

The Sphinx Stack supports a reasonable set of default extensions. If your project requires additional extensions, add them to the project's `docs/requirements.txt` file.

It's your responsibility to ensure that extensions you add to this list are compatible with the rest of the Sphinx Stack.

Documentation source files

1. Copy all documentation source files from your original project to the new project, keeping their original structure. These file may include but are not limited to:
    - `.md`
    - `.rst`
    - `.txt`
    - `.json`
    - images
    - scripts
2. Validate the migration by running `make run`.

Apply customisation

If your projects have custom configurations or styles, ensure that you identify and apply these changes to the new documentation project.

For general information on customising the extension configuration, see *Configure your project* (page 11).

Static resources

The extension provides a set of static resources, such as images, fonts, CSS files, and HTML templates, that are used to style the documentation for Canonical-branded design. These resources are bundled with the extension and are no longer provided as source files in the new Sphinx Stack.

If you have customised any of these resources in the original project, you need to manually migrate these changes to the new project.

For example, if you added customised styling in the original `.sphinx/_static/custom.css` file, follow the steps:

1. Compare the changes between your customised file and the default CSS file provided by the extension[52]. This comparison helps you identify the changes that need to be migrated to the new project.
2. Create a new CSS file under `docs/_dev/_static`. You can choose any other file location in the project directory, but it's recommended to keep the file structure similar to the original project.
3. Copy the additions and changes to the new empty file.
4. In the `conf.py`, add the new files into the pre-defined `html_css_files` list variable to overwrite the default settings.
5. Build the documentation to verify that the customised styling is applied correctly.

[52] https://github.com/canonical/canonical-sphinx/blob/main/canonical_sphinx/theme/static/custom.css

Directory structure changes

After you migrate to the extension, some directories and files are either deleted from the project or moved to a new location.

Assuming that all previous documentation files were in the `docs/` sub-directory, the following list illustrates the changes in the directory structure after the migration.

```
.
├── .github
│   └── workflows
│       ├── automatic-doc-checks.yml
│       └── markdown-style-checks.yml
├── .sphinx                     # renamed to `docs/_dev`
│   ├── fonts                   # removed, files are part of the extension
│   │   ├── Ubuntu-B.ttf
│   │   ├── ubuntu-font-licence-1.0.txt
│   │   ├── UbuntuMono-B.ttf
│   │   ├── UbuntuMono-RI.ttf
│   │   ├── UbuntuMono-R.ttf
│   │   ├── Ubuntu-RI.ttf
│   │   └── Ubuntu-R.ttf
│   ├── images                  # removed, files are part of the extension
│   │   ├── Canonical-logo-4x.png
│   │   ├── front-page-light.pdf
│   │   ├── front-page.png
│   │   └── normal-page-footer.pdf
│   ├── _static                 # removed, files are part of the extension
│   │   ├── 404.svg
│   │   ├── custom.css
│   │   ├── favicon.png
│   │   ├── footer.css
│   │   ├── footer.js
│   │   ├── furo_colors.css
│   │   ├── github_issue_links.css
│   │   ├── github_issue_links.js
│   │   ├── header.css
│   │   ├── header-nav.js
│   │   └── tag.png
├── _templates                  # moved to `docs/_templates`
│   ├── sidebar                 # removed
│   │   └── search.html
│   ├── 404.html                # removed
│   ├── base.html               # removed
│   ├── footer.html
│   ├── header.html
│   └── page.html               # removed
│   ├── build_requirements.py   # removed
│   ├── get_vale_conf.py
│   ├── latex_elements_template.txt     # removed, now part of the extension
│   ├── pa11y-ci.json           # renamed to `pa11y.json`
│   └── spellingcheck.yaml
├── metrics                     # removed
│   └── scripts
│       ├── build_metrics.sh
│       └── source_metrics.sh
├── reuse                       # removed
```

(continues on next page)

(continued from previous page)

```
│   └── links.txt
├── .custom_wordlist.txt        # moved to `docs/.custom_wordlist.txt`
├── .gitignore
├── .readthedocs.yaml
├── .wordlist.txt               # removed
├── .wokeignore                 # removed, check replaced by Vale
├── conf.py                     # removed, now part of the extension
├── custom_conf.py              # renamed and moved to `docs/conf.py`
├── doc-cheat-sheet-myst.md     # removed
├── doc-cheat-sheet.rst         # removed
├── index.rst                   # moved to `docs/index.rst`
├── init.sh                     # removed
├── make.bat                    # removed
├── Makefile                    # moved to `docs/Makefile`
├── Makefile.sp                 # removed
└── readme.rst                  # renamed to `README.md`
```

Configure the Ulwazi theme

Sphinx Stack 3.0 introduced major improvements to the look and feel of Canonical documentation with its new theme, Ulwazi. Because of this dramatic change, the update requires more configuration changes than usual.

This guide covers the initial setup and configuration of Ulwazi. Applying the diff between your project and Sphinx Stack 3.0 introduces the same changes described in these sections, but for heavily modified projects, it helps to step through the changes one at a time.

Install and set up Ulwazi

The removed lines shown in the diffs are from Sphinx Stack 2.0, but the same basic changes can be applied for projects upgrading from Sphinx Stack 1.3 and higher. If your project is based on Sphinx Stack 1.2 or lower, *update the entire project* (page 25).

In the `docs/requirements.txt` file, replace the canonical-sphinx extension with Ulwazi.

Listing 3: requirements.txt

```
- canonical-sphinx~=0.7
+ ulwazi~=1.0
```

Then, replace the `canonical_sphinx` module in your configuration file with `ulwazi`. While you're here, you should also remove the `notfound.extension` module. Because the sphinx-notfound-page extension affects 404 page style and layout, Ulwazi sets it up automatically.

Listing 4: conf.py

```
  extensions = [
-     "canonical_sphinx",
-     "notfound.extension",
+     "ulwazi",
      # ...
  ]
```

If you write documentation with MyST markdown, add the MyST parser as well. This was previously set up behind the scenes with the canonical-sphinx extension.

Listing 5: conf.py

```
  extensions = [
-     "canonical_sphinx",
-     "notfound.extension",
+     "ulwazi",
+     "myst_parser",
      # ...
  ]
```

If your project is based on Sphinx Stack 1.3 or higher, then the MyST parser is already listed as a dependency.

Add the following line to your configuration:

Listing 6: conf.py

```
html_theme = "ulwazi"
```

At this point, your documentation should build successfully, albeit with some rendering quirks that we'll iron out shortly. In the `docs` directory, clean your environment and make sure the documentation builds with:

```
make clean html
```

Add fundamental configuration

The documentation builds, but a lot of information is still missing. Ulwazi offers a lot of freedom when it comes to elements in the banner and footer, but to align your docs with Canonical standards, you must update your configuration according to this section.

To ensure that a persistent element tells users they're looking at product docs, append "documentation" to the `project` string. This gets rendered on the banner and will serve as the home page link once we're done.

Listing 7: conf.py

```
- project = "Ubuntu Server"
+ project = "Ubuntu Server documentation"
```

Then, update the `html_title` variable that gets appended to each page's title.

Listing 8: conf.py

```
- html_title = project + " documentation"
+ html_title = project
```

The rest of the Ulwazi configuration is done in the configuration file's `html_context` dictionary. Start by replacing your current `html_context` dictionary as follows:

Listing 9: conf.py

```
- html_context = {
-     "product_page": "",
-     "discourse": "",
-     "mattermost": "",
-     "matrix": "",
-     "github_url": "",
-     "repo_default_branch": "main",
-     "repo_folder": "/docs/",
-     "display_contributors": False,
-     "github_issues": "enabled",
-     "author": author,
-     "license": {
-         "name": "",
-         "url": "",
-     },
- }
+ html_context = {
+     "discourse": "",
+     "mattermost": "",
+     "matrix": "",
+     "github_url": "",
+     "feedback": True,
+ }
```

Update each value with your project's URL for the platform. If your project does *not* have a presence on one of the platforms, leave the entry intact and the string empty.

Next, add the following entries to configure the banner:

Listing 10: conf.py

```
  html_context = {
      "discourse": "",
      "mattermost": "",
      "matrix": "",
      "github_url": "",
      "feedback": True,
+     "logo_title": "Canonical",
+     "logo_link_URL": "https://canonical.com",
+     "logo_img_URL": "https://assets.ubuntu.com/v1/82818827-CoF_white.svg",
  }
```

The `logo_title` and `logo_link_URL` should match your project's parent product page. For most projects, this will be the Canonical product page, so the previous values won't need to be changed. However, for a project like the Charmcraft docs, which are hosted under the Juju product page, these would be updated as follows:

Listing 11: conf.py

```
  html_context = {
      "discourse": "",
      "mattermost": "",
      "matrix": "",
      "github_url": "",
```

(continues on next page)

(continued from previous page)

```
      "feedback": True,
-     "logo_title": "Canonical",
-     "logo_link_URL": "https://canonical.com",
+     "logo_title": "Juju",
+     "logo_link_URL": "https://canonical.com/juju",
      "logo_img_URL": "https://assets.ubuntu.com/v1/82818827-CoF_white.svg",
  }
```

Lastly, declare the license information to be displayed in the footer.

Listing 12: conf.py

```
  html_context = {
      "discourse": "",
      "mattermost": "",
      "matrix": "",
      "github_url": "",
      "feedback": True,
      "logo_title": "Canonical",
      "logo_link_URL": "https://canonical.com",
      "logo_img_URL": "https://assets.ubuntu.com/v1/82818827-CoF_white.svg",
+     "author": author,
+     "license": {
+         "name": "",
+         "url": "",
+     },
  }
```

Set `license.name` to the SPDX identifier[53] for your project's license and `license.url` to the license file in your repository.

And that's it! At this point, you can rebuild your documentation and see how everything looks.

While this is the minimal configuration expected for all Canonical documentation projects, Ulwazi can be further catered to your project with optional configuration, described in the Ulwazi Configuration reference[54].

Enable PDF builds

Unlike the canonical-sphinx theme, Ulwazi doesn't handle PDF builds. PDF builds should now be handled with the `sphinx-modern-pdf-style` extension.

In the `docs/requirements.txt` file, add the extension.

Listing 13: requirements.txt

```
+ sphinx-modern-pdf-style~=0.1
```

Then, set up and enable the extension by making the following changes to your project's configuration file:

[53] https://spdx.org/licenses/
[54] https://documentation.ubuntu.com/ulwazi/content/configuration/

Listing 14: conf.py

```
  extensions = [
+     "sphinx_modern_pdf_style",
      # ...
  ]

+ set_modern_pdf_config = True
```

Troubleshooting

Help diagnose and resolve common issues you might encounter when developing documentation with the Sphinx Stack and deploying on Read the Docs.

Troubleshoot documentation issues

This section helps you diagnose and resolve common issues you might encounter when developing documentation with the Sphinx Stack and deploying it on Read the Docs. It covers the following areas:

- *Build Errors* (page 37): How to resolve common build errors
- *Local deployment issues* (page 40): How to troubleshoot issues when deploying your documentation locally for development and testing
- *Runtime Errors* (page 40): How to troubleshoot runtime issues related to Sphinx rendering peculiarities or link configurations
- *Read the Docs Failures* (page 41): How to troubleshoot issues when building and deploying your documentation on Read the Docs

Build errors

In this guide, you will find information on how to troubleshoot local or GitHub PR build errors that may occur when building a documentation set.

'Module not found' error

The build fails with a `ModuleNotFoundError: No module named '<module>'` error message when you run it locally (or in a GitHub PR check).

Probable cause

A `ModuleNotFoundError` means the Python interpreter running the build cannot locate the named module at all. The most common causes for a local build are:

- **The package is missing from** `requirements.txt`. The module is not installed because it is not declared as a dependency. If it works for you but fails for a colleague or in CI, it is likely installed in your environment but absent from `requirements.txt`.
- **A local module or extension is not on** `sys.path`. Custom extensions or `conf.py` helpers that live inside the repository are not importable unless their directory is added to the path in `conf.py`.

Resolution

Read the full traceback and note the exact module name reported after `No module named` error message. If it is a third-party package (for example `myst-parser`), check whether it is listed in `requirements.txt`. If it is missing, add it and reinstall. If it is present, confirm it is actually installed in the active environment with `pip show <package>`. If a newer version is available, consider updating it and adjusting the version constraints in `requirements.txt` accordingly.

Rebuild to confirm the fix

```
make clean; make run
```

Tip:

If the build works for you but fails for a colleague or in a GitHub PR check, the module is almost always installed in your local environment but missing from `requirements.txt`. Add it there so every environment installs it.

For local extensions, you need to append the source to your system path by adding the following line to your `conf.py` file:

```
sys.path.insert(0, os.path.abspath('<path>'))
```

'Pip resolution too deep' error

This error occurred recently while building several Sphinx Stack based documentation sets independently on different machines on a specific day when a new version of `myst-parser` was released. The issue could be resolved by pinning the `myst-parser` version in the `requirements.txt` file.

Probable cause

This error may occur when the `requirements.txt` file has conflicting dependencies or a dependency tree is too complex for `pip` to resolve efficiently.

Documentation based on the Sphinx Stack often hit this error due to an unpinned or incompatible version of a package.

Resolution

Review the dependencies listed in the `requirements.txt` file and resolve any conflicts caused by incompatible or unpinned package versions or newly released versions.

To fix the issue, try:

- Restricting problem packages to recent versions (using `package~=version`)
- Using a constraints file

Sporadic “No such file or directory” errors

If your Read the Docs builds are sporadically failing due to a file missing error, first check the build output to make sure the issue isn’t related to dependencies.

Once you have verified this is not the case, double check your `conf.py` configuration.

Probable cause

The `sphinx-llm` extension by default creates a parallel process that touches build files while the main process (or other extensions, like `sphinx-tags`) could still be using them. This can cause the missing file error.

Resolution

Warning:

Note that this workaround can cause your build times to grow, as the parallel nature of the normal config cuts build times considerably.

To fix this, tell `sphinx-llm` in your `conf.py` to not build in parallel:

```
# Run sphinx-llm markdown generation sequentially to prevent race conditions
llms_txt_build_parallel = False
```

Local deployment issues

Port or address already in use

Possible cause

When you run `make html` or `make run` to start a local development server, the build fails with an error message similar to the following:

```
[sphinx-autobuild] Serving on http://127.0.0.1:8000
[sphinx-autobuild] Waiting to detect changes...
                   ERROR:    [:vale-ignore:`Errno 98``] error while attempting to
bind on address ('127.0.0.1', 8000): address already in use
```

This happens when the port number specified for the local development server is already in use by another process. By default, the local development server runs on port 8000, but if another process is using that port, the build will fail. Many times, `Ctrl-C` is used to stop the local development server, but the process may not have been terminated properly, leaving the port in use. The underlying socket may still be in use, preventing the local development server from starting on the same port.

Resolution

You can use the `SPHINX_PORT` environment variable to specify a different port number for the local development server. For example, to use port 8081, run the following command:

```
SPHINX_PORT=8081 make run
```

An elegant solution is to use `SPHINX_PORT=0` to let the system automatically select an available port for the local development server. Make sure to check the output of the command to see which port was selected, and use the system selected port number to access the local development server in your web browser.

Runtime errors

“&” character in the URL breaks links

A link in the documentation set is broken when the URL contains an “&” character. For example, a link to `https://example.com/?param1=value1&param2=value2` may be rendered as `https://example.com/?param1=value1&param2=value2` in the generated HTML, causing the link to break.

Cause

This is a known bug[55] that has been resolved in the myst-parser extension. However, the latest version of the extension is incompatible with the Sphinx Stack, which retains support for Python 3.10 and Sphinx 7.

Resolution

In `.rst` sources, keep `&` in the URL; Sphinx will escape it correctly in the generated HTML. For example: `https://example.com/?param1=value1&param2=value2`.

If you are using MyST[56], add the link in an `eval-rst` directive. You may also want to include a comment explaining the workaround so it'll be easier to undo later on.

```
```{eval-rst}
.. This URL includes an & which is broken by the current version of myst-parser (fixed in
v5.1.0)

`Link text <https://example.com/?param1=value1&param2=value2>`_
```
```

Read the Docs failures

In this guide, you will find information on how to troubleshoot issues related to building the documentation set on Read the Docs.

Stable version won't build from the latest tag

If your project has the `stable` version configured to build from tags, such as with the default semantic versioning behavior[57], your `stable` version can become out-of-step and continue building a particular tag, even when the repository has newer tags.

Possible causes

An unwanted tag might have been pushed to the repository and then removed. Once Read the Docs creates a version from a tag, it doesn't later verify that the tag still exists, so the version will persist and become a zombie.

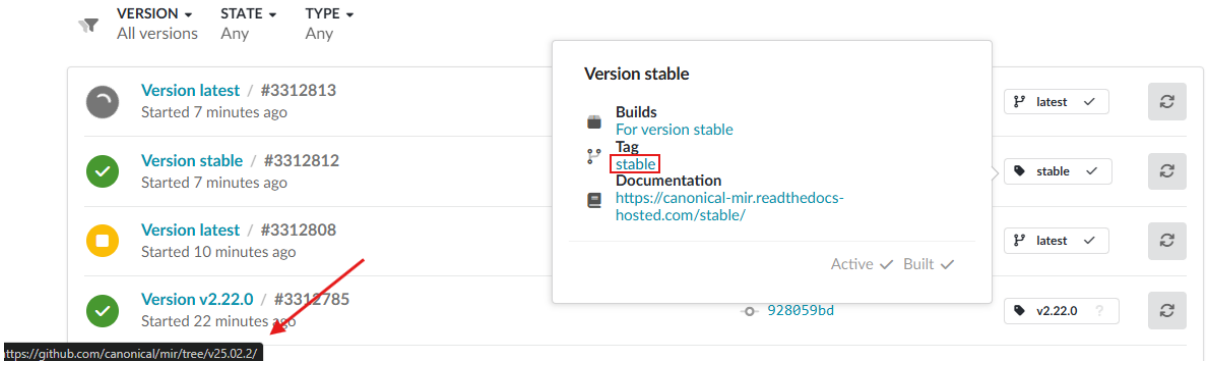
If the unwanted tag is a higher iterator than any existing tag, the zombie version will always take precedence. For example, if tag `20.2.0` was pushed by accident and then replaced with `2.20.0`, the corresponding version 20.2.0 will persist, and `stable` will continue pointing to it.

[55] https://github.com/executablebooks/MyST-Parser/issues/1028
[56] https://myst-parser.readthedocs.io/en/latest/
[57] https://docs.readthedocs.com/platform/stable/versions.html#versions-are-git-tags-and-branches

Diagnosis

There's a roundabout procedure to verify whether your project is affected. Start by opening your project dashboard on Read the Docs.

On the *Builds* tab, locate the most recent `stable` build. For that version, hover over the status indicator. In the hover box, open the *stable* link. If the resulting GitHub page is a 404, then your project has a zombie version.



Resolution

On your project dashboard, open the *Versions* tab and click *Add version*.

Find the zombie version, deactivate it, then update it.

Rebuild `stable` by retriggering it on the dashboard or pushing a new tag to the repository.

Issue tracking

readthedocs/readthedocs.org#12450[58]

Creating a new version slug fails

When you try to create a new version, you may see an error stating that the slug is already in use, even though you don't visually see the version occupying it in the RTD GUI.

Possible causes

This occurs when a version is deactivated in the RTD GUI, for example by turning off its *Active* switch, and someone later tries to create a new version with the same slug. Read the Docs has no delete option in its web GUI, so the deactivated version continues to occupy that slug indefinitely.

When a version is deactivated (see Inactive versions[59]), its documentation content is deleted and builds can no longer be triggered from the GUI. However, the version itself remains and

[58] https://github.com/readthedocs/readthedocs.org/issues/12450
[59] https://docs.readthedocs.com/platform/stable/versions.html#version-states

retains its slug. It may not appear in the *Versions* tab, making it effectively invisible in the web GUI while still causing the "slug is already in use" error.

Resolution

The easiest way to deal with an invisible inactive version is to open its settings page directly via URL:

```
https://app.readthedocs.com/dashboard/<project-name>/version/<slug>/edit/
```

For example:

```
https://app.readthedocs.com/dashboard/canonical-kafka-charm/version/latest/edit/
```

From there, you can make the version active again, which will make it visible in the web GUI, or change its slug.

Authentication errors

At times, documentation projects that were building successfully may suddenly experience build failures due to authentication issues with GitHub.

Possible cause

A common cause of sudden authentication failures is that the GitHub Webhook goes out of sync due to infrastructure issues on RTD or GitHub. Those issues are usually resolved with time, rather than manual action.

Resolution

Confirm that the git repository URL setting in Read the Docs points to a valid repository. Verify that the public SSH key from your Read the Docs project is installed as a deploy key on your the GitHub repository. If these are already in place then try to resynchronize the webhook.

If the above steps do not solve the problem, delete and reinstate the webhook[60].

[60] https://docs.readthedocs.com/platform/latest/guides/setup/git-repo-manual.html#manual-integration-setup

`readthedocs.yaml` file issues

The GitHub pull request build fails stating the `.readthedocs.yaml` file is not accessible.

Probable cause

Inaccessibility of the `.readthedocs.yaml` file in the build system can be caused if the file is missing or if it is not in the location configured at RTD.

Resolution

Check the location of the `.readthedocs.yaml` file in your repository to ensure it is in the root directory (default location). Read the Docs requires a `.readthedocs.yaml` file in the repository root to trigger a build; if this file is missing, the build will fail.

The Canonical Sphinx Stack assumes, by default, that documentation content lives under `/docs/` and that `.readthedocs.yaml` is in the repository root, but neither location is a hard requirement.

If your project does not use the default root directory, ensure that `.readthedocs.yaml` exists and specify its correct location in the RTD build configuration file as described in the Read the Docs documentation[61].

Optional features and customisation

As your documentation grows, you may need more advanced features to support richer content. This can include customising the build system, adding diagrams as code, rendering API references, interactive tables, and custom HTML.

While some of these features are available by default in the Sphinx Stack, others may require additional extensions.

The following guides will help you get started:

Optional customisation

After the initial setup, you can customize your documentation to fit your project's needs. This includes adding extensions, modifying the build process, and adding custom templates.

[61] https://docs.readthedocs.com/platform/stable/guides/setup/monorepo.html#how-to-use-a-readthedocs-yaml-file-in-a-sub-folder

Redirect pages

If a file in your documentation set is moved, deleted or renamed, it can no longer be found at its original location and users will be shown a 404 Not Found page. This can be frustrating for users.

To provide a better user experience, it is good practice to set up redirects to point to the new location, new file name, or to an alternative place where the information can be found. With documentation on Read the Docs, you can configure redirects manually via the admin panel[62]. However, since this must be done separately, configuring redirects in RTD doesn't scale for complex and mature documentation where a lot of redirects must be maintained.

The **best practice** is to define and track redirects in your project source, so that when a file is moved, renamed or deleted, you can create a redirect at the same time. In this way, redirects will always be in place when such changes go live. Redirects can either be **external**, meaning that they point to a URL outside your documentation, or **internal**, where you want to point to a new location within your documentation.

To enable handling redirects in your project repository, the Sphinx Stack comes with the sphinx-rerediraffe extension. This extension allows you to store redirects in a separate file, `redirects.txt`.

Configure sphinx-rerediraffe

Create a file called `redirects.txt` in the same directory as your `conf.py` file. This file will host all of your redirects.

Next, in your `conf.py` file, declare the `rediraffe_redirects` variable and assign it the path of the redirects file you just created relative to `conf.py`:

Listing 15: conf.py

```
# Add redirects, so they can be updated here to land alongside docs being moved
rediraffe_redirects = "redirects.txt"
```

Define a redirect

Inside your `redirects.txt` file, add your redirects in the following patterns:

Listing 16: redirects.txt

```
# Client-side page redirects
# --------------------------
# Comments start with a single hash (#)
# Each redirect appears on its own line in the format:
#
# redirect/from/ redirect/to/
#
# Paths are relative to the root of the `docs` directory
# The "from" path must be a file that *does not* exist
# The "to" path must be a file that *does* exist
```

(continues on next page)

[62] https://docs.readthedocs.io/en/stable/guides/redirects.html

(continued from previous page)

```
# The separator between "from" and "to" paths is a single space (not a tab)

path/old-name/ path/new-name/        # To rename a file
path/file-name/ new-path/file-name/  # To move a file
path/old-name/ new-path/new-name/    # To move and rename a file
```

Redirects are relative to the root of the `docs/` directory, so for simplicity, it's best to put the redirects.txt file in the root directory. This is an example of a working[63] `redirect.txt` file.

Layer redirects

Multiple files can be redirected to the same target:

- A → D
- B → D
- C → D

Your `redirects.txt` file will look like:

Listing 17: redirects.txt

```
installation tutorial/basic-installation/
basic-installation tutorial/basic-installation/
```

This is common if files have been renamed more than once, creating layers of redirects.

Bridge project and documentation builds

The Sphinx Stack can be used as a standalone docs repository, or embedded inside a parent project. This guide demonstrates how to bridge the docs build with a Python project's main build. Once bridged, project contributors can install, build, and check the docs from the root of the project with the main build system.

Parent projects and the build (page 134) describes the full benefits of bridging the build in a larger project.

Plan and requirements

The bridge is built by making up to three changes to the build.

The bridge **shims the docs build targets in the main build**. Any build system is capable of adding targets that call other systems. When shimmed, the docs targets like `html` and `clean` pass through to `docs/Makefile`, with arguments.

The bridge also **merges the virtual environments**, removing the need for a separate docs environment. This change is optional but recommended. To combine environments, your project must provide **Python 3.11** or higher to the Sphinx Stack. Any Python dependency manager will do, and this guide illustrates with three:

[63] https://github.com/canonical/ubuntu-server-documentation/blob/main/docs/redirects.txt

- pip 25.1 and higher
- uv 0.4.27 and higher
- Poetry 1.2.0a2 and higher

After adding the bridge, it's also possible to **adjust the documentation workflows** to use your project's main build. The workflows were designed with Make, so they only work if you use it for your build system.

Example project layout

This guide illustrates the bridge through an example project. In the example project, the file tree contains:

```
Project
│
├── ...
├── docs
│   └── Makefile
├── .venv
├── Makefile
└── pyproject.toml
```

The example project's root `Makefile` has two conventional targets that need adjustment:

- `setup` for building the virtual environment and syncing dependencies
- `clean` for cleaning up the virtual environment and temporary files

Set the build paths

Where your main build sets environment variables, redeclare the docs environment variables that specify the build paths:

- `DOCS_BUILDDIR` is the destination for the docs. If you have special distribution needs you can override this, but for most builds this can be left as-is.
- `DOCS_VENVDIR` is the virtual environment of the docs. If you're merging the virtual environments, set this as a relative path from the docs directory to your project's virtual environment.
- `VALE_DIR` is the path to the Vale binary. The full path depends on the location of your virtual environment, so it's best to copy this as-is.

In the example project, this looks like:

Listing 18: Makefile

```
# Env vars for the docs build
export DOCS_BUILDDIR ?= _build
export DOCS_VENVDIR ?= ../.venv
export VALE_DIR ?= $(DOCS_VENVDIR)/lib/python*/site-packages/vale
```

Integrate the docs setup

The next step is to incorporate the docs installation target, and optionally the dependencies, into the main build.

Separate virtual environments

If you don't plan to merge the virtual environments, override the installation target by calling all three doc installation targets in a row.

In the example project, this is written as:

Listing 19: Makefile

```
# Override the
.PHONY: docs-install
docs-install:
    $(MAKE) -C docs install --no-print-directory
    $(MAKE) -C docs vale-install --no-print-directory
    $(MAKE) -C docs pymarkdownlnt-install --no-print-directory
```

Merged virtual environments

To merge virtual environments, the `setup` target must handle both development and docs packages, and enumerate all docs packages in `pyproject.toml`. Your project will also need dependency groups to organize the packages. The result will be one virtual environment fed by three dependency groups in `pyproject.toml`:

- `dev` for development builds
- `docs` for extra docs packages that your project needs
- `docs-sphinx-stack` for the core docs packages set by the Sphinx Stack

First, add the dependency groups. The docs group should depend on the Sphinx Stack group:

Listing 20: pyproject.toml

```
[dependency-groups]
dev = [
    # Packages for main development and testing
]
docs = [
    # Packages for extra docs features
    {include-group = "docs-sphinx-stack"},
]
docs-sphinx-stack = [
    # Core docs packages
]
```

If your `pyproject.toml` file didn't already have a `dev` dependency group, review the packages listed in the `dependencies` key, then move any non-runtime dependencies to the `dev` dependency group.

If your project needs extra docs features, like the Mermaid or LaTeX Sphinx extensions, add their packages to the `docs` group.

Copy the contents of `docs/requirements.txt` into the `docs-sphinx-stack` group.

In the main build, override the docs installation target to install the `docs` dependency group, then make the project's `setup` target depend on the docs target. In the example project, it's written like this:

pip

Listing 21: Makefile

```
.PHONY: setup
setup: docs-install
    pip install --group dev

# ...

# Override for `install` target in docs project
.PHONY: docs-install
docs-install:
    pip install --group docs
    $(MAKE) -C docs vale-install --no-print-directory
    $(MAKE) -C docs pymarkdownlnt-install --no-print-directory
```

uv

Listing 22: Makefile

```
.PHONY: setup
setup: docs-install
    uv sync --group dev

# ...

# Override for `install` target in docs project
.PHONY: docs-install
docs-install:
    uv sync --no-dev --group docs
    $(MAKE) -C docs vale-install --no-print-directory
    $(MAKE) -C docs pymarkdownlnt-install --no-print-directory
```

Poetry

Listing 23: Makefile

```
.PHONY: setup
setup: docs-install
    poetry install --only dev

# ...

# Override for `install` target in docs project
.PHONY: docs-install
docs-install:
    poetry install --only docs
    $(MAKE) -C docs vale-install --no-print-directory
    $(MAKE) -C docs pymarkdownlnt-install --no-print-directory
```

If your docs aren't written in Markdown, remove the command that runs the `pymarkdownlnt-install` target.

Shim the remaining targets

The docs build has many targets, but only a handful of them overlap or collide with most project builds, so we only need to override two more. The rest can pass straight through to the docs build.

In the example project, the main build calls the targets like this:

Listing 24: Makefile

```
# Override for `clean` target in docs project. We don't want to touch `.venv`,
# so we pass a temp dir instead.
.PHONY: docs-clean
docs-clean:
    DOCS_VENVDIR=$(mktemp) $(MAKE) -C docs clean --no-print-directory

# Override for `help` target
.PHONY: docs-help
```

(continues on next page)

(continued from previous page)

```
docs-help:
    @echo "Commands in the documentation subproject:"
    $(MAKE) -C docs help --no-print-directory
    @echo "Run these commands with 'make docs-<command>' in the project root."

# Shim for the rest of the targets in docs Makefile
.PHONY: docs-%
docs-%: docs-install
    $(MAKE) -C docs $(@:docs-%=%) --no-print-directory
```

Variables and Makefiles

When calling another Makefile with `$(MAKE) -C`, also known as a sub-Make call, variables with default values in the child Makefile won't be overridden. To override them, you must set them explicitly with *export* or as as command-line arguments.
For example, within the main build, if you need to customize `SPHINX_AUTOBUILD_OPTS`, pass it to the docs build like this:

Listing 25: Makefile

```
        $(MAKE) -C docs run SPHINX_AUTOBUILD_OPTS="$(SPHINX_AUTOBUILD_OPTS)"
```

Adjust the Read the Docs build

With the Makefile in a different location than usual, and its being a separate process, it's simplest to override the Read the Docs build in `.readthedocs.yaml` to call the same build targets that developers use locally.

If you use an uncommon system, you might need to install it during the workflow's `create_environment` job.

If you merged the virtual environments, make sure to set `DOCS_VENVDIR=${READTHEDOCS_VIRTUALENV_PATH}` in all commands.

Here's what it looks like in the example project:

Listing 26: .readthedocs.yaml

```
build:
  os: ubuntu-24.04
  tools:
    python: "3.12"
  jobs:
    post_checkout:
      - git fetch --tags --unshallow # Also fetch tags
    create_environment:
      - python3 -m venv "${READTHEDOCS_VIRTUALENV_PATH}"
    install:
      - make docs-install DOCS_VENVDIR="${READTHEDOCS_VIRTUALENV_PATH}"
    build:
    html:
      - make docs DOCS_VENVDIR="${READTHEDOCS_VIRTUALENV_PATH}" DOCS_BUILDDIR="
$READTHEDOCS_OUTPUT/html/"
```

Adjust the doc workflows

If your project uses the Sphinx Stack workflows *and* Make, adjust the workflows to use the bridged targets.

For the main checks, override the target names and paths through the workflow inputs[64]:

Listing 27: .github/workflows/automatic-doc-checks.yml

```
jobs:
  documentation-checks:
    uses: canonical/documentation-workflows/.github/workflows/documentation-checks.
yaml@main
    with:
      working-directory: "."
      fetch-depth: 0
      install-target: docs-install
      spelling-target: docs-spelling
      woke-target: docs-woke
      linkcheck-target: docs-linkcheck
      pa11y-target: docs-pa11y
```

If your docs are written in Markdown, override the path and command inputs in the Markdown linter workflow:

Listing 28: .github/workflows/markdown-style-checks.yml

```
- name: Create venv
  working-directory: "."
  run: make docs-install
- name: Lint markdown
  working-directory: "."
  run: make docs-lint-md
```

Enable Google Analytics

Many documentation sites collect page view and user behavior data with Google Analytics. Collecting this data requires the user's consent.

Once you're through with the guide, analytics will be collected on all public versions of your documentation.

[64] https://github.com/canonical/documentation-workflows/blob/main/.github/workflows/documentation-checks.yaml#L5-L54

Add or update templates

The header template inserts the Google Analytics tracking tag onto each page. If a user consents to being tracked, their traffic and search data will be collected with each page interaction.

The footer template provides a link for users to change their data collection preferences.

The process for sourcing these files depends on your Sphinx Stack version. Check which version is listed in your project's `docs/_dev/version` file.

On Sphinx Stack 1.6 or higher

Starting in version 1.6 of the Sphinx Stack, these templates are available by default. If you're on one of these versions, skip ahead to *Update your configuration file* (page 54).

On Sphinx Stack 1.5 or lower

In version 1.5 of the Sphinx Stack and lower, the templates were sourced from a separate repository.

Download the latest version of the templates:

- `header.html`[65]
- `footer.html`[66]

Next, create a `docs/_templates` directory and move the templates into it.

If you find that your project already has a `header.html` file, add the following HTML immediately after the opening `<header>` tag:

```
<!-- Google Tag Manager -->
<script>
(function(w, d, s, l, i) {
  w[l] = w[l] || [];
  w[l].push({
    'gtm.start': new Date().getTime(),
    event: 'gtm.js'
  });
  var f = d.getElementsByTagName(s)[0],
    j = d.createElement(s),
    dl = l != 'dataLayer' ? '&l=' + l : '';
  j.async = true;
  j.src =
    'https://www.googletagmanager.com/gtm.js?id=' + i + dl;
  f.parentNode.insertBefore(j, f);
})(window, document, 'script', 'dataLayer', 'GTM-KNX3CJC');
</script>
```

Similarly, if your project already has a `footer.html` file, add the following highlighted line to the `right-details` div:

[65] https://github.com/canonical/sphinx-stack/blob/main/docs/_templates/header.html
[66] https://github.com/canonical/sphinx-stack/blob/main/docs/_templates/footer.html

```
<div class="right-details">
  <a href="" class="js-revoke-cookie-manager muted-link">Manage your tracker settings</a>
</div>
```

Update your configuration file

Now that the templates are in place, you need to make Sphinx aware of them in your project's configuration file.

If you're on version 1.6 of the Sphinx Stack or higher, these lines already exist in your configuration file. You only need to uncomment them in the following steps. In lower versions, you need to add them yourself.

Add or uncomment the following line:

Listing 29: conf.py

```
templates_path = ["_templates"]
```

The script and style sheet for the cookie banner are hosted remotely. Include them in your docs by adding or uncommenting the following lines in the `conf.py` file:

Listing 30: conf.py

```
html_css_files = ["https://assets.ubuntu.com/v1/d86746ef-cookie_banner.css"]

html_js_files = ["https://assets.ubuntu.com/v1/287a5e8f-bundle.js"]
```

Test the cookie banner

To verify that everything is working, run the following commands in the `docs/` directory:

```
make clean
make install
make run
```

You may need to clear your browser cache for changes to take effect.

Customise PDF output

Overview

The Sphinx Stack supports PDF output via LaTeX using the `make pdf` command. This build process relies on system packages, Sphinx configurations, and a LaTeX template from the `canonical-sphinx` extension.

Customising PDF output involves two levels of configuration:

- **Sphinx configuration**: built-in options for configuring LaTeX build process in `conf.py`, for example: the engine used to generate the PDF, output file name, and input file paths.

- **LaTeX configuration**: the LaTeX packages, styling, and configuration for the PDF output, set through the `latex_elements`[67] dictionary in the project `conf.py`. In the Sphinx Stack, a default set of LaTeX elements is provided by the `canonical-sphinx` extension. Changing the LaTeX configuration requires overriding the default values loaded from the extension.

This guide covers common practices and tips for customising PDF output from your project using the Sphinx Stack and the `canonical-sphinx` extension.

For basic instructions about building the PDF, see *Build and preview* (page 12).

Configure document properties

The `latex_documents`[68] variable in the Sphinx `conf.py` file controls properties of the intermediate LaTeX file and the final PDF output. The values are defined in a tuple of six elements:

```
latex_documents = [
    (
        'startdocname',   # Root document (e.g., 'index' or 'pdf-index')
        'targetname.tex', # Output LaTeX file name (no spaces)
        'title',          # Document title (can be empty to use the root doc's title)
        'author',         # Author name(s).
        'theme',          # Document type: 'manual' or 'howto'
        True,             # toctree_only: if True, only include docs in toctree
    ),
]
```

- `startdocname`: The root document for the PDF (without the `.rst` extension).
- `targetname`: The filename for the generated LaTeX source file. The PDF file name is derived from this filename with the `.pdf` extension. Blank space characters are not allowed.
- `title`: The title for the PDF document on the cover page.
- `author`: The author(s) of the document. Use `\\and` to separate multiple authors.
- `theme`: Either `manual` or `howto`.
  - `manual`: This is the default and is intended for comprehensive, book-style documentation. It produces a PDF with chapters, sections, and a table of contents. Use this for user guides, reference manuals, or any documentation that should be structured as a book.
  - `howto`: This type is for shorter, task-oriented documents. It produces a simpler PDF without chapters, and is best for single-topic guides or tutorials.
- `toctree_only`: Boolean. If set to True, the `startdocname` document itself is not included in the output, only the documents referenced by the `toctree` directive are included. This is useful for creating a PDF-specific index file.

For more details, see latex_documents[69] in the Sphinx documentation.

[67] https://www.sphinx-doc.org/en/master/latex.html#the-latex-elements-configuration-setting
[68] https://www.sphinx-doc.org/en/master/usage/configuration.html#confval-latex_documents
[69] https://www.sphinx-doc.org/en/master/usage/configuration.html#confval-latex_documents

Change PDF document filename

By default, the PDF output filename is derived from the `project` name in the `conf.py` file: lowercase characters with blank spaces removed.

To override the filename, update the second element (`targetname`) of the `latex_documents` tuple in `conf.py`. The following example shows how to replace blank spaces in the project name with underscores:

```
latex_documents = [
    (
      'index',
      f"{project.replace(' ', '_')}.tex",
      '',
      'Author Name',
      'manual',
      True,
    ),
]
```

Change PDF document title

By default, the PDF title on the cover page comes from the title of the main index document. To override it, update the third element (title) of the `latex_documents` tuple in `conf.py`. Use an empty string (`''`) to keep the default behaviour.

Use a different index document for PDF builds

Because the PDF output has a different usage and structure from the HTML output, it is sometimes useful to create a PDF-specific index document. For example, you may want to create a PDF-specific index file that includes only a subset of the pages in the HTML index.

To use a different index document for PDF builds:

1. Create a PDF-specific index document, for example, `pdf-index.rst`.
2. Update the first element (`startdocname`) of the `latex_documents` tuple in `conf.py` to point to the new index document.
3. Set the last element (`toctree_only`) of the `latex_documents` tuple in `conf.py` to `False` to ensure only referenced documents are included in the PDF output.
4. Exclude the PDF-specific index document from the HTML build. This is done by changing the `exclude_patterns` list in `conf.py`:

```
# Identify the Sphinx builder being used
if '-b' in sys.argv:
    builder = sys.argv[sys.argv.index('-b') + 1]
elif '-M' in sys.argv:
    builder = sys.argv[sys.argv.index('-M') + 1]
else:
    builder = 'html'  # default builder
```

(continues on next page)

(continued from previous page)

```
# Exclude the PDF-specific index from the HTML build
if builder in ['html', 'dirhtml']:
    exclude_patterns.append('pdf-index.rst')
```

Check both the HTML and PDF outputs to confirm that different index documents are used for each output.

Note:

The order and hierarchy of your `toctree` entries determine the chapters and sections in the PDF.
Any headings placed before the main `toctree` in your root document will cause all referenced documents to be nested under that heading in the PDF. To avoid this, do not add extra headings before the `toctree`.

Override the LaTeX template

The LaTeX template is a text file in the `canonical-sphinx` extension that provides the default styling and layout of the PDF document. The template contains a Python dictionary of LaTeX elements, which will be imported by Sphinx when the PDF is built.

Any additions or changes to the default settings of LaTeX elements in the PDF document requires overriding the default template.

1. Download the default template file latex_elements_template.txt[70] from the `canonical/canonical-sphinx` GitHub repository, and save it to your project directory. For example, at `_dev/latex_elements_custom.txt`.
2. In the Python dictionary, add or modify the LaTeX elements you want to change. Details of changing the dictionary are covered in the sub-sections below.
3. In your project's `conf.py` file, add or update the `latex_elements` dictionary to load the local override of the LaTeX template. Change the file path to the location of your local override file.

```
# Replace with the path to your local override file
latex_elements_file = "_dev/latex_elements_custom.txt"

with open(latex_elements_file, "rt") as file:
    latex_config = file.read()
    if latex_elements == {}:
      latex_elements = ast.literal_eval(latex_config)
```

Warning:

Defining other settings directly in `latex_elements` will override the values loaded from the template file or your local file.

[70] https://github.com/canonical/canonical-sphinx/blob/main/canonical_sphinx/theme/PDF/latex_elements_template.txt

Add more LaTeX packages to the preamble

You can use two methods to add additional LaTeX packages to the preamble:

- Add the `extrapackages` key in your local template file:

```
{
    ...
    'extrapackages': r'\usepackage{packagename}',
    ...
}
```

- Modify the values of the `preamble` key in your local template file. This is more flexible for adding LaTeX configurations and commands to the preamble.

Note:

The format of the element values is a multi-line string, so use a raw string with the `r` prefix.

Remove the table of contents

For a short, compact document where navigation is not needed, you may want to remove the table of contents from the PDF output.

To do this, provide a local copy of the default template file, and add a new key `tableofcontents` with an empty string as the value:

```
{
    ...
    'tableofcontents': '',
    ...
}
```

Include images or other assets

If the local template requires additional images or other assets, for example, a custom title page or header, the file paths must be added to the Sphinx `conf.py` file to be included in the PDF build.

Provide a `latex_additional_files` variable in your `conf.py` file as a list of file paths to the additional assets. If the variable already exists, add the new file paths to the list. The paths should be relative to the `conf.py` file.

```
# path relative to the conf.py file
latex_additional_files = [
    'path/to/image.svg',
    'path/to/other-asset.pdf',
]
```

Note:

For better quality in the PDF output, it is recommended to use vector images (like SVG or PDF) rather than raster images (like PNG or JPEG). Raster images may lose quality when scaled up in the PDF.
Do not use `.tex` as suffix, otherwise the file is processed as source files for the PDF build process. Instead, use `.tex.txt` or `.sty` to avoid conflicts with the LaTeX build process.

Use landscape layout

The PDF output uses portrait orientation by default. To use landscape orientation, you need to add more packages to the LaTeX preamble and use a specific LaTeX environment to rotate the content.

1. Add the `extrapackages` key to your local template file, and set the value to the `pdflscape` package:

```
{
    ...
    'extrapackages': r'\usepackage{pdflscape}',
    ...
}
```

Note:

The format of the element values is a multi-line string, so use a raw string with the `r` prefix.

2. Use the landscape environment in your documentation source file, and only in the PDF output.

   Wherever you want a section (such as a wide table or figure) to appear in the landscape view, use the `.. raw:: latex` directive to include raw LaTeX code that opens and closes the landscape environment. Only the content between `\begin{landscape}` and `\end{landscape}` will be rotated:

```
.. only:: latex

    .. raw:: latex

        \begin{landscape}

.. list-table:: Example of a landscape table
   :header-rows: 1

   * - Column 1
     - Column 2
     - Column 3
   * - Data 1
     - Data 2
     - Data 3
```

(continues on next page)

(continued from previous page)

```
.. only:: latex

    .. raw:: latex

        \end{landscape}
```

Check PDF build log files

If you encounter an issue that requires further debugging, check the PDF build logs for more detailed error messages. The full logs are generated in the `_build/latex` output directory, and then cleaned up after the build completes.

To temporarily save the log files for debugging:

1. Open the `Makefile` and locate the `pdf` target. Disable the cleanup step by commenting out the `@rm -r $(BUILDDIR)/latex` line.
2. Run the `make pdf` again.
3. Navigate to the output directory `_build/latex` and check the `*.log` and `*.tex` files.
4. After debugging, restore the cleanup step by uncommenting the same line.

Warning:

Keeping the build log files from the previous build might cause conflicts with the current build

Related

- *Build and preview* (page 12)
- *Configure your project* (page 11)

Use Spread to test commands in documentation

It's challenging to keep documentation in sync with products as they evolve. This process is aided by *Spread*, a test distributor that can work through your documentation and report failures in GitHub workflows.

By using Spread tests, you can rely on the tests as the source of truth for commands in your documentation, enabling fully tested documentation at build time.

What you'll need

- Multipass[71] installed on your machine
- Spread[72] installed on your machine

Warning:

Spread requires elevated permissions to run as root. Use the Go install method[73] recommended in the Spread README to install Spread.

[73] https://github.com/canonical/spread?tab=readme-ov-file#install

Create a test suite

From the root of your project, create the file `spread.yaml` and insert the following contents:

Listing 31: project_name/spread.yaml

```
project: project_name

path: /project_name
```

Match the `project` name to your main directory's name. The `path` designates the directory where the Spread materials exist.

So that Spread knows about your tests, add the following section to the end of `spread.yaml`:

Listing 32: project_name/spread.yaml

```
project: project_name

path: /project_name

suites:
  tests/spread/:
    summary: example test
    systems:
      - ubuntu-24.04-64
```

The `suites` section is how you tell Spread about the various Spread tests in your project along with the systems you want Spread to use. In this example, Spread looks for tests in the `project_name/tests/spread` directory and runs them on Ubuntu 24.04 LTS. If you create a new `task.yaml` file in a different directory, remember to add a corresponding suite for it in `spread.yaml`.

[71] https://documentation.ubuntu.com/multipass/latest/how-to-guides/install-multipass/
[72] https://github.com/canonical/spread

Set up the Multipass backend

Each job in Spread has a backend, or a way to obtain a machine for running your Spread tests. Spread can run on various backends, like Google[74], QEMU[75], or, as this guide sets up, Multipass.

Copy the following `backends` section of `spread.yaml` between the `path` and `suites` sections:

Listing 33: project_name/spread.yaml

```
project: project_name

path: /project_name

backends:
  multipass:
    type: adhoc
    allocate: |
      multipass_image=24.04
      instance_name="example-multipass-vm"

      # Launch Multipass VM
      multipass launch --cpus 2 --disk 10G --memory 2G --name "${instance_name}" "$
{multipass_image}"

      # Enable PasswordAuthentication for root over SSH.
      multipass exec "$instance_name" -- \
        sudo sh -c "echo root:${SPREAD_PASSWORD} | sudo chpasswd"
      multipass exec "$instance_name" -- \
        sudo sh -c \
        "if [ -d /etc/ssh/sshd_config.d/ ]
        then
          echo 'PasswordAuthentication yes' > /etc/ssh/sshd_config.d/10-spread.conf
          echo 'PermitRootLogin yes' >> /etc/ssh/sshd_config.d/10-spread.conf
        else
          sed -i /etc/ssh/sshd_config -E -e 's/^#?PasswordAuthentication.*/
PasswordAuthentication yes/' -e 's/^#?PermitRootLogin.*/PermitRootLogin yes/'
        fi"
      multipass exec "$instance_name" -- \
        sudo systemctl restart ssh

      # Get the IP from the instance
      ip=$(multipass info --format csv "$instance_name" | tail -1 | cut -d\, -f3)
      ADDRESS "$ip"

    discard: |
      instance_name="example-multipass-vm"
      multipass delete --purge "${instance_name}"

    systems:
      - ubuntu-24.04-64:
          workers: 1

suites:
```

(continues on next page)

[74] https://github.com/canonical/spread/blob/master/README.md#google
[75] https://github.com/canonical/spread/blob/master/README.md#qemu

(continued from previous page)

```
  tests/spread/:
    summary: example test
    systems:
      - ubuntu-24.04-64
```

The backends section contains the following pieces:

- The backend is designated as `type: adhoc` as you must explicitly script the procedure to allocate and discard the Multipass VM.
- The `allocate` section defines the image and name of the VM, launches the VM, and sets up the proper SSH permissions Spread then logs in to the VM with root permissions and inserts the Spread test. The last two lines tell Spread the IP address of the Multipass VM and set the environment variable `ADDRESS`.
- The `discard` section deletes the Multipass VM once the Spread test has finished running.
- The `systems` key notes which systems the backend uses. Note that this key must match the `systems` used by at least one test under `suites`.

Create a Spread task

Put your Spread files alongside your project's existing tests. The rest of this guide assumes they're in a top-level `tests/spread` directory.

Each Spread test requires a dedicated `task.yaml` file that contains all the commands you want to test. A single `task.yaml` can help you validate an entire assumed workflow, for instance, an end-to-end tutorial.

An example `task.yaml` file is shown below:

Listing 34: task.yaml

```
summary: Example Spread test

kill-timeout: 5m

prepare: |
  echo "Use this section to install any prerequisites"

execute: |
  echo "This is the first command that Spread will run"

  echo "This is the second command that Spread will run"
```

The `summary` section contains a brief description of the documentation you're testing, the `prepare` section contains any initial setup your test needs, and the `execute` section contains your documentation's commands. The `kill-timeout` option has a default of 10 minutes and doesn't need to be included if you expect your test to complete in that time frame.

Note:

For a real-world example, see `task.yaml` for the Rockcraft Go tutorial.[76]

[76] https://github.com/canonical/rockcraft/blob/main/docs/tutorial/code/go/task.yaml

Include the tested commands in documentation

By using the `literalinclude` directive in Sphinx, you can insert the exact commands from `task.yaml` in your documentation file.

For example, consider the following `task.yaml` file:

Listing 35: task.yaml

```
summary: Clone and build the Sphinx STack

kill-timeout: 5m

execute: |
  # [docs:clone-sphinx-stack]
  git clone https://github.com/canonical/sphinx-stack.git
  # [docs:clone-sphinx-stack-end]

  # [docs:build-documentation]
  cd sphinx-stack/docs
  make run
  # [docs:build-documentation-end]
```

Include the commands from `task.yaml` in your documentation with:

reStructuredText

Listing 36: Example `literalinclude` blocks

```
Clone the Sphinx Stack:

.. literalinclude:: relative-path-to/task.yaml
    :language: bash
    :start-after: [docs:clone-sphinx-stack]
    :end-before: [docs:clone-sphinx-stack-end]
    :dedent: 2

Enter the ``docs`` directory and build the project:

.. literalinclude:: relative-path-to/task.yaml
    :language: bash
    :start-after: [docs:build-documentation]
    :end-before: [docs:build-documentation-end]
    :dedent: 2
```

Listing 37: Example `literalinclude` blocks

````
Clone the Sphinx Stack:

```{literalinclude} relative-path-to/task.yaml
:language: bash
:start-after: [docs:clone-sphinx-stack]
:end-before: [docs:clone-sphinx-stack-end]
:dedent: 2
```

Enter the `docs` directory and build the project:

```{literalinclude} relative-path-to/task.yaml
:language: bash
:start-after: [docs:build-documentation]
:end-before: [docs:build-documentation-end]
:dedent: 2
```
````

By using the options `:start-after:` and `:end-before:`, the documentation file sources and includes all commands appearing in `task.yaml` between the specified lines.

Run tests locally

List all available Spread tests in the code repository:

```
spread --list
```

The terminal should respond with all the tests defined in `spread.yaml`. For example:

```
user@host:project_name$ spread --list

multipass:ubuntu-24.04-64:tests/spread/example_documentation_test
```

Run all Spread tests locally with `spread`. You can also run a single Spread test by specifying:

```
spread -vv -debug multipass:ubuntu-24.04-64:tests/spread/example_documentation_test
```

Depending on the complexity of your test, Spread can take several minutes to complete. The `-vv -debug` flags provide useful debugging information as the test runs.

Check the results

During runtime, the terminal outputs various messages about allocating the Multipass VM, connecting to the VM, sending the Spread test to the VM and executing the test. If the test is successful, the terminal will output something similar to the following:

```
2025-02-04 16:17:10 Successful tasks: 1
2025-02-04 16:17:10 Aborted tasks: 0
```

Another sign of a successful test is whether the Multipass VM was deleted as expected. Check by running `multipass list`, and if the Spread test was successful (and you have no other Multipass VMs created at the time), the terminal should respond with:

```
user@host:project_name$ multipass list

No instances found.
```

If the Spread test failed, then the `-debug` flag will open a shell into the Multipass VM so that additional debugging can happen. In that case, the terminal will output something similar to the following:

```
2025-02-04 16:17:10 Starting shell to debug...
2025-02-04 16:17:10 Sending script for multipass:ubuntu-24.04-64 (multipass:ubuntu-
24.04-64:tests/spread/example_documentation_test):
```

Use custom HTML templates

If the default template in the Sphinx Stack doesn't fully meet your needs – whether you want a unique layout, a custom header or footer, or a specialized sidebar for certain pages – you can create and use a custom template for your project.

This guide shows you how to extend or override the default templates in the Sphinx Stack to tailor the look and structure of your documentation.

Note:

Base template customizations can be made to your documentation. However, they are not officially supported by the team maintaining the Spinx Stack. Use them at your own discretion.

Setup

First, create the `docs/_templates` directory; all your custom templates will need to be stored in this directory.

Then uncomment this line in `docs/conf.py` so your project will use local templates (where available):

Listing 38: conf.py

```
templates_path = ["_templates"]
```

In most cases, you will need to copy the default templates from the canonical-sphinx theme[77] as a starting point and edit as needed.

See also

Sphinx uses the Jinja templating engine for its HTML templates; see the Jinja template syntax reference[78] for more details.

Use custom template for all pages

Sphinx looks for a template called `page.html` as the entry point and main page template for documentation pages. To customize your project's look and structure, check this file and determine which parts – such as the header, footer, or sidebars – need to be edited or overridden.

Here are some examples.

Remove on-page TOC

To remove the on-page TOC in the right sidebar, make a copy of page.html[79] in the `docs/_templates` directory, and remove the applicable lines. This will apply to all pages.

Listing 39: page.html

```
{% block right_sidebar %}
<div class="toc-sticky toc-scroll">
  {% if not furo_hide_toc_orig %}
   <div class="toc-title-container">
     <span class="toc-title">
      {{ _("Contents") }}
     </span>
   </div>
   <div class="toc-tree-container">
     <div class="toc-tree">
       {{ toc }}
     </div>
```

(continues on next page)

[77] https://github.com/canonical/canonical-sphinx/tree/main/canonical_sphinx/theme/templates
[78] https://jinja.palletsprojects.com/en/latest/templates/
[79] https://github.com/canonical/canonical-sphinx/blob/main/canonical_sphinx/theme/templates/page.html

(continued from previous page)

```
    </div>
   {% endif %}
    {% if meta and ((meta.discourse and discourse_prefix) or meta.relatedlinks) %}
    <div class="relatedlinks-title-container">
      <span class="relatedlinks-title">
```

Add icon for GitHub link in header

To customize the default header by adding an icon for the GitHub link, first make a copy of header.html[80] in the `docs/_templates` directory.

Then modify the conditional statement related to the GitHub URL with your code.

Listing 40: header.html

```
[...]
          {% if github_url %}
          <li>
            <a href="{{ github_url }}" class="p-navigation__sub-link p-dropdown__link">
              <!-- GitHub icon (inline SVG) -->
              <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0
16 16" fill="currentColor" style="vertical-align:middle; margin-right:4px;">
                <path d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59.4.07.55-.17.55-
.38
                0-.19-.01-.82-.01-1.49-2.01.37-2.53-.49-2.69-.94-.09-.23-.48-.94-.82-1.13-
.28-.15-.68-.52
                -.01-.53.63-.01 1.08.58 1.23.82.72 1.21 1.87.87 2.33.66.07-.52.28-.87.51-
1.07-1.78-.2-3.64-.89
                -3.64-3.95 0-.87.31-1.59.82-2.15-.08-.2-.36-1.02.08-2.12 0 0 .67-.21 2.2.
82a7.65 7.65 0 012 0c1.53-1.03
                2.2-.82 2.2-.82.44 1.1.16 1.92.08 2.12.51.56.82 1.28.82 2.15 0 3.07-1.87
3.75-3.65 3.95.29.25.54.73.54 1.48
                0 1.07-.01 1.93-.01 2.2 0 .21.15.46.55.38A8.013 8.013 0 0016 8c0-4.42-3.
58-8-8-8z"/>
              </svg>
              GitHub
            </a>
          </li>
          {% endif %}
[...]
```

[80] https://github.com/canonical/canonical-sphinx/blob/main/canonical_sphinx/theme/templates/header.html

Use custom template for specific pages

If you want to use a custom template for specific pages in your project, you can do so by using conditional logic in `page.html`.

First, create the base template with your modifications (e.g. `special-header.html`, `special-page.html`) and place it in the `docs/_templates` directory.

Next, make a copy of page.html[81].

Partial template changes

To make partial changes (e.g. custom header) to specific pages, modify only the relevant parts of `page.html` where you want the custom layout or behavior to apply. For example, wrap the body block in a conditional statement so the custom header (e.g. `special-header.html`) applies only to the "how-to/custom-templates" and "how-to/build" page.

Listing 41: _templates/page.html

```
{% block body -%}
   {% if pagename in ["how-to/build", "how-to/custom-templates"] %}
       {% include "special-header.html" %}
   {% else %}
       {% include "header.html" %}
   {% endif %}
   {{ super() }}
{%- endblock body %}
```

Whole template changes

To make changes to the whole template (e.g. a custom layout for a landing page or marketing page), modify the `extends` statement in `page.html` to specify the pages that will use different templates. For example, the `special-page.html` template applies only to the "how-to/customise" and "how-to/diagrams-as-code" page.

Listing 42: _templates/page.html

```
{% if pagename in ["how-to/customise", "how-to/diagrams-as-code"] %}
    {% extends "special-page.html" %}
{% else %}
    {% extends "furo/page.html" %}
{% endif %}
```

Note:

The pages "how-to/customise" and "how-to/diagrams-as-code" will use `special-page.html` as the base template, but all other blocks (e.g. footer, body, etc) will follow the default `page.html`.

[81] https://github.com/canonical/canonical-sphinx/blob/main/canonical_sphinx/theme/templates/page.html

Add page-specific configuration

You can override some global configuration for specific pages.

For example, you can configure whether to display Previous/Next buttons at the bottom of pages by setting the `sequential_nav` variable in the `docs/conf.py` file.

```
html_context = {
    ...
    "sequential_nav": "both"
}
```

You can then override this default setting for a specific page (for example, to turn off the Previous/Next buttons by default, but display them in a multi-page tutorial).

To do so, add file-wide metadata[82] at the top of a page. See the following examples for how to enable Previous/Next buttons for one page:

reST:

```
:sequential_nav: both

[Page contents]
```

MyST:

```
---
sequential_nav: both
---

[Page contents]
```

Possible values for the `sequential_nav` field are `none`, `prev`, `next`, and `both`. See *UI behavior* (page 95) for descriptions of each value.

Another example for page-specific configuration is the `hide-toc` field (provided by Furo[83]), which can be used to hide the page-internal table of content. See Hiding Contents sidebar[84].

Manage sitemaps for versioned documentation

Use this guide to publish a sitemap index for versioned documentation.

The `sphinx-sitemap` extension doesn't support multiple versions by default. Configuring your versioned documentation to use an appropriate version may be sufficient, as search engines and other web systems crawl websites for the purposes of indexing.

If you want sitemaps for all your documentation's versions, you need to deploy your own `robots.txt` file and sitemap index. Supporting multiple versions is recommended for documentation with LTS releases because it makes past versions more prominent to search engines.

[82] https://www.sphinx-doc.org/en/master/usage/restructuredtext/field-lists.html
[83] https://pradyunsg.me/furo/quickstart/
[84] https://pradyunsg.me/furo/customisation/toc/

Note:

If you are implementing a sitemap on an RTD instance that is not a subproject, and it uses `{link}` for the `sitemap_url_scheme`, RTD will replace your sitemap with their own. This is a known bug. The only current workaround is to use a different sitemap name[85] and a custom `robots.txt` pointing to it.

[85] https://sphinx-sitemap.readthedocs.io/en/latest/advanced-configuration.html#changing-the-filename

Create a sitemap index

Before you begin, ensure each version of your documentation has a sitemap generated by the `sphinx-sitemap` extension with the appropriate version. For the available sitemap settings and their default values, see *Sitemap configuration* (page 92).

Next, create a `sitemapindex.xml` file in the same directory as the `conf.py` configuration file, and point to the sitemap files of each of your documentation sets.

Take the Sphinx Stack for an example. Assume that it has three versions, 1.0, 2.0, and 3.0, and uses the URL schema of `<version>/<filename>`.

Listing 43: sitemapindex.xml

```
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<sitemap>
<loc>https://canonical-sphinx-stack.readthedocs-hosted.com/stable/sitemap.xml</loc>
</sitemap>
<sitemap>
<loc>https://canonical-sphinx-stack.readthedocs-hosted.com/3.0/sitemap.xml</loc>
</sitemap>
<sitemap>
<loc>https://canonical-sphinx-stack.readthedocs-hosted.com/2.0/sitemap.xml</loc>
</sitemap>
<sitemap>
<loc>https://canonical-sphinx-stack.readthedocs-hosted.com/1.0/sitemap.xml</loc>
</sitemap>
</sitemapindex>
```

Select the sitemap that crawlers use

A `robots.txt` file specifies the sitemap that crawlers use to index a website. Create the file in the same directory as the configuration file.

If necessary, block any paths you don't want crawled. Google describes how to do this in How to write and submit a robots.txt file[86].

At the end of `robots.txt`, point to the future path of `sitemapindex.xml`:

[86] https://developers.google.com/crawling/docs/robots-txt/create-robots-txt#create_rules

Listing 44: robots.txt

```
Sitemap: https://canonical-sphinx-stack.readthedocs-hosted.com/stable/sitemapindex.xml
```

Publish the sitemap index and robots file

Add both files to the configuration file:

Listing 45: conf.py

```
html_extra_path = [
    "sitemapindex.xml",
    "robots.txt",
]
```

This provides a `sitemapindex.xml` file, which points to the `sphinx-sitemap` generated sitemap for each version.

Automate sitemap-index generation

You may want to automate the generation of the `sitemapindex.xml` file. To see how this is done for the Ubuntu documentation library project, which generates a sitemap containing subproject sitemaps, see the script here[87].

Create diagrams as code using Mermaid

Diagrams help users quickly understand and visualize complex ideas, but they can easily become outdated and difficult to maintain. Creating diagrams as code solves this by keeping them alongside the software source, making updates and reviews simpler.

Mermaid is a popular choice that allows diagrams to be created, maintained, and updated directly in the documentation. This guide explains how to set up the sphinxcontrib-mermaid[88] extension and use Mermaid diagrams in your documentation, with examples included.

Note:

While there are many other tools and/or approaches for creating diagrams in visualizations in your documentation (e.g. C4 model[89], Dia[90], PlantUML[91], Structurizr[92], etc), we only provide support for `sphinxcontrib-mermaid` in the Sphinx Stack.

[89] https://c4model.com/
[90] http://dia-installer.de/
[91] https://plantuml.com/
[92] https://structurizr.com/

[87] https://github.com/canonical/ubuntu-documentation-library/blob/main/scripts/generate_sitemap.py
[88] https://sphinxcontrib-mermaid-demo.readthedocs.io/en/latest/index.html

Installation and setup

First, add the sphinxcontrib-mermaid extension to your `requirements.txt` file so it's installed as a dependency:

```
canonical-sphinx[full]
packaging
sphinxcontrib-svg2pdfconverter[CairoSVG]
sphinx-last-updated-by-git
sphinx-sitemap
sphinxcontrib-mermaid
```

Then add `sphinxcontrib.mermaid` to the `extensions` list in your `conf.py` file:

```
extensions = [
    [...]
    "sphinxcontrib.mermaid",
]
```

You are now ready embed Mermaid diagrams and visualization in your documentation using the `mermaid` directive.

Optional configuration

You can further configure Mermaid's default settings in your project's `conf.py`, such as specifying the image output format (e.g., "png", "raw"), enabling zoom on diagrams, or pinning the Mermaid version[93] used for rendering.

See Mermaid configuration values[94] for more information.

Add a new diagram

Use the `mermaid` directive to embed a Mermaid diagram into your documentation. You start by declaring the type of diagram (e.g. `flowchart`, `sequenceDiagram`, `timeline`, etc), followed by definition and contents. It is also possible to specify additional configuration or custom styles, depending on the diagram type.

Some examples will be covered below.

See also

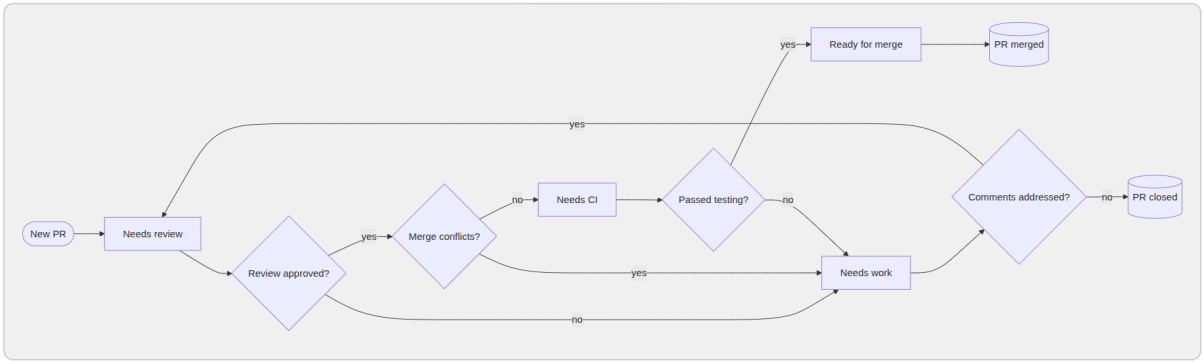
See the Mermaid - Diagram syntax[95] reference for details on the syntax and customization options for each diagram type.

[93] https://unpkg.com/browse/mermaid/
[94] https://sphinxcontrib-mermaid-demo.readthedocs.io/en/latest/#config-values
[95] https://mermaid.js.org/intro/syntax-reference.html

Flowchart diagram with default settings

The left-to-right flowchart below uses the default Mermaid settings.

Diagram



MyST

```
```{mermaid}
flowchart LR
    A([New PR])
    B[Needs review]
    C{Review approved?}
    D{Merge conflicts?}
    E[Needs work]
    F[Needs CI]
    G{Passed testing?}
    H[Ready for merge]
    I[(PR merged)]
    J{Comments addressed?}
    K[(PR closed)]

    A --> B
    B --> C
    C -- no --> E
    C -- yes --> D
    D -- yes --> E
    D -- no --> F
    F --> G
    G -- yes --> H
    G -- no --> E
    H --> I
    E --> J
    J -- yes --> B
    J -- no --> K
```
```

rST

```
.. mermaid::

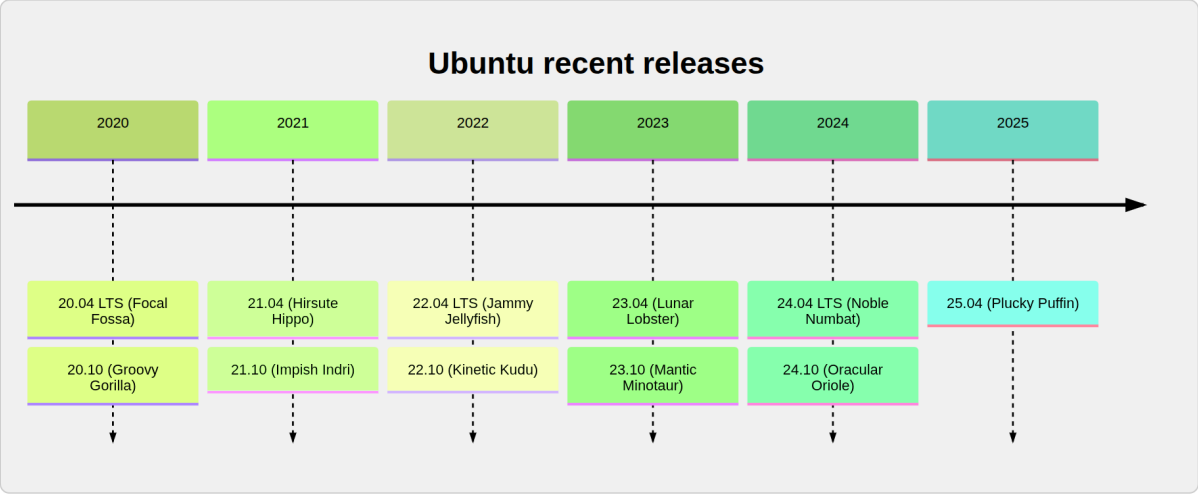
   flowchart LR
       A([New PR])
       B[Needs review]
       C{Review approved?}
       D{Merge conflicts?}
       E[Needs work]
       F[Needs CI]
       G{Passed testing?}
       H[Ready for merge]
       I[(PR merged)]
       J{Comments addressed?}
       K[(PR closed)]

       A --> B
       B --> C
       C -- no --> E
       C -- yes --> D
       D -- yes --> E
       D -- no --> F
       F --> G
       G -- yes --> H
       G -- no --> E
       H --> I
       E --> J
       J -- yes --> B
       J -- no --> K
```

Timeline diagram with pre-defined theme

The timeline diagram below uses a pre-defined Mermaid theme[96].

Diagram



[96] https://mermaid.js.org/syntax/timeline.html#themes

MyST

````
```{mermaid}
%%{init: {"theme":"forest"}}%%
timeline
    title Ubuntu recent releases

    2020 : 20.04 LTS (Focal Fossa) : 20.10 (Groovy Gorilla)
    2021 : 21.04 (Hirsute Hippo) : 21.10 (Impish Indri)
    2022 : 22.04 LTS (Jammy Jellyfish) : 22.10 (Kinetic Kudu)
    2023 : 23.04 (Lunar Lobster) : 23.10 (Mantic Minotaur)
    2024 : 24.04 LTS (Noble Numbat) : 24.10 (Oracular Oriole)
    2025 : 25.04 (Plucky Puffin)
```
````

rST

```
.. mermaid::

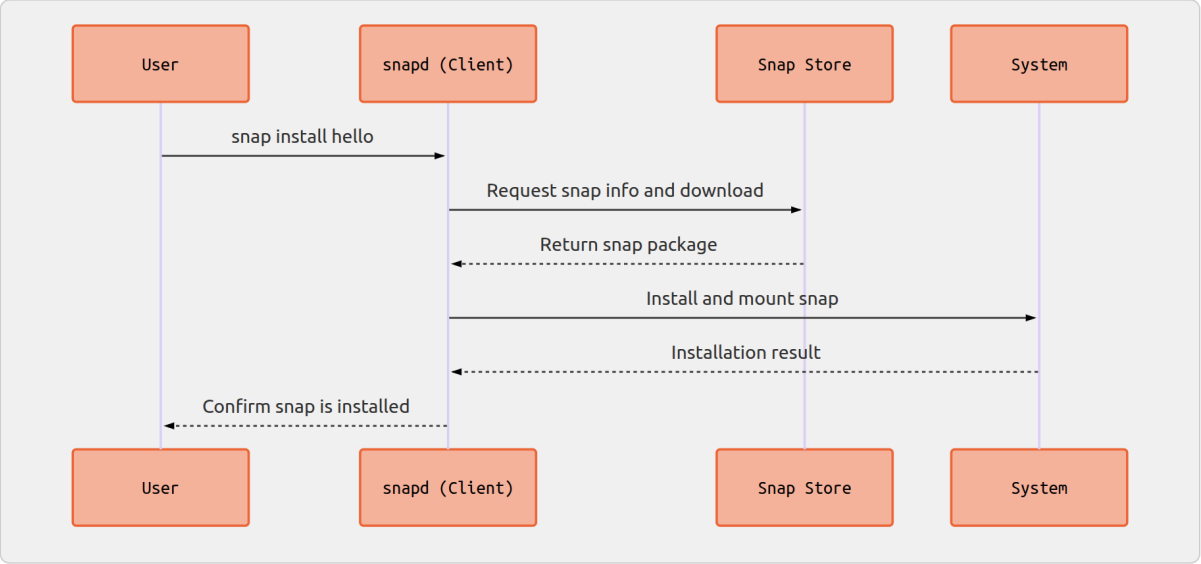
   %%{init: {"theme":"forest"}}%%
   timeline
       title Ubuntu recent releases

       2020 : 20.04 LTS (Focal Fossa) : 20.10 (Groovy Gorilla)
       2021 : 21.04 (Hirsute Hippo) : 21.10 (Impish Indri)
       2022 : 22.04 LTS (Jammy Jellyfish) : 22.10 (Kinetic Kudu)
       2023 : 23.04 (Lunar Lobster) : 23.10 (Mantic Minotaur)
       2024 : 24.04 LTS (Noble Numbat) : 24.10 (Oracular Oriole)
       2025 : 25.04 (Plucky Puffin)
```

Sequence diagram with global custom CSS

The sequence diagram below has custom styling applied using a global CSS file. A global CSS file enables the styles to be easily applied to all sequence diagrams, based on the classes defined in your stylesheet. You can also use the global CSS file to customize the diagrams in dark mode.

Diagram



Custom CSS

First, create a Mermaid stylesheet in the `_static` directory and customize it as needed. For example, `mermaid-sequence.css`.

Then, enable custom stylesheets in `conf.py`. Uncomment this line so Sphinx can use local stylesheets:

Listing 46: conf.py

```
html_static_path = ["_static"]
```

Link to your custom Mermaid stylesheet:

Listing 47: conf.py

```
html_css_files = [
    "mermaid-sequence.css",
]
```

Here's an example stylesheet that changes the default typefaces and colors of Mermaid components. It also sets a solid background color for all Mermaid diagrams.

Listing 48: mermaid-sequence.css

```
.actor {
  stroke: #eb6536 !important;
  stroke-width: 2;
  fill: #f5b29b !important;
}

text.actor {
  fill: black;
  stroke: none;
```

(continues on next page)

(continued from previous page)

```
  font-family: Helvetica;
}

.actor-line {
  stroke: #77216f;
}

.mermaid {
    background-color: #f0f0f0;
    border: 1px solid #ccc;
    border-radius: 6px;
    padding: 0.5em;
}
```

MyST

````
```{mermaid}
sequenceDiagram
    participant User
    participant Snap as snapd (Client)
    participant Store as Snap Store
    participant System

    User->>Snap: snap install hello
    Snap->>Store: Request snap info and download
    Store-->>Snap: Return snap package
    Snap->>System: Install and mount snap
    System-->>Snap: Installation result
    Snap-->>User: Confirm snap is installed
```
````

rST

```
.. mermaid::

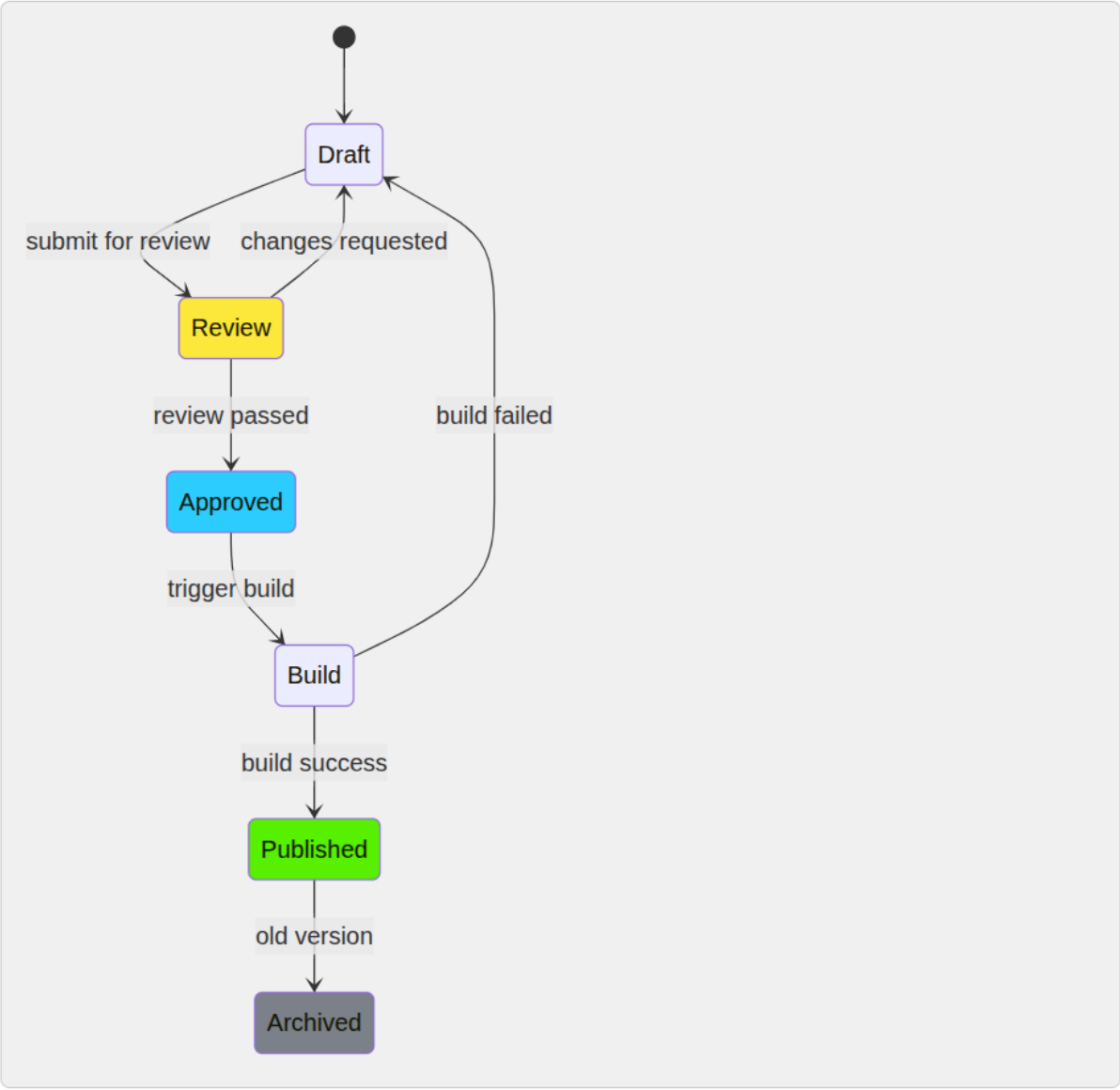
   sequenceDiagram
       participant User
       participant Snap as snapd (Client)
       participant Store as Snap Store
       participant System

       User->>Snap: snap install hello
       Snap->>Store: Request snap info and download
       Store-->>Snap: Return snap package
       Snap->>System: Install and mount snap
       System-->>Snap: Installation result
       Snap-->>User: Confirm snap is installed
```

State diagram with image-specific styles

The state diagram below has image-specific custom styling applied using the `classDef` keyword[97].

Diagram



[97] https://mermaid.js.org/syntax/stateDiagram.html#styling-with-classdefs

MyST

```
```{mermaid}
stateDiagram-v2
    classDef review fill:#FCE83A
    classDef approved  fill:#2DCCFF
    classDef published fill:#56F000
    classDef archived fill:#7B8089

    [*] --> Draft
    Draft --> Review : submit for review
    Review --> Draft : changes requested
    Review --> Approved : review passed
    Approved --> Build : trigger build
    Build --> Published : build success
    Build --> Draft : build failed
    Published --> Archived : old version

    class Review review
    class Approved approved
    class Published published
    class Archived archived
```
```

rST

```
.. mermaid::

    stateDiagram-v2
        classDef review fill:#FCE83A
        classDef approved  fill:#2DCCFF
        classDef published fill:#56F000
        classDef archived fill:#7B8089

        [*] --> Draft
        Draft --> Review : submit for review
        Review --> Draft : changes requested
        Review --> Approved : review passed
        Approved --> Build : trigger build
        Build --> Published : build success
        Build --> Draft : build failed
        Published --> Archived : old version

        class Review review
        class Approved approved
        class Published published
        class Archived archived
```

Projects using Mermaid

Here are some Canonical projects that use Mermaid for diagramming in their documentation:

- Checkbox[98]
- cloud-init[99]
- Charmed MySQL K8s[100]
- Ubuntu Pro[101]

Import docstrings with Sphinx `autodoc`

Module and function details are useful reference material to have in documentation, but the process of manually pulling all the necessary details over can become tedious. The Sphinx `autodoc` extension[102] provides the capability to automatically pull in docstrings and module information for Python code.

Prerequisites

To use the Sphinx `autodoc` extension with the Sphinx Stack, you need:

- Python module files located within the same repository as your documentation

OR

- The code repository added as a Git submodule[103] into the documentation repository

Setup

In the `conf.py` file in your docs directory, update the `sys.path` so that Sphinx can find your module files. At the top of the file, add a `sys.path.insert` that adds your `<code>` directory:

Listing 49: `conf.py`

```
import sys
from pathlib import Path

relative_code_path = Path('..', '<code>')
absolute_code_path = relative_code_path.resolve()
absolute_code_path_str = str(absolute_code_path)
sys.path.insert(0, absolute_code_path_str) # insert at index 0 so it occurs early in the
list
```

Then, further down in the `conf.py` file, add `sphinx.ext.autodoc` to the list of extensions:

[98] https://canonical-checkbox.readthedocs-hosted.com/stable/explanation/remote/
[99] https://docs.cloud-init.io/en/latest/explanation/boot.html
[100] https://canonical-charmed-mysql-k8s.readthedocs-hosted.com/8.0/explanation/flowcharts/
[101] https://documentation.ubuntu.com/pro/support-overview/
[102] https://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html
[103] https://git-scm.com/book/en/v2/Git-Tools-Submodules

Listing 50: `conf.py`

```
extensions = [
    ...
    "sphinx.ext.autodoc",
]
```

Usage

See Sphinx's `autodoc` instructions[104] for details.

Known issues and limitations

There are a few issues and limitations that should be taken into consideration.

Language

The extension's usage is limited to Python code. There are extensions for some other languages but they have not been tested with the Sphinx Stack, such as sphinxcontrib-rust[105] for Rust.

Docstring format

The `autodoc` extension pulls the docstrings straight into the the reStructuredText (reST) document, which requires the docstrings to be in reST format. For docstrings in the Numpy or Google style, the napoleon[106] extension can convert the docstrings into reST prior to processing by `autodoc`.

For documentation that is written in MyST Markdown, wrap the `eval-rst` directive around the `autodoc` calls:

````
```{eval-rst}

.. py:currentmodule:: code.<module-name>

.. automodule:: <module-name>
    :members:

```
````

[104] https://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html#usage
[105] https://sphinxcontrib-rust.readthedocs.io/en/stable/
[106] https://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html#module-sphinx.ext.napoleon

Canonical examples

Table 1: Canonical autodoc examples

Product	`conf.py`	Raw Doc	Rendered Doc
Jubilant	conf.py[107]	jubilant.rst[108]	jubilant reference page[109]
Ops	conf.py[110]	ops.rst[111]	*ops* reference page[112]

Add OpenAPI integration

OpenAPI specifications provide a standardized way to document REST APIs. This guide demonstrates how to add a minimal OpenAPI integration to your project using Swagger UI[113], including a basic specification and an opt-in tag toggle based on a build-time environment variable to keep the documentation clean by default.

Locate the specification

The stub OpenAPI document can be found under `docs/how-to/assets/openapi.yaml`. You can use it as a starting point for your API description:

```
openapi: 3.0.3
info:
  title: Hello API
  version: 1.0.0
paths:
  /hello:
    get:
      summary: Say hello
      parameters:
        - in: query
          name: name
          schema:
            type: string
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                type: object
```

(continues on next page)

[107] https://github.com/canonical/jubilant/blob/023ce73353352133c43dfb17b4a6cfad0f3e7816/docs/conf.py
[108] https://github.com/canonical/jubilant/blob/023ce73353352133c43dfb17b4a6cfad0f3e7816/docs/reference/jubilant.rst
[109] https://documentation.ubuntu.com/jubilant/reference/jubilant/
[110] https://github.com/canonical/operator/blob/main/docs/conf.py
[111] https://github.com/canonical/operator/blob/main/docs/reference/ops.rst
[112] https://documentation.ubuntu.com/ops/latest/reference/ops/
[113] https://swagger.io/tools/swagger-ui/

(continued from previous page)

```
              properties:
                message:
                  type: string
                  example: "Hello, world!"
```

Enable the viewer

The OpenAPI interactive viewer is disabled by default. Exporting the `OPENAPI` environment variable when building in your terminal makes Sphinx copy the specification to the output directory and add an `openapi` tag that we can use in the documentation source:

```
export OPENAPI=1
```

Then rebuild the docs to pick up the specification:

```
make clean html
```

Tip:

Unset the variable to go back to the default build:

```
unset OPENAPI
```

To enable it for a single command, prefix it like this:

```
OPENAPI=1 make clean html
```

Confirm the viewer works

Serve the documentation locally:

```
make run
```

Navigate to `/how-to/openapi/`. When the feature flag is enabled, the page renders the Swagger UI for the shipped specification right here. The snippet that does this should be included in your documentation source with the `.. raw:: html` directive:

```
   <link rel="stylesheet" type="text/css" href="https://unpkg.com/swagger-ui-dist@5.11.0/
swagger-ui.css" />
   <div id="openapi-hello-swagger"></div>
   <script src="https://unpkg.com/swagger-ui-dist@5.11.0/swagger-ui-bundle.js" charset=
"UTF-8"
       crossorigin="anonymous"></script>
   <script src="https://unpkg.com/swagger-ui-dist@5.11.0/swagger-ui-standalone-preset.js"
charset="UTF-8"
       crossorigin="anonymous"></script>
   <script>
       window.addEventListener('load', function () {
           const specUrl = new URL('/openapi.yaml', window.location.href).toString();
           SwaggerUIBundle({
```

(continues on next page)

(continued from previous page)

```
            url: specUrl,
            dom_id: '#openapi-hello-swagger',
            presets: [SwaggerUIBundle.presets.apis, SwaggerUIStandalonePreset],
            plugins: [],
            deepLinking: true,
            defaultModelsExpandDepth: -1,
            supportedSubmitMethods: []
        });
    });
  </script>
```

Warning:

The interactive viewer is hidden now because `OPENAPI` was not set when this documentation was built. Export `OPENAPI=1` when building the documentation to see it here.

Building on RTD

To enable the OpenAPI viewer on Read the Docs[114] tentatively, define the `OPENAPI` environment variable in your project's admin settings[115].

If you're certain that you want the OpenAPI viewer enabled for all builds, drop all conditional statements from `conf.py` and the reST files. In `conf.py`, replace the following lines:

```
html_extra_path = []

# ...
if os.getenv("OPENAPI", ""):
    tags.add("openapi")
    html_extra_path.append("how-to/assets/openapi.yaml")
```

With this:

```
html_extra_path = ["how-to/assets/openapi.yaml"]
```

Next, remove any tag-based logic from the reST files:

```
.. only:: openapi
```

[114] https://readthedocs.org/
[115] https://docs.readthedocs.com/platform/stable/guides/environment-variables.html

Troubleshooting

- The `make html` command fails with *unknown tag*: Make sure you exported the environment variable in the same shell session that runs `make`.
- The Swagger UI appears but cannot load the specification: Make sure that `openapi.yaml` exists in the *root* of your build output or adjust the import paths according to your build structure. Also, overriding the `html_extra_path` setting in `conf.py` may interfere with this configuration because it is used to copy the specification to the output directory.

Add interactive data tables

Data-heavy documentation often benefits from tables that users can explore interactively. The Sphinx DataTables[116] extension provides interactive tables where users can sort columns and filter rows, making large datasets more navigable and useful. For examples, see the extension's documentation.

The extension isn't available by default in the Sphinx Stack. To read more about table functionality that is available out of the box, see *Tables (MyST)* (page 121) and *Tables (reST)* (page 106).

Include the `sphinx-datatables` extension

To include the extension in your documentation, add `sphinx-datatables` to `docs/requirements.txt`.

Then, add `sphinx_datatables` to the `extensions` list in `docs/conf.py`.

Make a table interactive by adding a special CSS class to the directive:

reST

```
.. csv-table::
    :class: sphinx-datatable
    :file: /how-to/assets/animals.csv
    :header-rows: 1
```

MyST

````
```{csv-table}
:class: sphinx-datatable
:file: /how-to/assets/animals.csv
:header-rows: 1
```
````

[116] https://sharm294.github.io/sphinx-datatables/

Link to other documentation sets with Intersphinx

This guide provides instructions on how to set up external references using the Intersphinx extension. Linking to external Sphinx-based documentation sets via hyperlinks is difficult to maintain. Intersphinx provides a more robust alternative, linking to sections of other Sphinx projects based on labels mapped out in an inventory list as `objects.inv`. To effectively use Intersphinx, sections in the target documentation set must have labels.

Configure the extension

In the `conf.py` file in your docs directory, add or enable `sphinx.ext.intersphinx` under `extensions`:

Listing 51: conf.py

```
extensions = [
    ...
    "sphinx.ext.intersphinx",
    ]
```

Then, connect to the other project in the `intersphinx_mapping` setting:

Listing 52: conf.py

```
# Add configuration for intersphinx mapping
# Map only the Sphinx documentation sets that you need to link to from your docs set.
intersphinx_mapping = {
    'project-key': ('https://example.com', None)
}
```

Replace `project-key` with the internal identifier for the project that you'll specify in the document markup. For example, if the identifier were `chisel`, an Intersphinx link to it would be `:external+chisel:ref:`target``.

The value after the URI points to a custom path location. If a project stores its `objects.inv` at a special location, replace `None` with the path to it.

Check the external target labels

To check external target page labels, either search the project source code manually or inspect the `objects.inv` file. In this file, each section has a unique label.

To inspect the file manually, run:

```
source .venv/bin/activate
python -m sphinx.ext.intersphinx https://example.com/objects.inv
```

The output lists the different pages and their labels:



```
std:doc
    explanation/faq                          FAQ
: explanation/faq/
    explanation/index                        Explanation
: explanation/
    explanation/mode-of-operation            How Chisel works
: explanation/mode-of-operation/
std:label
    chise_faq                                FAQ
: explanation/faq/#chise-faq
    chisel-releases_ref                      chisel-releases
: reference/chisel-releases/#chisel-releases-ref
    chisel_helloworld_tutorial               Getting started with Chisel
: tutorial/getting-started/#chisel-helloworld-tutorial
```

The appropriate labels are under `std:label`.

Add references to the text

To add in-line references, follow this structure:

Format	Syntax
reST	`:external+project_key:ref:`an_external_target``
MyST	{ref}`project_key:an_external_target`

Customize the documentation check CI

The Sphinx Stack provides a GitHub Actions workflow, `.github/workflows/documentation-checks.yaml`, to automate spelling, link, and inclusive language checks. This guide describes how to configure the working directory and Python version, and how to add or remove documentation checks.

You can also *run these checks locally* (page 15).

Change the working directory

By default, the workflow uses `docs` as the documentation directory. If your documentation is located in a different directory, you must modify `documentation-checks.yaml` in two places:

- The workflow is configured to run on pull requests that modify files in the `docs` directory. Find the `"docs/**"` string under `on.pull_request.paths`, and replace `docs` with the path to your documentation. For example, you can change the documentation directory from `docs` to `doc`:

```
on:
  push:
```

(continues on next page)

(continued from previous page)

```
    branches:
      - main
  pull_request:
    paths:
-     - "docs/**"
+     - "doc/**"
```

- In the remainder of `documentation-checks.yaml`, the working directory is configured with the `DOCS_DIR` environment variable set under `env`. Change the value of `DOCS_DIR` from `"docs"` to the path to your documentation.

```
 env:
-  DOCS_DIR: "docs"
+  DOCS_DIR: "doc"
   PYTHON_VERSION: "3.10"
```

Note:

Do not include a trailing slash character when changing the value of the `DOCS_DIR` variable.

Modify the Python version

The Python version is configured with the `PYTHON_VERSION` environment variable set under `env`. Change this value to use a different version:

```
 env:
   DOCS_DIR: "docs"
   DOCS_DIR: "doc"
-  PYTHON_VERSION: "3.10"
+  PYTHON_VERSION: "3.12"
```

Note:

Place the Python version number in quotation marks. Version numbers without quotation marks will be parsed as floating-point numbers, and the GitHub Action will fail.

Add or remove a Makefile target

The `documentation-checks.yaml` workflow uses a matrix strategy to run three jobs in parallel: the spelling, links, and inclusive language checks. Each check in the matrix defines a `name` and a `target`. The `name` is how the check is listed when it runs on GitHub, and the `target` corresponds to a Make target defined in the Sphinx Stack `Makefile`. The job itself consists of a single command: `make <target>`.

To add a new `Makefile` target to the matrix, add a `name` and `target` pair to the array under `jobs.checks.strategy.matrix.check`. To remove a check, simply remove the lines containing the check's `name` and `target`. For example, you can remove the spelling check and add a style guide check:

```
jobs:
  checks:
    name: ${{ matrix.check.name }}
    runs-on: ubuntu-24.04
    strategy:
      fail-fast: false
      matrix:
        check:
-         - name: Spelling check
-           target: spelling
+         - name: Style guide check
+           target: vale
          - name: Link check
            target: linkcheck
          - name: Inclusive language check
            target: woke
```

3.3.3. Reference

Syntax guides, list of enabled sphinx extensions, and reference information for the Sphinx Stack GitHub workflows.

Also see the following information:

- Sphinx Stack repository[117]

Contents

GitHub workflows

The Sphinx Stack provides several GitHub Actions workflows to run checks on documentation projects.

Spelling, link, and inclusive language checks

The `documentation-checks.yaml` workflow runs several checks that correspond to targets in the Sphinx Stack `Makefile`:

- Spelling check (`spelling`)
- Link check (`linkcheck`)
- Inclusive language check (`woke`)

Refer to the how-to guides for details about how to *modify this workflow* (page 88) or *run documentation checks locally* (page 15).

[117] https://github.com/canonical/sphinx-stack

Default configuration

The documentation workflow is configured as follows:

Key	Description	Default
`working-directory`	The root of the documentation project.	`docs`
`python-version`	The Python interpreter to use for the workflow's jobs.	`3.10`
`fetch-depth`	The number of commits to fetch from your repository.	`0` (the full history is fetched)
`runs-on`	The host system for the workflow's runners.	`ubuntu-24.04`

Check for removed URLs

Added in version 1.2.0.

The Sphinx Stack includes a GitHub Actions workflow to identify when pages have been removed. This includes moving pages to another path, or removing them completely.

This does not cover higher-level changes to URL paths, such as changes to the project name or URL slug pattern on RTD.

This check ensures that redirects are implemented when pages are moved, or appropriate information is provided when anything is removed. It only runs on pull request builds.

Default Sphinx configuration

Sphinx projects are configured through a build configuration file named `conf.py`. The Sphinx Stack ships with the following configuration to support its default behavior and features:

Required configuration

Project, repository, and site metadata

The `Project Information` section contains settings for your project's official name, Open Graph Protocol (OGP) metadata, and global variables that are passed into the Sphinx context across your entire site.

Configuration setting	Description
`project`	Specifies the official name of your project.
`ogp_site_name`, `ogp_image`	Defines the preview website name and preview image.
`html_favicon`	Defines the small icon shown in the browser tab, bookmarks, and sometimes the browser history for your documentation.
`html_context`	Specifies a dictionary of custom values that Sphinx passes into the HTML template rendering context, including: • Product website links (`product_page`) • Community or contact links (`discourse`, `mattermost`, `matrix`) • Documentation source and issue integration (`github_url`, `repo_default_branch`, `repo_folder`, `github_issues`) • UI behavior (`sequential_nav`, `display_contributors`) • Footer and license metadata (`author`, `license`)
`html_theme_options`	Sets options that the HTML theme reads when rendering pages.

Sitemap

Configuration setting	Description
`html_baseurl`	Specifies the base URL of your documentation website.
`sitemap_url_scheme`	Determines how page URLs are formed. It is set to `'{link}'` by default. This uses the value of `html_baseurl` to generate the full URL for each page for the sitemap.
`sitemap_show_lastmod`	Specifies whether to include the last modification time in the sitemap.
`sitemap_excludes`	Lists the non-public pages to be excluded from the sitemap. Wildcards are supported. For details, see Excluding Pages[118].

To understand how the Sphinx Stack generates sitemaps, see *Sitemaps* (page 135). For instructions on supporting versioned documentation, see *Manage sitemaps for versioned documentation* (page 70).

[118] https://sphinx-sitemap.readthedocs.io/en/latest/advanced-configuration.html#excluding-pages

sphinx-llm extension

`llms_txt_description` specifies the short description included in the `llms.txt` file generated by the `sphinx_llm.txt` extension.

Sphinx link checker

Configuration settings in the Link checker exceptions section tell Sphinx which URLs to skip, which anchor checks to relax, and how persistent to be when a request is slow or fails.

Configuration setting	Description
`linkcheck_ignore`	Lists the URLs to ignore entirely. Use this configuration setting when the whole link is unreliable or intentionally excluded.
`linkcheck_anchors_ignore_for_url`	Lists anchor fragments to ignore for matching URLs. Use this configuration setting when the page is valid, but its section anchors are not worth verifying.
`linkcheck_timeout`	Specifies how long in seconds to wait for a response before timing out
`linkcheck_retries`	Specifies how many times to retry failures

Feedback button

Configuration setting	Description
`html_context["github_url"]`	Specifies the URL of the GitHub repository where feedback issues are created.
`disable_feedback_button`	Disables the feedback button when set to `True`.
`html_context["feedback_link"]`	Set this if your team doesn't use GitHub issues for feedback. This setting takes precedence over the link constructed with `github_url` if set.

Optional configuration

The Sphinx Stack contains several features that you can configure or turn off if they aren't suitable for your documentation.

HTML templates

The Sphinx Stack provides default header and footer templates. The `templates_path` setting specifies the directories that Sphinx searches for project-specific templates.

For information about how to create your own templates, see *Use custom HTML templates* (page 66).

Redirects

Configuration setting	Description
`rediraffe_redirects`	Specifies redirect mappings either as a dictionary or as the path, relative to `conf.py`, to a file containing the mappings.
`rediraffe_dir_only`	Controls whether redirect destination URLs omit `/index.html`. When set to `True`, destination URLs use directory-style paths.

For instructions on adding redirect mappings, see *Redirect pages* (page 45).

Extensions

Configuration setting	Description
`extensions`	Lists the Sphinx extensions enabled for the documentation project. Each extension must be installed in the documentation build environment.

For extensions included with the Sphinx Stack, see *Default Sphinx extensions* (page 95).

Extension support

The only extensions formally supported by the Sphinx Stack are those included in its default requirements.txt file[119].

[119] https://github.com/canonical/sphinx-stack/blob/main/docs/requirements.txt

UI behavior

The `sequential_nav` key in `html_context` controls which Previous and Next navigation buttons are displayed at the bottom of pages. Valid values are:

Value	Description
`both`	Both the Previous and Next buttons are shown at the bottom.
`none`	No Previous or Next button is shown at the bottom.
`prev`	Only the Previous button is shown at the bottom.
`next`	Only the Next button is shown at the bottom.

For page-specific navigation settings, see *Add page-specific configuration* (page 70).

Default Sphinx extensions

These extensions are enabled in the Sphinx Stack by default.

`ulwazi`
: Sphinx theme for Canonical documentation, based on the Vanilla framework[120].

`myst_parser`
: Adds support for the MyST markdown flavor.

`sphinx_design`
: Provides UI components like responsive grids, cards, and badges.

`sphinxcontrib.jquery`
: Ensures jQuery is loaded to maintain compatibility with older extensions and themes.

`sphinxext.opengraph`
: Generates OpenGraph metadata to create rich preview cards when links are shared on social media.

`sphinx_rediraffe`
: Handles URL redirects for moved or deleted pages. Redirects can be defined as a dictionary in the `conf.py` file or in a separate file.

`sphinx.ext.intersphinx`
: Generates automatic links to the documentation objects of other Sphinx projects.

`sphinx_config_options`
: Provides customized directives to easily document system or project configuration options.

`sphinx_llm.txt`
: Generates Markdown artifacts that improve readability for LLMs.

`sphinx_related_links`
: Generates and injects a list of related links into documentation pages.

[120] https://vanillaframework.io/

sphinx_roles
: Provides custom inline text roles for standardized formatting or linking across the project.

sphinx_sitemap
: Automatically generates a standard sitemap.xml file to improve indexing for search engine optimization (SEO).

sphinx_terminal
: Renders styled terminal sessions and command-line output within the documentation.

sphinx_youtube_links
: Provides a directive to embed YouTube videos into a page.

sphinxcontrib.cairosvgconverter
: Converts SVG images into PDF.

canonical_sphinx
: Required for the Sphinx Stack. Provides the Furo-based theme and custom templates.

  Removed in version 3.0: Replaced by the Ulwazi theme.

notfound.extension
: Allows creating custom 404 pages.

  Removed in version 3.0: Now set up automatically by the Ulwazi theme.

sphinx_contributor_listing
: Automatically generates a list of contributors to the project repository.

  Removed in version 3.0: No longer aligns with Canonical design practices.

sphinx_filtered_toctree
: Filters Table of Contents (`toctree`) entries based on specific tags or conditions.

  Removed in version 3.0: Was a holdover from a previously removed feature.

sphinx_last_updated_by_git
: Fetches and displays the accurate 'last updated' date for pages by reading Git commit metadata.

  Removed in version 3.0: Now set up automatically by the sphinx-sitemap extension.

sphinx_reredirects
: Handles URL redirects for moved or deleted pages.

  Removed in version 3.0: Replaced by the sphinx-reredirraffe extension.

sphinx_tabs.tabs
: Provides tabs.

  Removed in version 3.0: Replaced by the sphinx-design extension.

sphinx_ubuntu_images
: Generates lists of official Ubuntu disk images for specific release ranges and architectures.

  Removed in version 3.0: Removed due to its low adoption rate, but can be reinstalled in downstream projects that use its directive.

reStructuredText syntax

The Sphinx Stack supports reStructuredText[121] (reST).

See the following sections for syntax help and conventions.

Note:

This guide assumes that you are using the Sphinx Stack[122]. Some of the mentioned syntax requires the Sphinx extensions enabled in the Sphinx Stack.

[122] https://github.com/canonical/sphinx-stack

For general style conventions, see the Canonical Documentation Style Guide[123].

Headings

Input	Description
`Title` `=====`	Page title and H1 heading
`Heading` `-------`	H2 heading
`Heading` `~~~~~~~`	H3 heading
`Heading` `^^^^^^^`	H4 heading
`Heading` `.......`	H5 heading

Underlines must be at least as long as the title or heading.

[121] https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html
[123] https://docs.ubuntu.com/styleguide/en

Inline formatting

Input	Output
`:guilabel:`UI element``	*UI element*
``code``	`code`
`:file:`file path``	`file path`
`:command:`command``	**command**
`:kbd:`Key``	Key
`*Italic*`	*Italic*
`**Bold**`	**Bold**

Code blocks

To start a code block, explicitly start a code block with `.. code-block::`. The code block must be surrounded by empty lines.

Input	Output

`.. code-block::`
 	
`  # Demonstrate a code block`
`  code:`
`  - example: true`
	`# Demonstrate a code block`
	`code:`
	`- example: true`

`.. code-block:: yaml`
 	
`  # Demonstrate a code block`
`  code:`
`  - example: true`
	`# Demonstrate a code block`
	`code:`
	`- example: true`

Terminal output

A terminal view emulates the command line experience more accurately than a code block. This is particularly useful in tutorials or guides that are terminal-heavy, or where it's helpful to show the directory a command is run from.

To include a terminal view, use the following directive:

Input	Output

`.. terminal::`
 	
`    input line 1`
`    input line 2`
 	
`    output line 1`
`    output line 2`
 	
`    output line 3`
	`user@host:~$ input line 1 > input line 2`
	
	`output line 1`
	`output line 2`
	
	`output line 3`

By default, everything before the first blank line in the directive's content is rendered as input, and any content that follows is rendered as output. The terminal directive can only display one input command, but that command can span multiple lines, as in the previous example.

To render only the output of a command, include the `:output-only:` flag in the directive's options:

Input	Output

`.. terminal::`
`    :output-only:`
 	
`    This is rendered as output.`
	`This is rendered as output.`

To customize the prompt (`user@host:~$` by default), specify any of the following options:

- `:user:`
- `:host:`
- `:dir:`

Input	Output

`.. terminal::`
`  :user: author`
`  :host: canonical`
`  :dir: ~/path`
 	
`  input`
 	
`  output`
	`author@canonical:~/path$ input`
	`output`

The copy button for input commands is **opt-in**. You must include the `:copy:` flag in the directive's options for the button to be displayed.

To make the terminal scroll horizontally instead of wrapping long lines, include the `:scroll:` option.

For more details, refer to the sphinx-terminal README[124].

Links

Link markup depends on whether you need an external URL or a page in the same documentation set.

External links

To link to documents in other Sphinx projects, use *Intersphinx* (page 87) with the `:ref:` or `:doc:` role:

Input	Output
`:external+ubuntu-desktop:doc:`index``	Ubuntu Desktop documentation[125]
`:external+ubuntu-desktop:ref:`install-ubuntu-desktop``	Install Ubuntu Desktop[126]

To link to other websites, use the hyperlink reference syntax:

[124] https://github.com/canonical/sphinx-terminal/blob/main/README.md
[125] https://ubuntu.com/desktop/docs/en/latest/
[126] https://ubuntu.com/desktop/docs/en/latest/tutorial/install-ubuntu-desktop/#install-ubuntu-desktop

Input	Output
`` `Canonical home <https://canonical.com>`__ ``	Canonical home[127]

If necessary, it's possible to write a standalone hyperlink, which won't contain any link text:

Input	Output
`https://canonical.com`	https://canonical.com

The documentation checks will likely flag it as a spelling error.

Plain text

Outside of directives, reST interprets every URL it finds as a hyperlink. If you need a link to be rendered as plain text, escape the colon in the protocol:

Input	Output
https\://canonical.com/	

Sidebar links

You can add links to related websites or Discourse topics to the sidebar.

To add a link to a related website, add the following field at the top of the page:

```
:relatedlinks: https://github.com/canonical/lxd-sphinx-extensions, [RTFM](https://www.google.com)
```

To override the title, use Markdown syntax. Note that spaces are ignored; if you need spaces in the title, replace them with ` `, and include the value in quotes if Sphinx complains about the metadata value because it starts with `[`. For example: `[My Title](https://...)`.

To add a link to a Discourse topic, configure the Discourse instance in the *conf.py* file. Then add the following field at the top of the page:

```
:discourse: <topic-id>
```

[127] https://canonical.com

Manual pages

When mentioning command-line utilities, you may wish to link to the corresponding manual page for the command. Ensure that the `manpages_url` setting in your `conf.py` is set appropriately and use the `:manpage:` role within your text to create a link.

For example, to link to man pages from the 24.04 LTS (Noble Numbat) release, include the following in your `conf.py`:

```
manpages_url = "https://manpages.ubuntu.com/manpages/noble/en/man{section}/{page}.
{section}.html"
```

Then within the document, use the following reST:

```
You can use the :manpage:`dd(1)` utility to write the disk image to your
SD card. If the image is compressed, use :manpage:`aunpack(1)` to extract
it first.
```

YouTube

To add a link to a YouTube video, use the following directive:

Input	Output
`.. youtube:: https://www.youtube.com/watch?v=iMLiK1fX4I0` `   :title: Demo`	

The video title is extracted automatically and displayed when hovering over the link. To override the title, add the `:title:` option.

Internal references

Sections

To reference a section within the documentation (either on the same page or on another page), add a label to that section and reference that label.

You can add a label anywhere in any document. When referencing a label that isn't attached to a heading, you must add link text. If you don't, the reference won't work.

Input	Output	Description
`.. _label_ID:`		Adds the label `label_ID`. **Note:** When defining the label, you must prefix it with an underscore. Do not use the starting underscore when referencing the label.
`:ref:`a_section_label``	*Sections* (page 102)	References a label that has a title.
`:ref:`Provided link text <a_random_label>``	*Provided link text* (page 102)	References a label and specifies a title.

Pages

If a documentation page does not have a label, you can still reference it by using the `:doc:` role with the file name and path.

Input	Output
`:doc:`index``	*Reference* (page 90)
`:doc:`Provided link text <index>``	*Provided link text* (page 90)

Only use the `:doc:` role when you cannot use the `:ref:` role, thus only if there is no label at the top of the file and you cannot add it. When using the `:doc:` role, your reference will break when a file is renamed or moved.

Formatted link text

With the exception of inline code, reST doesn't support special formatting for link text, such as *emphasized* and **strong** text.

Use the `:literalref:` role to format a reference's link text as inline code:

Input	Output
`:literalref:`example <https://example.com>``	example[128]
`:literalref:`label <a_random_label>``	*label* (page 102)

The link text is automatically excluded from the spelling check.

Navigation

Every documentation page must be included as a sub-page to another page in the navigation.

This is achieved with the toctree[129] directive in the parent page:

```
.. toctree::
   :hidden:

   sub-page1
   sub-page2
```

If a page should not be included in the navigation, you can suppress the resulting build warning by putting `:orphan:` at the top of the file. Use orphan pages sparingly and only if there is a clear reason for it.

Tip:

Instead of hiding pages that you do not want to include in the documentation from the navigation, you can exclude them from being built. This method will also prevent them from being found through the search.
To exclude pages from the build, add them to the `exclude_patterns` variable in the `conf.py` file.

[128] https://example.com
[129] https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-toctree

Lists

Input	Output

`- Item 1`
`- Item 2`
`- Item 3`
	• Item 1
	• Item 2
	• Item 3

`1. Step 1`
`#. Step 2`
`#. Step 3`
	1. Step 1
	2. Step 2
	3. Step 3

`a. Step 1`
`#. Step 2`
`#. Step 3`
	a. Step 1
	b. Step 2
	c. Step 3

You can also nest lists:

Input

```
1. Step 1

  - Item 1

    * Sub-item
  - Item 2

    i. Sub-step 1
    #. Sub-step 2

#. Step 2

  a. Sub-step 1

    - Item

  #. Sub-step 2
```

Output

1. Step 1
   - Item 1
     - Sub-item
   - Item 2
     i. Sub-step 1
     ii. Sub-step 2
2. Step 2

   a. Sub-step 1
   - Item
   1. Sub-step 2

In numbered lists, number the first item and use `#.` for all subsequent items to generate the step numbers automatically.

Use `-` for unordered lists. When using nested lists, you can use `*` for the nested level.

Definition lists

Input	Output

`Term 1:`
`  Definition`
`Term 2:`
`  Definition`
	Term 1:
	Definition
	Term 2:
	Definition

Tables

reST supports different markup for tables. Grid tables are most similar to tables in Markdown, but list tables are usually much easier to use. See the *Sphinx documentation* (page 106) for all table syntax alternatives.

Both markups result in the following output:

Header 1	Header 2
Cell 1 Second paragraph cell 1	Cell 2
Cell 3	Cell 4

Grid tables

See grid tables[130] for reference.

```
+----------------------+------------+
| Header 1             | Header 2   |
+======================+============+
| Cell 1               | Cell 2     |
|                      |            |
| 2nd paragraph cell 1 |            |
+----------------------+------------+
| Cell 3               | Cell 4     |
+----------------------+------------+
```

List tables

See list tables[131] for reference.

```
.. list-table::
    :header-rows: 1

    * - Header 1
      - Header 2
    * - Cell 1

        2nd paragraph cell 1
      - Cell 2
    * - Cell 3
      - Cell 4
```

Data tables

If you have a small amount of CSV data, you can include the data in the doc source.

For example:

```
.. csv-table::
    :header: "Animal", "Number of legs", "Size"

    "Worm", 0, "Small"
```

(continues on next page)

[130] https://docutils.sourceforge.io/docs/ref/rst/restructuredtext.html#grid-tables
[131] https://docutils.sourceforge.io/docs/ref/rst/directives.html#list-table

(continued from previous page)

```
    "Penguin", 2, "Medium"
    "Horse", 4, "Large"
    "Ant", 6, "Small"
    "Octopus", 8, "Medium"
```

If you have a large amount of CSV data, or the data is generated by an automated process, you can include the data from a file.

For example:

```
.. csv-table::
    :file: /assets/animals.csv
    :header-rows: 1
```

Both markups result in the following output:

Animal	Number of legs	Size
Worm	0	Small
Penguin	2	Medium
Horse	4	Large
Ant	6	Small
Octopus	8	Medium

Customize the column widths, character encoding, and so on, as described in the csv-table reference[132].

The Sphinx Stack can also render interactive tables, which are described in *Add interactive data tables* (page 86).

Notes

```
.. admonition:: <title>
    :class: <class>
```

[132] https://docutils.sourceforge.io/docs/ref/rst/directives.html#csv-table

Images

Input	Output
`.. image:: https://assets.ubuntu.com/v1/b3b72cb2-canonical-logo-166.png`	

`.. figure:: https://assets.ubuntu.com/v1/b3b72cb2-canonical-logo-166.png`
`  :width: 100px`
`  :alt: Alt text`
 	
`  Figure caption`
	
	Fig. 1: Figure caption

For local pictures, start the path with / (for example, `/images/image.png`).

Use `PNG` format for screenshots and `SVG` format for graphics.

If producing multiple output formats, use `*` as the file extension to have Sphinx select the best image format for the output

See Five golden rules for compliant alt text[133] for information about how to word the alt text.

Reuse

A big advantage of reST in comparison to plain Markdown is that it allows the reuse of content.

[133] https://abilitynet.org.uk/resources/digital-accessibility/five-golden-rules-compliant-alt-text

Substitution

To reuse sentences and entire paragraphs that have little markup or special formatting, define substitutions[134] for them by putting the same directives in any reST file:

Listing 53:

```
.. |version_number| replace:: 0.1.0

.. |rest_text| replace:: *Multi-line* text
                         that uses basic **markup**.

.. And so on
```

Note:

Mind that substitutions can't be redefined; for instance, accidentally including a definition twice causes an error:

```
ERROR: Duplicate substitution definition name: "rest_text".
```

The definitions from the above examples are rendered as follows:

Input	Output
`|version_number|`	0.1.0
`|rest_text|`	*Multi-line* text that uses basic **markup**.
`|site_link|_`	Website link[135]

File inclusion

To reuse longer sections or text with more advanced markup, you can put the content in a separate file and include the file or parts of the file in several locations.

To select parts of the text in a file, use `:start-after:` and `:end-before:` if possible. You can combine those with `:start-line:` and `:end-line:` if the same text occurs more than once. Using only `:start-line:` and `:end-line:` is error-prone though.

You cannot put any labels into the content that is being reused (because references to this label would be ambiguous then). You can, however, put a label right before including the file.

By combining file inclusion and substitutions defined directly in a file, you can even replace parts of the included text.

[134] https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html#substitutions
[135] https://example.com

Input	Output

`.. include:: /how-to/index.rst`
`  :start-after: .. _how-to-guides:`
`  :end-before: ==========`
	How-to guides

Files that only contain text that is reused somewhere else should be placed in the `reuse` directory and end with the extension `.txt` to distinguish them from normal content files.

To make sure inclusions don't break, consider adding comments (`.. some comment`) to the source file as markers for starting and ending.

Tabs

The recommended way of creating tabs is with the Sphinx design[136] extension.

Input	Output

`.. tab-set::`
 	
`  .. tab-item:: Tab 1`
`  :sync: key1`
 	
`  Content Tab 1`
 	
`  .. tab-item:: Tab 2`
`  :sync: key2`
 	
`  Content Tab 2`
	Tab 1
	Content Tab 1
	Tab 2
	Content Tab 2

[136] https://sphinx-design.readthedocs.io/en/latest/

Glossary

You can define glossary terms in any file. Ideally, all terms should be collected in one glossary so they can then be referenced from any file.

Input	Output

`.. glossary::`
 	
`  an example term`
`  Definition of an example term.`
	an example term
	Definition of an example term.

`:term:`an example term``	*an example term*

More useful markup

Input	Output	Description
`.. versionadded:: X.Y`	Added in version X.Y.	Can be used to distinguish between different versions.

`| Line 1`
`| Line 2`
`| Line 3`
	Line 1
	Line 2
	Line 3
		Line breaks that are not paragraphs. Use this sparingly.

`----`	A horizontal line	Can be used to visually divide sections on a page.
`.. This is a comment`		Not visible in the output.
`:abbr:`API (Application Programming Interface)``	API (Application Programming Interface)	Hover to display the full term.
`:spellexception:`PurposelyWrong``		Explicitly exempt a term from the spelling check.
`.. meta::` `  :description: Your description of the page.`	Not displayed	Set a custom HTML meta description with this directive at the top of the document. Don't put newlines in the description.

MyST syntax

The Sphinx Stack supports MyST Markdown[137].

See the following sections for syntax help and conventions.

Note:

This guide assumes that you are using the Sphinx Stack[138]. Some of the mentioned syntax requires the Sphinx extensions enabled in the Sphinx Stack.

[138] https://github.com/canonical/sphinx-stack

For general style conventions, see the Canonical Documentation Style Guide[139].

Headings

Input	Description
`# Title`	Page title and H1 heading
`## Heading`	H2 heading
`### Heading`	H3 heading
`#### Heading`	H4 heading
…	Further headings

Nesting

In MyST, triple backticks (```) wrap both code blocks and directives, which can cause collisions if you need to place one element inside another. To nest a code block or directive inside another, add an extra backtick to the outer element's fences:

Input	Output

``{admonition} Nested code block
``python
import pathlib
``
``
	Nested code block
	`import pathlib`

[137] https://myst-parser.readthedocs.io
[139] https://docs.ubuntu.com/styleguide/en

Additional levels of nesting require additional backticks for each parent block.

Inline formatting

Input	Output
`{guilabel}`UI element``	*UI element*
`` `code` ``	code
{file}`file path`	file path
{command}`command`	**command**
{kbd}`Key`	Key
Italic	*Italic*
Bold	**Bold**

Code blocks

Start and end a code block with three back ticks:

Input	Output
``` # Demonstrate a code block code: - example: true ```	# Demonstrate a code block code: - example: true

``yaml
# Demonstrate a code block 	
code:
- example: true
``
	# Demonstrate a code block
	code:
	- example: true

To include back ticks in a code block, increase the number of surrounding back ticks:

Input	Output

``
``
``
	``

Terminal output

A terminal view emulates the command line experience more accurately than a code block. This is particularly useful in tutorials or guides that are terminal-heavy, or where it's helpful to show the directory a command is run from.

To show a terminal view, use the following directive:

Input	Output

``{terminal}
 	
input line 1
input line 2
 	
output line 1
output line 2
 	
output line 3
``
	user@host:~$ input line 1 > input line 2
	output line 1
	output line 2
	
	output line 3

By default, everything before the first blank line in the directive's content is rendered as input, and any content that follows is rendered as output. The terminal directive can only display one input command, but that command can span multiple lines, as in the previous example.

To render only the output of a command, include the `:output-only:` flag as a directive option:

Input	Output

``{terminal}
:output-only:
 	
This is rendered as output.
``
	This is rendered as output.

To customize the prompt (`user@host:~$` by default), specify any of the following options:

- `:user:`
- `:host:`
- `:dir:`

Input	Output

```` ```{terminal}
:user: author
:host: canonical
:dir: ~/path
 	
input
 	
output
``` ````
	`author@canonical:~/path$ input`
	`output`

The copy button for input commands is **opt-in**. You must include the `:copy:` flag in the directive's options for the button to be displayed.

To make the terminal scroll horizontally instead of wrapping long lines, include the `:scroll:` option.

For more details, refer to the `sphinx-terminal` README[140].

Links

Link markup depends on whether you need an external URL or a page in the same documentation set.

External links

To link to documents in other Sphinx projects, use *Intersphinx* (page 87) with the `{ref}` or `{doc}` role:

Input	Output
`{external+ubuntu-desktop:doc}`index``	Ubuntu Desktop documentation[141]
`{external+ubuntu-desktop:ref}`install-ubuntu-desktop``	Install Ubuntu Desktop[142]

[140] https://github.com/canonical/sphinx-terminal/blob/main/README.md
[141] https://ubuntu.com/desktop/docs/en/latest/
[142] https://ubuntu.com/desktop/docs/en/latest/tutorial/install-ubuntu-desktop/#install-ubuntu-desktop

For external links, use Markdown syntax. You can also use just the URL, but this will usually cause issues with the spelling check, so you should specify the link text as code in this case.

Input	Output
`[Canonical website](https://canonical.com)`	Canonical website[143]
`https://canonical.com`	[144]
`` [`https://canonical.com`](https://canonical.com) ``	`https://canonical.com`

Plain text

If you need a link rendered as plain text, escape the colon in the protocol:

Input	Output
https\://canonical.com/	

Sidebar links

You can add links to related websites or Discourse topics to the sidebar.

To add a link to a related website, add the following to the page's Markdown front matter:

```
---
relatedlinks: https://github.com/canonical/lxd-sphinx-extensions, [RTFM](https://www.google.com)
---
```

If you override the title, note that spaces are ignored; if you need spaces in the title, replace them with ` `, and include the value in quotes if Sphinx complains about the metadata value because it starts with `[`. For example: `[My Title](https://...)`.

To add a link to a Discourse topic, configure the Discourse instance in the :file:`conf.py` file. Then add the following field to the page's Markdown front matter:

```
---
discourse: <topic-id>
---
```

[143] https://canonical.com
[144] https://canonical.com

Manual pages

When mentioning command line utilities, you may wish to link to the corresponding manual page for the command. Ensure that the `manpages_url` setting in your `conf.py` is set appropriately and use the `{manpage}` role within your text to create a link.

For example, to link to man pages from the 24.04 LTS (Noble Numbat) release, include the following in your `conf.py`:

```
    manpages_url = "https://manpages.ubuntu.com/manpages/noble/en/man{section}/{page}.
{section}.html"
```

Then within the document:

```
You can use the {manpage}`dd(1)` utility to write the disk image to your
SD card. If the image is compressed, use {manpage}`aunpack(1)` to extract
it first.
```

YouTube

To add a link to a YouTube video, use the following directive:

Input	Output
```` ```{youtube} https://www.youtube.com/watch?v=iMLiK1fX4I0 :title: Demo ``` ````	

The video title is extracted automatically and displayed when hovering over the link. To override the title, add the `{title}` option.

Internal references

Sections

To reference a section within the documentation (either on the same page or on another page), add a label to that section and reference that label.

You can add a label anywhere in any document. When referencing a label that isn't attached to a heading, you must add link text. If you don't, the reference won't work.

Input	Output	Description
`(label_ID)=`		Adds the label `label_ID`.
`{ref}`a_section_label_myst``	*Sections* (page 118)	References a label that has a title.
`{ref}`link text <a_random_label_myst>``	*link text* (page 119)	References a label and specifies a title.
`[`xyz`](a_random_label_myst)`	*xyz* (page 119)	Use Markdown syntax if you need markup on the link text.

Pages

If a documentation page does not have a label, you can still reference it by using the `{doc}` role with the file name and path.

Input	Output
`{doc}`index``	*Reference* (page 90)
`{doc}`Provided link text <index>``	*Provided link text* (page 90)

Only use the `{doc}` role when you cannot use the `{ref}` role, thus only if there is no label at the top of the file and you cannot add it. When using the `{doc}` role, your reference will break when a file is renamed or moved.

Navigation

Every documentation page must be included as a sub-page to another page in the navigation.

This is achieved with the `toctree`[145] directive in the parent page:

```
```{toctree}
:hidden:

sub-page1
sub-page2
```
```

[145] https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-toctree

If a page should not be included in the navigation, you can suppress the resulting build warning by putting the following instruction at the top of the file:

```
---
orphan: true
---
```

Use orphan pages sparingly and only if there is a clear reason for it.

Tip:

Instead of hiding pages that you do not want to include in the documentation from the navigation, you can exclude them from being built. This method will also prevent them from being found through the search.
To exclude pages from the build, add them to the `exclude_patterns` variable in the `conf.py` file.

Lists

Input	Output

`- Item 1`
`- Item 2`
`- Item 3`
	• Item 1
	• Item 2
	• Item 3

`1. Step 1`
`1. Step 2`
`1. Step 3`
	1. Step 1
	2. Step 2
	3. Step 3

`1. Step 1`
`  - Item 1`
`    * Sub-item`
`  - Item 2`
`1. Step 2`
`  1. Sub-step 1`
`  1. Sub-step 2`
	1. Step 1
	  • Item 1
	  – Sub-item
	  • Item 2
	2. Step 2
	  1. Sub-step 1
	  2. Sub-step 2

In numbered lists, use `1.` for all items to generate the step numbers automatically. You can also use a higher number for the first item to start with that number.

Use `-` for unordered lists. When using nested lists, you can use `*` for the nested level.

Definition lists

Input	Output

`Term 1`
`: Definition`
 	
`Term 2`
`: Definition`
	Term 1
	Definition
	Term 2
	Definition

Tables

You can use standard Markdown tables. However, using the reST list table[146] syntax is usually much easier. See the Sphinx documentation[147] for all table syntax alternatives.

Both markups result in the following output:

Header 1	Header 2
Cell 1 Second paragraph cell 1	Cell 2
Cell 3	Cell 4

Markdown tables

```
| Header 1                           | Header 2 |
|------------------------------------|----------|
| Cell 1<br><br>2nd paragraph cell 1 | Cell 2   |
| Cell 3                             | Cell 4   |
```

List tables

See list tables[148] for reference.

```
```{list-table}
   :header-rows: 1

* - Header 1
  - Header 2
* - Cell 1
```

(continues on next page)

[146] https://docutils.sourceforge.io/docs/ref/rst/directives.html#list-table
[147] https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#table-directives
[148] https://docutils.sourceforge.io/docs/ref/rst/directives.html#list-table
```

(continued from previous page)

```
    2nd paragraph cell 1
  - Cell 2
* - Cell 3
  - Cell 4
```

Data tables

If you have a small amount of CSV data, you can include the data in the doc source.

For example:

```
```{csv-table}
:header-rows: 1

"Animal", "Number of legs", "Size"

"Worm", 0, "Small"
"Penguin", 2, "Medium"
"Horse", 4, "Large"
"Ant", 6, "Small"
"Octopus", 8, "Medium"
```
```

If you have a large amount of CSV data, or the data is generated by an automated process, you can include the data from a file.

For example:

```
```{csv-table}
:file: /reuse/animals.csv
:header-rows: 1
```
```

Both markups result in the following output:

Animal	Number of legs	Size
Worm	0	Small
Penguin	2	Medium
Horse	4	Large
Ant	6	Small
Octopus	8	Medium

Customize the column widths, character encoding, and so on, as described in the `csv-table` reference[149].

[149] https://mystmd.org/guide/directives#directive-csv-table

The Sphinx Stack can also render interactive tables, which are described in *Add interactive data tables* (page 86).

Notes

```
```{admonition} <title>
:class: <class>
```
```

Images

Input	Output
``` ![Alt text](https://assets.ubuntu.com/v1/ b3b72cb2-canonical-logo-166.png) ```	

``
``{figure} https://assets.ubuntu.com/v1/
b3b72cb2-canonical-logo-166.png
:width: 100px
:alt: Alt text
 	
Figure caption
``
``
	
	Fig. 2: Figure caption

For local pictures, start the path with / (for example, `/images/image.png`).

Use `PNG` format for screenshots and `SVG` format for graphics.

See Five golden rules for compliant alt text[150] for information about how to word the alt text.

[150] https://abilitynet.org.uk/resources/digital-accessibility/five-golden-rules-compliant-alt-text

Reuse

A big advantage of MyST in comparison to plain Markdown is that it allows the reuse of content.

Substitution

To reuse sentences or paragraphs that have little markup and special formatting, use substitutions[151].

Substitutions can be defined in the following locations:

Globally, in a file named `reuse/substitutions.yaml` that is loaded into the `myst_substitutions`[152] variable in `conf.py`. Or if you have a limited amount of substitutions, enter them directly into the `myst_substitutions` variable in `conf.py`:

Listing 54:

```
import os
import yaml

if os.path.exists('./reuse/substitutions.yaml'):
    with open('./reuse/substitutions.yaml', 'r') as fd:
        myst_substitutions = yaml.safe_load(fd.read())
else:
    myst_substitutions = {
        "version_number": "0.1.0",
        "formatted_text": "*Multi-line* text\n that uses basic **markup**.",
        "site_link": "[Website link](https://example.com)"
  }
```

Listing 55:

```
# Key/value substitutions to use within the Sphinx doc.
{version_number: "0.1.0",
  formatted_text: "*Multi-line* text\n that uses basic **markup**.",
  site_link: "[Website link](https://example.com)"}
```

Locally, putting the definitions at the top of a single file in the following format:

```
---
myst:
  substitutions:
    version_number: "0.1.0"
    formatted_text: "*Multi-line* text
                     that uses basic **markup**."
    advanced_reuse_key: "This is a substitution that includes a code block:
                         ```
                         code block
                         ```"
---
```

[151] https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html#substitutions
[152] https://myst-parser.readthedocs.io/en/v0.13.5/using/syntax-optional.html#substitutions-with-jinja2

You can combine both options by defining a default substitution in `reuse/substitutions.py` and overriding it at the top of a file.

The definitions from the above examples are rendered as follows:

Input	Output
`{{version_number}}`	0.1.0
`{{formatted_text}}`	*Multi-line* text that uses basic **markup**.
`{{site_link}}`	Website link[153]
`{{advanced_reuse_key}}`	This is a substitution that includes a code block: `code block`

Content isn't substituted on GitHub, so use substitution names that indicate the included content (for example, `note_not_supported` instead of `reuse_note`).

File inclusion

To reuse longer sections or text with more advanced markup, you can put the content in a separate file and include the file or parts of the file in several locations.

To select parts of the text in a file, use `:start-after:` and `:end-before:` if possible. You can combine those with `:start-line:` and `:end-line:` if the same text occurs more than once. Using only `:start-line:` and `:end-line:` is error-prone though.

You cannot put any labels into the content that is being reused (because references to this label would be ambiguous then). You can, however, put a label right before including the file.

By combining file inclusion and substitutions, you can even replace parts of the included text.

Input	Output

```` ```{include} /how-to/index.rst
   :start-after: .. _how-to-guides:
   :end-before: ==========
``` ````
	How-to guides

File inclusion does not work on GitHub, so you should always add a comment linking to the included file.

Files that only contain text that is reused somewhere else should be placed in the `reuse` directory and end with the extension `.txt` to distinguish them from normal content files.

To make sure inclusions don't break, consider adding HTML comments (`<!-- some comment -->`) to the source file as markers for starting and ending.

[153] https://example.com

Tabs

The recommended way of creating tabs is with the Sphinx design[154] extension.

Input	Output

``{tab-set}
 	
``{tab-item} Tab 1
:sync: key1
 	
Content Tab 1
``
 	
``{tab-item} Tab 2
:sync: key2
 	
Content Tab 2
``
 	
``
	Tab 1
	Content Tab 1
	Tab 2
	Content Tab 2

Collapsible sections

There is no support for details sections in MyST, but you can insert HTML to create them.

Input	Output
<details> <summary>Details</summary> Content </details>	Content

[154] https://sphinx-design.readthedocs.io/en/latest/

Glossary

You can define glossary terms in any file. Ideally, all terms should be collected in one glossary so they can then be referenced from any file.

Input	Output

```` ```{glossary}
 	
MyST example term
  Definition of the example term.
``` ````
	MyST example term
	Definition of the example term.

`{term}`MyST example term``	*MyST example term*

More useful markup

Input	Output	Description
```` ```{versionadded} X.Y ``` ````	Added in version X.Y.	Can be used to distinguish between different versions.
`---`	A horizontal line	Can be used to visually divide sections on a page.
`<!-- This is a comment -->`		Not visible in the output.
`{abbr}`API (Application Programming Interface)`	API	Hover to display the full term.
`{spellexception} `PurposelyWrong`		Explicitly exempt a term from the spelling check.
```` --- myst: html_meta: description: Your description of the page. --- ````	Not displayed	Set a custom HTML meta description in the MyST Markdown front matter at the top of the document. Don't put newlines in the description.

3.3.4. Explanation

Explore topics about the concepts and ideas in the Sphinx Stack.

The Sphinx Stack is built using standard Python tools, and is both deep and flexible.

Sphinx Stack structure

The Sphinx Stack is a template Sphinx[155] project. It provides a default file structure, a theme, and dependencies for Canonical documentation.

[155] https://www.sphinx-doc.org/en/master/

Sphinx

Sphinx is a documentation static site generator that converts reStructuredText or Markdown files into HTML. It's the core software in the Sphinx Stack.

The `docs/conf.py` file is a configuration file that defines the properties of the Sphinx project such as project metadata and extensions.

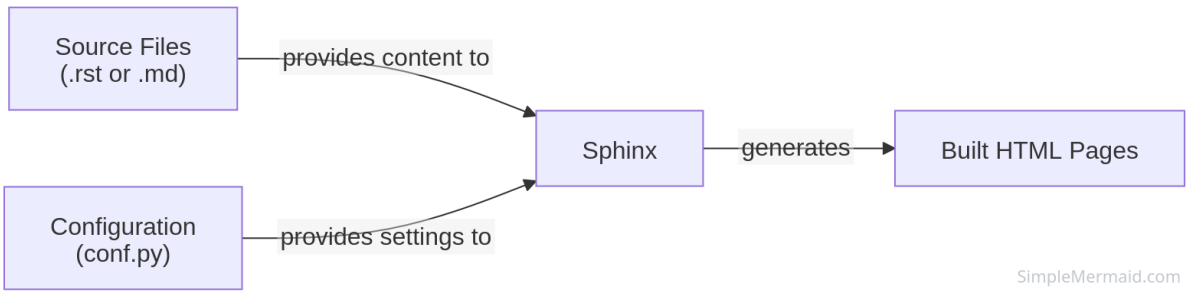


Fig. 3: Sphinx as a documentation static site generator

Python

Because Sphinx is a Python application, the Sphinx Stack depends on Python and a Python package manager. Most of its dependencies are Python packages. Local builds of the Sphinx Stack require a Python virtual environment to isolate the project from the host system.

To be able to work on a Sphinx Stack project, your host needs Python 3.11, pip, and venv.

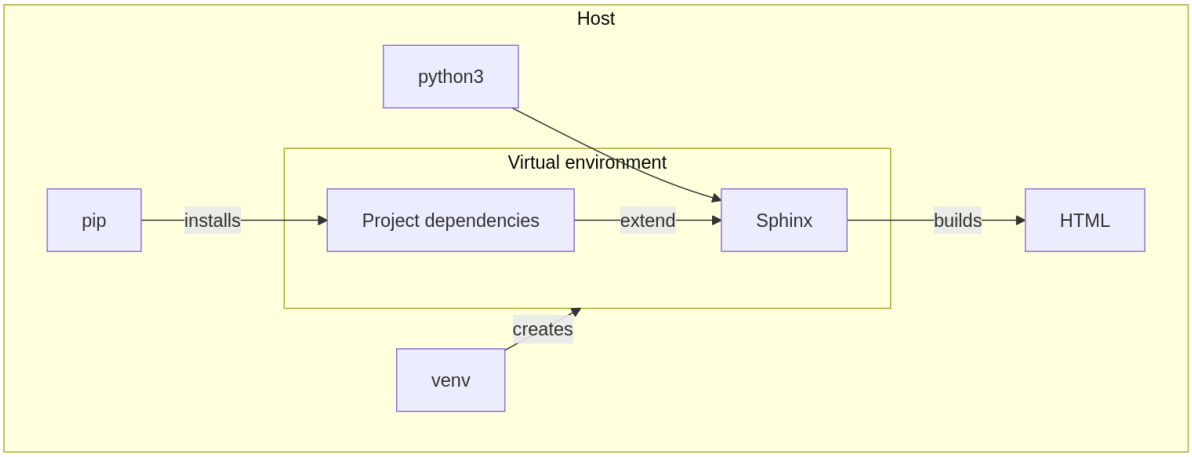


Fig. 4: Python's role in the Sphinx Stack

Sphinx extensions

The syntax and behavior of Sphinx can be modified with extensions. These can be used to create diagrams, test code, and more.

The Sphinx Stack includes a curated and tested set of extensions.

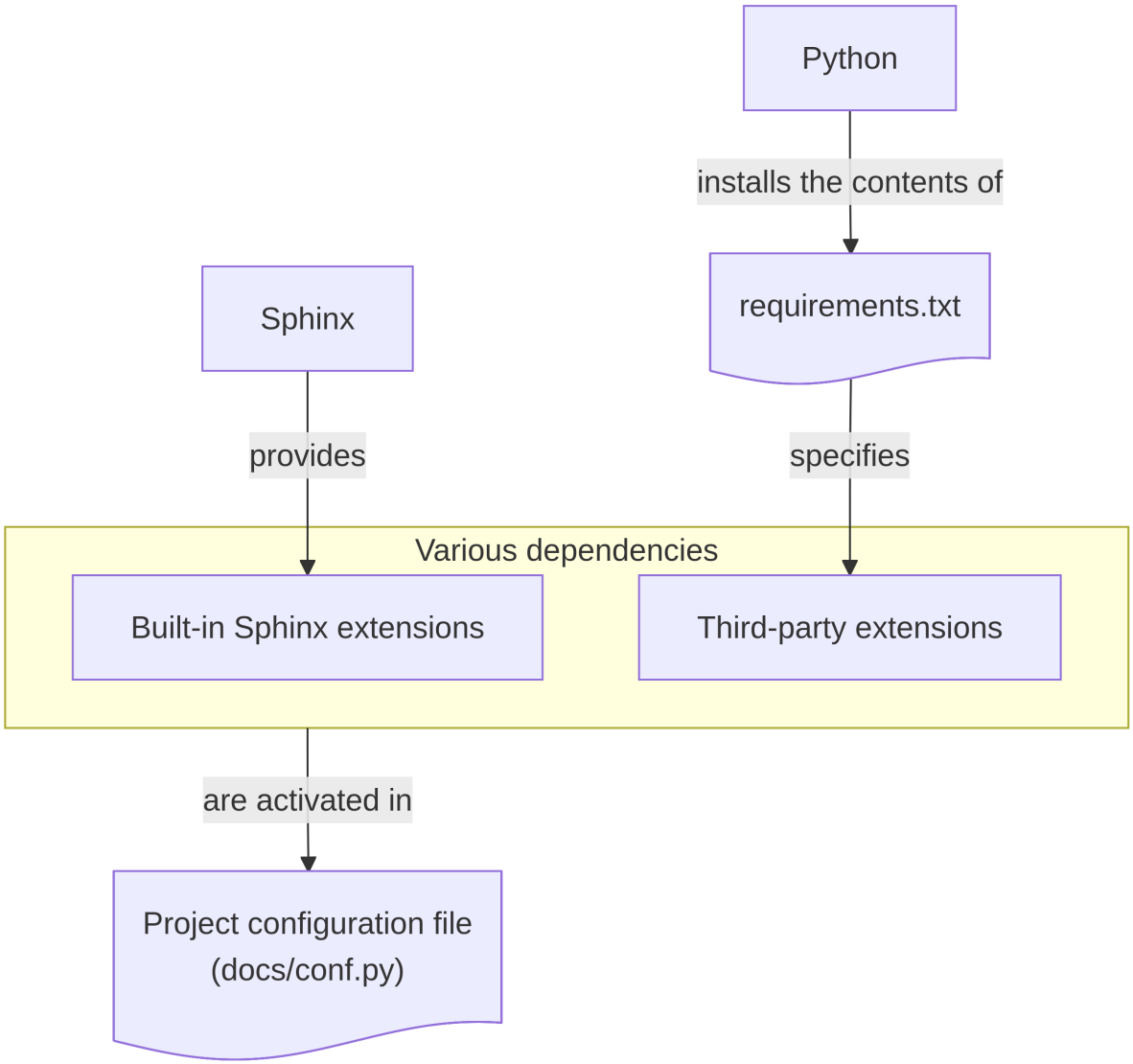


Fig. 5: Extension types

Built-in extensions

Built-in extensions do not need to be installed separately from Sphinx and can be enabled through the configuration file. The `conf.py` file has already been configured to enabled typical extensions necessary for documentation work.

Third-party extensions

If an extension is not built into Sphinx, you must include it in the `requirements.txt` file before enabling it in the Sphinx configuration file.

Extensions are Python packages, and the Sphinx Stack manages them with a requirements.txt[156] file.

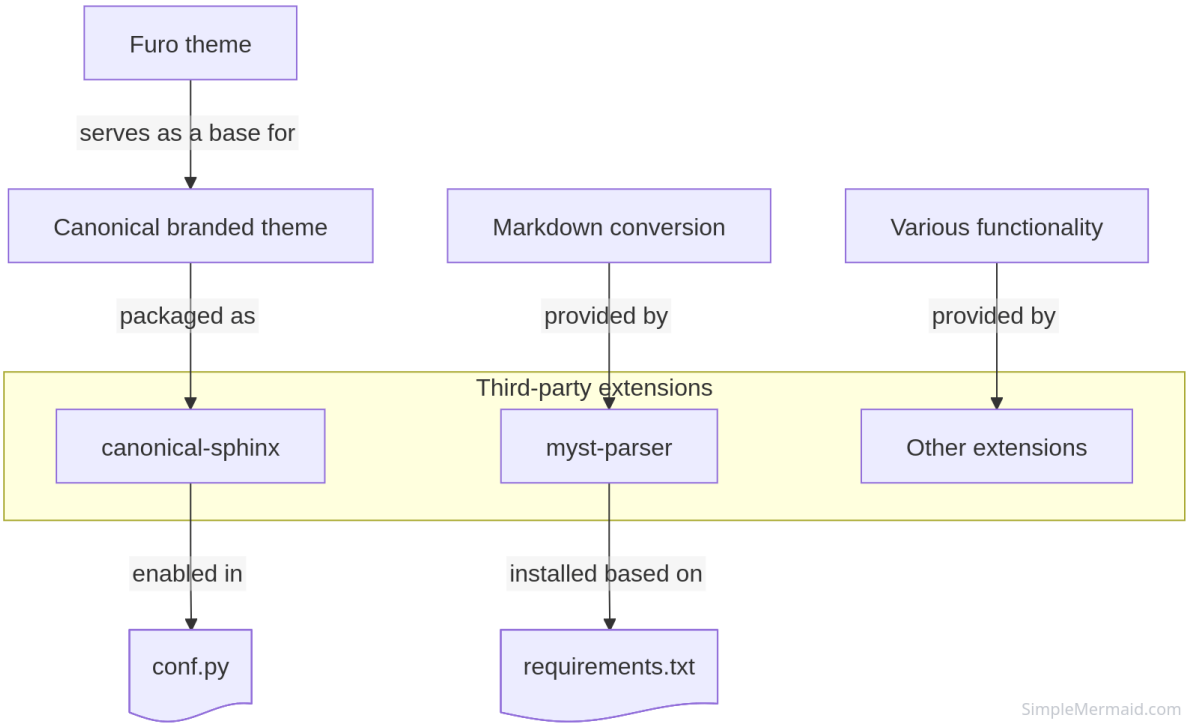


Fig. 6: Third-party extensions

Markdown support

By default, Sphinx uses reStructuredText. Markdown is supported through the MyST parser[157], which is enabled with the `myst-parser` extension.

Canonical theme

The Canonical theme is packaged as a standalone canonical-sphinx[158]) extension. It is based on Furo[159] and is designed to follow Canonical branding.

[156] https://pip.pypa.io/en/stable/reference/requirements-file-format/
[157] https://myst-parser.readthedocs.io/en/latest/
[158] https://github.com/canonical/canonical-sphinx
[159] https://github.com/pradyunsg/furo

Command-line tools

The Sphinx Stack uses Make as its local build system. The Makefile in the Sphinx Stack provides a command-line interface for setting up the virtual environment, installing dependencies, building the documentation, and more.

Makefile

Some of the Makefile targets (such as `html` and `linkcheck`) provide Sphinx-native functionality for building documentation or performing tests in a simplified form while managing required dependencies. For example, instead of using the `sphinx-build linkcheck SOURCEDIR OUTPUTDIR` command, you can use `make linkcheck`.

See *Build* (page 132) to learn how the local build process works.

Additionally, the Makefile provides commands to trigger third-party CLI tools, such as the Vale prose linter for *Style guide linting* (page 18).

Read The Docs configuration file

Read The Docs is a documentation building and hosting platform. It takes the documentation created using Sphinx (or other tools) and builds and publishes it online.

If you are publishing your documentation through Read the Docs, the Read the Docs build logic is declared in `.readthedocs.yaml`. The Sphinx Stack comes with a pre-configured `.readthedocs.yaml` with default values that should work for the majority of projects.

See *Publish on Read the Docs* (page 20) to learn how configure your Read the Docs instance.

Build

The Sphinx Stack uses Make as its build system. Make was chosen because it's well-tested and available on all platforms. The majority of the build configuration is defined in `docs/Makefile`.

Make is also the user interface for operating a project's docs. Authors and developers call all docs actions with `make <action>`.

The docs build depends on environment variables like `BUILDDIR` to locate critical files. They are conditional variables, so other systems can invoke the build at different locations without changing the docs `Makefile`.

Being primarily a Python project, all dependencies are stored in a virtual environment, `docs/.venv` by default. The environment is ephemeral and subject to frequent change. Installing the Sphinx Stack initializes it, while cleaning and updating tears it down and rebuilds it.

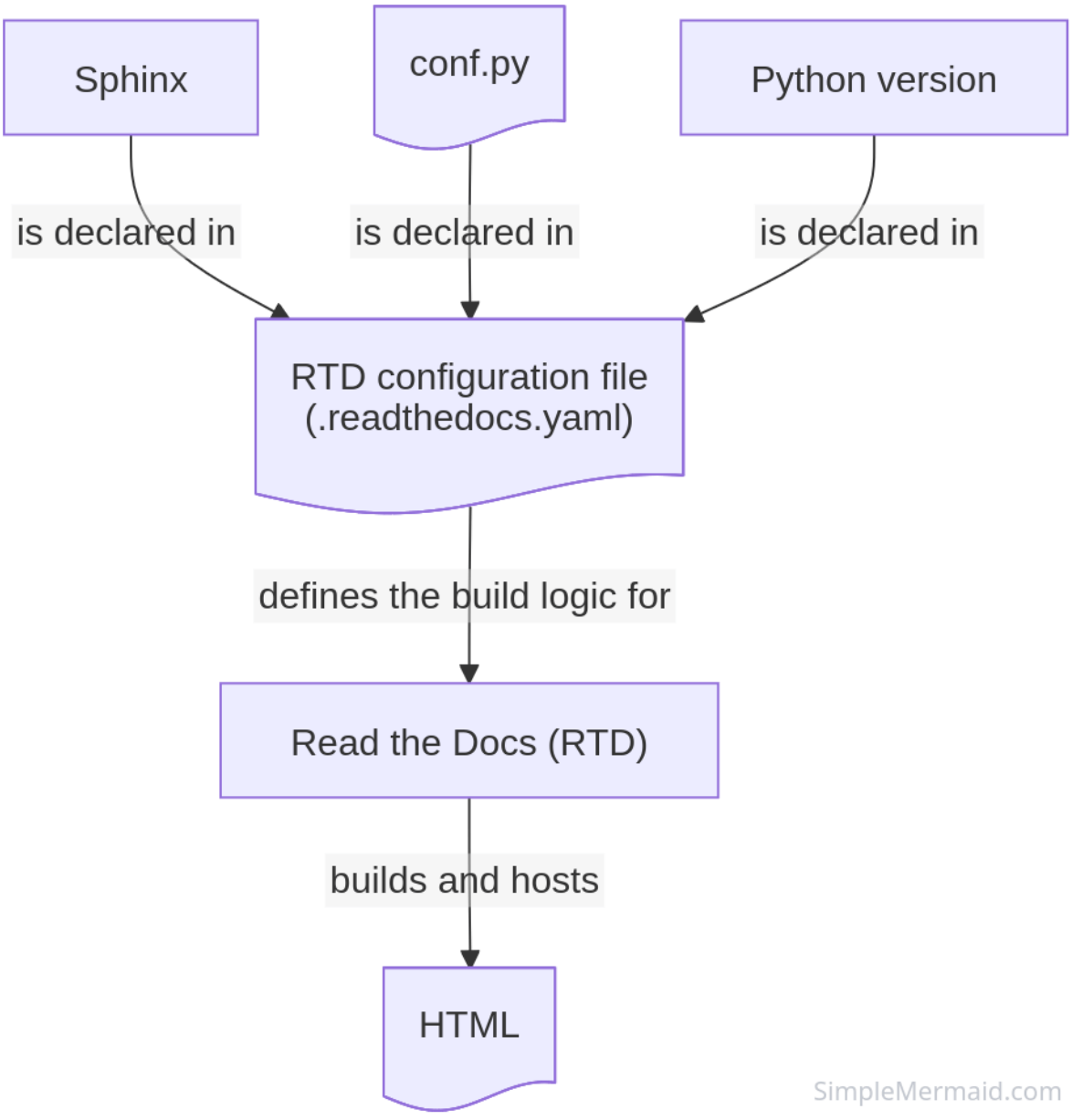


Fig. 7: Read the Docs build configuration

Parent projects and the build

The Sphinx Stack is arranged as a standalone project. When it's used in a larger project, the docs are a subsystem among other components.

If the parent project uses a build system, Make or otherwise, the doc build exists in parallel with the parent build. When embedded, the virtual environment and build recipe files would be arranged along these lines:

```
Project
│
├── ...
├── docs
│   ├── ...
│   ├── .venv
│   └── Makefile
├── src
│   └── ...
├── <parent build recipes>
└── <parent virtual environment>
```

With embedded docs, the parent build doesn't mesh with the docs build, which can cause difficulty:

- Contributors typically call the build commands from the root of the project. Some will find it inconvenient to run the docs commands by switching to the `docs` directory or manually calling the docs Makefile with `make -C docs <action>`.
- Storing multiple virtual environments bloats the host system. It's reasonable for project maintainers to prefer a shared build environment.
- The update process in the Sphinx Stack can make changes to many files in the `docs` directory. Updating is potentially much simpler if the parent project modifies only a minimum of files in the directory.
- With quality assurance and continuous integration, it's simpler if the project can use the same interface to run local and remote checks. More specifically, the parent build system and CI need a way to call the `links`, `spelling`, and `vale` checks in the Sphinx Stack.

One possible resolution is for the parent build to manually recreate the docs build, tightly coupling the parent build to the existing docs configuration. But this poses another challenge, because the docs `Makefile` might change during a Sphinx Stack update, requiring a rewrite of the parent build recipes.

The solution to these complications is to create a bridge between the two builds, from the parent build to the docs `Makefile`. *Bridge project and documentation builds* (page 46) is a guide for how to do this.

Sitemaps

Sitemaps help search engines discover and index documentation pages. The Sphinx Stack generates a sitemap with the sphinx-sitemap[160] extension. A sitemap will be available at different locations, depending on how it is generated.

Read the Docs-generated sitemaps

RTD generates a basic sitemap pointing to the index page, and relies on crawlers to index the site. This is sufficient for some projects, but RTD does not generate sitemaps for subprojects. This means any project under the Ubuntu documentation library project must generate its own sitemap.

Sitemaps generated by Read the Docs are available at the base domain of a project. For example, https://sphinx-sitemap.readthedocs.io/sitemap.xml.

`sphinx-sitemap`-generated sitemaps

The Sphinx Stack builds the documentation with the `dirhtml` builder by default. The `dirhtml` builder is required because sitemap links use directory-style URLs. Projects using an older version of the Sphinx Stack or a different builder generate malformed sitemap links.

The Sphinx Stack includes the `sphinx-sitemap` extension and its default configuration. Sitemaps generated by `sphinx-sitemap` are placed in the base of the URL schema used by the RTD instance. For example, https://sphinx-sitemap.readthedocs.io/en/latest/sitemap.xml.

For the available sitemap settings and their default values, see *Sitemap configuration* (page 92). For instructions on supporting versioned documentation, see *Manage sitemaps for versioned documentation* (page 70).

3.3.5. Release notes

This page lists the notes for past releases of the Sphinx Stack, which summarize new features, bug fixes and breaking changes in each version. It also contains the release and support policies for the Sphinx Stack.

Latest release

- *Sphinx Stack 3.0* (page 136)

[160] https://sphinx-sitemap.readthedocs.io/en/latest/index.html

Past releases

- *Canonical's Sphinx Starter Pack 2.0* (page 141)
- *Canonical's Sphinx Starter Pack 1.6* (page 146)
- *Canonical's Sphinx Starter Pack 1.5* (page 148)
- *Canonical's Sphinx Starter Pack 1.4* (page 151)

Release versioning

Sphinx Stack version naming follows the Semantic Versioning 2.0.0 scheme with modifications. It has numbers for major and minor version, but *not* patch versions.

Version	Example	Signifies
Major	**1**.3	Improvements that aren't backward-compatible.
Minor	1.**3**	New or improved features that are simple to adopt and backward-compatible.

Sphinx Stack 3.0

16 October 2026

These release notes cover new features and changes in the Sphinx Stack. For a more granular list of changes, check out the full commit log[161].

Bring these changes into your docs:

- *Update Starter Pack 1.3.0 and higher* (page 25)
- *Update Starter Pack 1.2.0 and lower* (page 28)

What's new

New theme

Sphinx Stack 3.0 is the first to ship with default support for Ulwazi[162], our new Sphinx theme based on the Vanilla web framework. Its 1.0 release is the culmination of a long effort from several authors, and it represents a leap forward in the look and feel of Canonical documentation.

To set up and configure Ulwazi, follow the steps in *Configure the Ulwazi theme* (page 33) as part of your next update. If you run into any bugs or issues during adoption, please report them on GitHub[163].

[161] https://github.com/canonical/sphinx-docs-starter-pack/compare/2.0...3.0
[162] https://github.com/canonical/ulwazi
[163] https://github.com/canonical/ulwazi/issues/new

New local documentation workflow

Sphinx Stack 3.0 introduces a new workflow for running the spelling, inclusive language, and link checks. Instead of calling the older remote workflow, which points right back to the project's Makefile, the new workflow is self-contained. This change brings more secure, performant, and easily customizable documentation checks to your project's CI pipeline.

To bring in these improvements, copy the new automatic-doc-checks.yaml[164] file into your project's `.github/workflows/` directory during your next update.

New CLI

The Sphinx Stack isn't a tool, but it's not only a collection of files, either. Every time you use it, you enter commands in the terminal, using special words like `html` and `vale`. It in effect contains a CLI experience like many of the tools Canonical ships, only without a standalone binary package. The ergonomics of the Sphinx Stack – how it fits, feels, and responds to your input – is important.

We're in the process of remaking the CLI. Our intent is to bring it into compliance with the design and user experience standards at Canonical, and put it at par with other best-in-class Canonical tools. Reworking all 14 commands using a new task runner takes much iteration and testing, so we've slated it for the Sphinx Stack 4.0 release. If you'd like to follow development, you can find the latest working version in the cli-redesign branch.

Breaking changes

Required job names

On GitHub, required jobs in CI are selected based on their name. So, if you require pull requests to pass the spelling, inclusive language, and link checks in your project, you must update your repository settings with the new job names.

For example, if your repository lists "documentation-checks / Spell check" as a required check, replace it with "Documentation / Spelling check".

Removed `none` role

The Sphinx Stack now defaults to version 1.0 of the sphinx-roles extension, which removed the `none` role.

Replace any uses of the `none` role with comments:

```
- :none:`This text isn't rendered.`
+ .. This text isn't rendered.
```

[164] https://github.com/canonical/sphinx-stack/blob/main/.github/workflows/automatic-doc-checks.yml

Removed sphinx-tabs extension

The sphinx-tabs extension is no longer included by default.

If your documentation contains `tabs` and `tab` directives, replace them with the `tab-set` and `tab-item` directives from the sphinx-design extension:

```
- .. tabs::
-
-    .. tab:: Item 1
-
-       This is the first item.
-
-    .. tab:: Item 2
-
-       This is the second item.
+ .. tab-set::
+
+     .. tab-item:: Item 1
+
+        This is the first item.
+
+     .. tab-item:: Item 2
+
+        This is the second item.
```

Removed sphinx-contributor-listing extension

The sphinx-contributor-listing extension is no longer included by default.

This extension isn't supported by Ulwazi, as it no longer aligns with our best practices for page headers and footers. If your project still includes references to this extension in its footer, remove them from the footer template when you update.

Removed sphinx-filtered-toctree extension

The sphinx-filtered-toctree extension is no longer included by default. This extension was part of a legacy feature and is no longer needed.

If your `conf.py` file assigns a value to `toc_filter_exclude`, remove the line:

```
- toc_filter_exclude = ["foo", "bar"]
```

Then, replace any `filtered-toctree` directives with the `toctree` directive:

```
- .. filtered-toctree::
-
-   page-a
-   :foo:page-b
+ .. toctree::
+
+   page-a
```

If you find a page prefixed with one of the strings previously listed with `toc_filter_exclude`, evaluate its usefulness. If it should be rendered, remove the prefix. If not, delete the entry and its respective file.

Removed sphinx-last-updated-by-git extension

The sphinx-last-updated-by-git extension is no longer included by default. Its inclusion in the `docs/requirements.txt` file was redundant, as it's already set up by the sphinx-sitemap extension, which all Canonical documentation projects should use.

Ensure that you have the sphinx-sitemap extension installed and configured. For guidance, refer to *Sitemaps* (page 135).

Removed sphinx-ubuntu-images extension

The sphinx-ubuntu-images extension is no longer included by default. This extension isn't commonly used and therefore isn't suited for the inclusion in the Sphinx Stack.

If your documentation *does* use the `ubuntu-images` directive, be sure to add it back to your `requirements.txt` and `conf.py` files after you update.

Removed sphinx-reredirects extension

The sphinx-reredirects extension is no longer included by default. Redirects, both internal and external, should be handled with the sphinx-rediraffe extension.

For guidance on redirecting pages with the sphinx-rediraffe extension, follow the steps in *Redirect pages* (page 45).

Deprecated features

The following features are deprecated. We plan to remove them in the 4.0 release.

Python 3.10

Python 3.10 reached end of life on 1 October 2026 and will not be supported in Sphinx Stack 4.0 and higher.

Several Sphinx Stack dependencies, including Sphinx itself, have already dropped Python 3.10 support in recent releases. To ensure that the Sphinx Stack can continue to incorporate and deliver upstream improvements, its 4.0 release will follow suit. This change will break compatibility with stock Ubuntu 22.04 LTS systems, which shipped with Python 3.10.4. Keep this in mind when setting up documentation environments and CI runners moving forward.

Callable workflows

The following Sphinx Stack workflows will no longer be callable in Sphinx Stack 4.0 and higher:

- `sphinx-python-dependency-build-check.yaml`
- `markdown-style-checks.yaml`
- `check-removed-urls.yaml`

These workflows are meant to be duplicated in downstream projects, and the inclusion of the `workflow_call` trigger has been misleading for many projects.

If you need to call these workflows across a wide spread of projects and don't want to maintain multiple copies, it's recommended that you maintain your own callable source workflows. This will give you more granular control over their behavior and will make you more resilient to Sphinx Stack changes.

requirements.txt file

The `docs/requirements.txt` file is being replaced with a `docs/pyproject.toml` file in Sphinx Stack 4.0.

A `pyproject.toml` file will allow more flexibility and reproducibility in documentation builds and is preferred in modern Python. If your project has additional tooling referencing the current `requirements.txt` file path, such as custom Make targets or CI workflows, then you'll need to update the path when you update to Sphinx Stack 4.0.

Dependency changes

Package	From	To
ulwazi		~=1.0
packaging	~=26.1	~=26.3
sphinx-autobuild		==2024.10.03
sphinx-llm	~=0.4	~=1.1
sphinx-rerediraffe	>0.0.3, <1.0.0	~=0.1
sphinx-roles	~=0.1	~=1.0
sphinx-terminal	~=1.0	~=1.1

Some Sphinx Stack dependencies can't be updated until we drop Python 3.10 in the 4.0 release. These dependencies are being tracked in https://github.com/canonical/sphinx-stack/issues/645. If you encounter a similar breakage with the current set of constraints, please describe it in the issue thread.

Bug fixes

The following bugs were fixed for the 3.0 release:

- #638[165]: The default `.gitignore` file can ignore certain docs paths
- #626[166]: The Markdown linter checks the `_build` directory
- #611[167]: Pa11y runs on the `_dev` directory

Contributors

We'd like to express a big thank you to everyone who contributed to this release:

`asanvaq`[168], `a-velasco`[169], `bepri`[170], `bjdean-canonical`[171], `dwilding`[172], `edibotopic`[173], `elijahgreenstein`[174], `erinecon`[175], `geoffreynyaga`[176], `izmalk`[177], `jahn-junior`[178], `JordanMG13`[179], `medubelko`[180], `mhsun-se`[181], `nhennigan`[182], `odadacharles`[183], `rajannpatel`[184], `rkratky`[185], `SecondSkoll`[186], `shaloo`[187], and `zayatamal`[188]

Canonical's Sphinx Starter Pack 2.0

30 April 2026

These release notes cover new features and changes in the project formerly known as Canonical's Sphinx Starter Pack. For a more granular list of changes, check out the full commit log.

Bring these changes into your docs:

- *Update Starter Pack 1.3.0 and higher* (page 25)
- *Update Starter Pack 1.2.0 and lower* (page 28)

[165] https://github.com/canonical/sphinx-stack/issues/638
[166] https://github.com/canonical/sphinx-stack/issues/626
[167] https://github.com/canonical/sphinx-stack/issues/611
[168] https://github.com/asanvaq
[169] https://github.com/a-velasco
[170] https://github.com/bepri
[171] https://github.com/bjdean-canonical
[172] https://github.com/dwilding
[173] https://github.com/edibotopic
[174] https://github.com/elijahgreenstein
[175] https://github.com/erinecon
[176] https://github.com/geoffreynyaga
[177] https://github.com/izmalk
[178] https://github.com/jahn-junior
[179] https://github.com/JordanMG13
[180] https://github.com/medubelko
[181] https://github.com/mhsun-se
[182] https://github.com/nhennigan
[183] https://github.com/odadacharles
[184] https://github.com/rajannpatel
[185] https://github.com/rkratky
[186] https://github.com/SecondSkoll
[187] https://github.com/shaloo
[188] https://github.com/zayatamal

What's new

Rename to Sphinx Stack

This project was named when the documentation team at Canonical started assembling a standard documentation toolkit. The word starter felt appropriate, as products needed their documentation bootstrapped, and we had yet to release a standard. More than two years later, this is no longer the case. This project has been deployed in hundreds of Canonical products. It represents a complete standard for Canonical documentation sets, not just those starting out. So, we decided that it's time for a new name.

With the release of version 2.0, we're renaming the project **Sphinx Stack**. The repository is now located at https://github.com/canonical/sphinx-stack.

Separate documentation repository

To simplify the process of setting up and maintaining the Sphinx Stack, we separated its documentation into a new Sphinx Stack documentation repository[189].

Now, when you create a new project with the Sphinx Stack, you no longer need to remove its documentation. New projects are also no longer initialized with any configuration specific to the Sphinx Stack docs. Now, there's less ambiguity as to whether you need to incorporate new values into your configuration file.

sphinx-llm support

This version introduces default support for the sphinx-llm[190] extension. It surfaces documents in accordance with the emerging llms.txt standard[191].

You can think of the `llms.txt` file as an extra sitemap for LLMs and systems using the Model Context Protocol. In your documentation builds, you'll now see an `llms.txt` file, an `llms-full.txt` file, and Markdown copies of your pages. The content of these files is copied straight from your existing docs – it doesn't transform or summarize the text.

Dependency constraints

In the past, we've had upstream packages break documentation builds without warning. To reduce this, we constrained the project's dependencies to their current major version.

Before every future major release, we'll test the newest versions and adjust the constraints appropriately.

[189] https://github.com/canonical/sphinx-stack-docs
[190] https://github.com/nvidia/sphinx-llm
[191] https://llmstxt.org

Copyright statement

We aligned the copyright statement in the footer with legal standards. To incorporate this change in your documentation, add the license information to `html_context` in your `conf.py` file:

Listing 56: docs/conf.py

```
html_context = {
    "license": {
        "name": "CC-BY-SA-3.0",
        "url": "https://github.com/canonical/sphinx-stack/blob/main/LICENSE",
    },
}
```

For `license.name`, use the standard shorthand identifier from SPDX[192].

For `license.url`, link to the LICENSE statement, typically found on the product's home page or in its GitHub repository.

Ulwazi testing

We invite you to *test Ulwazi* (page 33), our new documentation theme in early access.

It represents a leap forward in the user experience, and we'd like your feedback. If you run into any bugs or issues, please report them on GitHub[193].

Breaking changes

Renamed the `.sphinx/` directory

The `.sphinx/` directory, which contains config files and utility scripts, has been renamed to `_dev` to better reflect its purpose.

Once you've updated, remember to remove the `.sphinx` directory from your project. If your project has custom code that references this directory, you'll also need to update these references.

Removed the `reuse/` directory

To reduce complexity and improve source readability, we've removed the `reuse/` directory.

By default, this directory contained project-wide link URLs and substitutions. Links should now be included with the inline syntax or, if used more than three times in a file, included at the end of the file. Substitutions will need to use reST or MyST syntax and should be placed at the top of the file. The syntax for these changes is described in *External links* (page 100) and *Substitution* (page 110).

[192] https://spdx.org/licenses
[193] https://github.com/canonical/ulwazi/issues/new

Deprecated features

The following features are deprecated. We plan to remove them in the 3.0 release.

`none` role

The `none` role is deprecated in 2.0.

To prepare for this role's removal, replace any uses of it with comments.

```
- :none:`This text isn't rendered.`
+ .. This text isn't rendered.
```

sphinx-tabs extension

The sphinx-tabs extension is deprecated in 2.0.

If your documentation contains `tabs` and `tab` directives, replace them with the `tab-set` and `tab-item` directives from the sphinx-design extension:

```
- .. tabs::
-
-    .. tab:: Item 1
-
-       This is the first item.
-
-    .. tab:: Item 2
-
-       This is the second item.
+ .. tab-set::
+
+     .. tab-item:: Item 1
+
+        This is the first item.
+
+     .. tab-item:: Item 2
+
+        This is the second item.
```

sphinx-contributor-listing extension

The sphinx-contributor-listing extension is deprecated in 2.0. To simplify the header and footer design, we're ending support for per-page contributor lists in the next major release.

sphinx-filtered-toctree extension

The sphinx-filtered-toctree extension is deprecated in 2.0. This extension was part of a legacy feature and is no longer needed.

If your `conf.py` file assigns a value to `toc_filter_exclude`, remove the line:

Listing 57: docs/conf.py

```
- toc_filter_exclude = ["foo", "bar"]
```

Then, replace any `filtered-toctree` directives with the `toctree` directive. If you find a page prefixed with one of the strings previously listed with toc_filter_exclude, evaluate its usefulness. If it should be rendered, remove the prefix. If not, delete the entry and its respective file.

```
- .. filtered-toctree::
-
-   page-a
-   :foo:page-b
+ .. toctree::
+
+   page-a
```

sphinx-last-updated-by extension

The sphinx-last-updated-by-git extension is deprecated in 2.0. Its inclusion in the docs/requirements.txt file is redundant, as it's already set up by the sphinx-sitemap extension.

Ensure that you have the sphinx-sitemap extension installed and configured. For guidance, refer to *Sitemaps* (page 135).

sphinx-ubuntu-images extension

The sphinx-ubuntu-images extension is deprecated in 2.0. This extension isn't commonly used and won't be included by default in the next major release.

If your documentation uses the `ubuntu-images` directive, be sure to add it back to your `requirements.txt` and `conf.py` files after you update.

sphinx-reredirects extension

The sphinx-reredirects extension was deprecated in 1.6. Redirects, both internal and external, should now be handled with the sphinx-rediraffe extension.

Redirect pages (page 45) contains guidance on redirecting pages with the sphinx-rediraffe extension.

Contributors

We'd like to express a big thank you to everyone who contributed to this release:

a-velasco[194], asanvaq[195], AshleyCliff[196], izmalk[197], jahn-junior[198], keirthana[199], medubelko[200], mhsun-se[201], nhennigan[202], odadacharles[203], Saviq[204], srbouffard[205], and yhontyk[206],

Canonical's Sphinx Starter Pack 1.6

24 March 2026

These release notes cover new features and changes in Canonical's Sphinx Starter Pack 1.6. The full commit log[207] is available on GitHub.

Bring these changes into your docs:

- *Update Starter Pack 1.3.0 and higher* (page 25)
- *Update Starter Pack 1.2.0 and lower* (page 28)

What's new in 1.6

New extension for redirects

If your project has a lot of redirects, they can begin to clutter your build configuration file. To remedy this, we changed the default extension for redirects to sphinx-reredirraffe, a fork of sphinxext-rediraffe that supports external redirects and cleans up destination URLs.

In your docs, you can now list redirects in a `redirects.txt` file and configure your project to use them with:

Listing 58: conf.py

```
- redirects = {
-     "source-path": "destination-path"
- }
+ rediraffe_redirects = "redirects.txt"
+ rediraffe_dir_only = True
```

[194] https://github.com/a-velasco
[195] https://github.com/asanvaq
[196] https://github.com/AshleyCliff
[197] https://github.com/izmalk
[198] https://github.com/jahn-junior
[199] https://github.com/keirthana
[200] https://github.com/medubelko
[201] https://github.com/mhsun-se
[202] https://github.com/nhennigan
[203] https://github.com/odadacharles
[204] https://github.com/Saviq
[205] https://github.com/srbouffard
[206] https://github.com/yhontyk
[207] https://github.com/canonical/sphinx-stack/releases/tag/1.6

If your project uses relative redirects, these should be rewritten relative to the root of the docs directory.

For example, if you were previously redirecting `/how-to/set-up/project/` to `../set-up-project/`, the redirect should now read:

Listing 59: redirects.txt

```
"how-to/set-up/project/" "how-to/set-up-project/"
```

Remote cookie banner files

Following the recent changes to the cookie banner, we evaluated how its files are distributed and maintained. In version 1.6, these files are hosted centrally so you don't need to duplicate them to enable Google Analytics for your project.

If you've already enabled Google Analytics in your project, you should make it a priority to use the centrally-hosted files, as they contain important legal changes. For guidance, refer to *Enable Google Analytics* (page 52).

Virtual environment path

To make the Python virtual environment more discoverable, we moved it from `docs/.sphinx/venv` to `docs/.venv`. This also preserves the `.sphinx/` directory's scope, which is to house static utility files.

Once you've replaced all occurrences of the old path, remove the old virtual environment.

Ulwazi testing guide

Our new Sphinx theme based on the Vanilla framework, Ulwazi, is now available for testing. To try it out, follow the new guide, *Configure the Ulwazi theme* (page 33).

Starter Pack explanation

We published *Sphinx Stack structure* (page 128) to explain the functionality of and relationships between the core elements of the Starter Pack.

Deprecated features

The following features are deprecated and will be removed in an upcoming version of the Starter Pack.

sphinx-reredirects

The sphinx-reredirects extension is deprecated in 1.6. Redirects, both internal and external, should now be handled with sphinx-rediraffe.

Fixed for 1.6

- #535[208] Markdown checks are incompatible with default runner version
- #550[209] Makefile doesn't allow for pymarkdownlnt to exit with a status code of 1

Contributors

We would like to express a big thank you to all the people who contributed to this release:

`akcano`[210], `AnneCYH`[211], `arturo-seijas`[212], `jahn-junior`[213], `odadacharles`[214], `wctaylor`[215], and `yhontyk`[216]

Canonical's Sphinx Starter Pack 1.5

26 February 2026

These release notes cover new features and changes in Canonical's Sphinx Starter Pack 1.5. The full commit log[217] is available on GitHub.

Bring these changes into your docs:

- *Update starter pack 1.3.0 and higher* (page 25)
- *Update starter pack 1.2.0 and lower* (page 28)

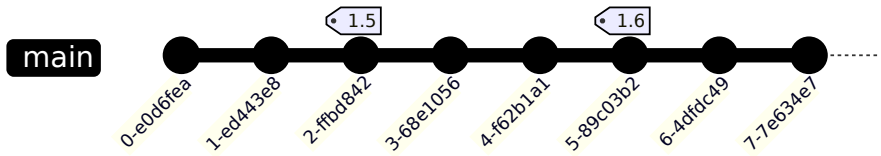
What's new in 1.5

New versioning

We've revised our approach to versions. Because the starter pack is an evergreen repository and not packaged software, it requires a different version process and scheme.

Starting with 1.5, the starter pack only provides major and minor version numbers. Patch numbers now correspond to the Git commits in the repository's `main` branch between tagged releases.

[208] https://github.com/canonical/sphinx-docs-starter-pack/pull/535
[209] https://github.com/canonical/sphinx-docs-starter-pack/issues/550
[210] https://github.com/akcano
[211] https://github.com/AnneCYH
[212] https://github.com/arturo-seijas
[213] https://github.com/jahn-junior
[214] https://github.com/odadacharles
[215] https://github.com/wctaylor
[216] https://github.com/yhontyk
[217] https://github.com/canonical/sphinx-stack/releases/tag/1.5



Going forward, we'll continue to tag major and minor releases, and distribute hotfixes to the tip of the repository's history. If untagged patch versions are too risky, or your software policy requires that you lock to a specific version, we recommend you pin to the latest minor version.

New upgrade guide

For docs based on starter pack 1.3 or higher, we published *Update the new Sphinx Stack* (page 25).

Contribution guide

We've added a contribution guide for the project in `CONTRIBUTING.md`. It's available in upgraded repositories and on GitHub[218].

This guide is for developers of the starter pack. It's not meant for your project. After updating, remove it from your project.

Quiet link check

The link check by default was too verbose, logging every checked link.

The check now only logs on errors.

Environment variable updates

For docs embedded into larger projects, the Make commands might be called in the project's main build system. Because docs builds used common names like `BUILDDIR`, it was possible that they could conflict with variables set by the parent build system meant for other software.

We made all the docs variables conditional, so you can set them ahead of time in the parent build system.

To mediate the risk of conflicts, we overhauled the variable names, adding a prefix and splitting along common word boundaries.

[218] https://github.com/canonical/sphinx-stack/blob/main/CONTRIBUTING.md

Old variable name	New variable name
VENVDIR	DOCS_VENVDIR
VENV	DOCS_VENV
SOURCEDIR	DOCS_SOURCEDIR
REQPDFPACKAGES	DOCS_PDFPACKAGES
VOCAB_CANONICAL	DOCS_VOCAB
SPHINXDIR	SPHINX_DIR
SPHINXBUILD	SPHINX_BUILD
VALEDIR	VALE_DIR
VALECONFIG	VALE_CONFIG
TARGET	CHECK_PATH

Dependency changes in 1.5

Package	From	To
canonical-sphinx	>=0.5.1	~=0.6
sphinx-related-links	>=0.1.1	>=0.1.2

Removed in 1.5

`wokeignore` directive

The `wokeignore` directive is removed because it was redundant. Use the *vale-ignore* (page 18) role instead.

Contribution guide template

We removed a contribution guide template that was placed in the documentation in error.

Fixed in 1.5

- #472[219] invalid characters in Makefile cause infinite loop
- #477[220] Version slug is duplicated in sitemaps
- #501[221] Example-product docs are used in links but no longer available
- #514[222] Workflows can be called by other projects

Contributors

We would like to express a big thank you to all the people who contributed to this release:

`a-velasco`[223], `akcano`[224], `jahn-junior`[225], `medubelko`[226], `minaelee`[227], `nhennigan`[228], and `odadacharles`[229]

Canonical's Sphinx Starter Pack 1.4

28 January 2026

These release notes cover new features and changes in Canonical's Sphinx Starter Pack 1.4.0. The full commit log[230] is available on GitHub.

What's new in 1.4.0

Bridging the build

The docs build and its Make targets are hosted in the `docs` directory. If the docs are in a larger project with its own build system, it can be difficult to maintain and run the two separate builds.

We've taken some initial steps to make the docs build easier to integrate into larger projects. With 1.4.0, the docs `Makefile` is now properly callable by a parent build system, because the

[219] https://github.com/canonical/sphinx-docs-starter-pack/pull/472
[220] https://github.com/canonical/sphinx-docs-starter-pack/pull/477
[221] https://github.com/canonical/sphinx-docs-starter-pack/pull/501
[222] https://github.com/canonical/sphinx-docs-starter-pack/pull/514
[223] https://github.com/a-velasco
[224] https://github.com/akcano
[225] https://github.com/jahn-junior
[226] https://github.com/medubelko
[227] https://github.com/minaelee
[228] https://github.com/nhennigan
[229] https://github.com/odadacharles
[230] https://github.com/canonical/sphinx-docs-starter-pack/compare/1.3.1...1.4.0

main build can ensure the `VENVDIR`, `BUILDDIR`, and `VALEDIR` variables are set to the right locations in the project.

The new guide *Bridge project and documentation builds* (page 46) covers in detail how to integrate the docs build in a parent Python project.

Build explanation

We've released an *explanation of the starter pack build* (page 132) to share with you some of its design and inner workings.

Removed in 1.4.0

Metrics check

The metrics check was a set of scripts called with `make allmetrics` that collected measures and statistics about source documents and rendered HTML files.

We've removed this check because it was underused and underdeveloped. There have been no reported usages of it from any active projects at Canonical.

Fixed bugs and issues

1.4.0

- #496[231] myst-parser causes Python dependency check to time out
- #509[232] Markdown linter falsely flags contents of `.sphinx` dir

1.4.1

- #444[233] Patch the `version` file so the upgrade script works

Contributors

We would like to express a big thank you to all the people who contributed to this release:

`medubelko`[234], `minaelee`[235], and `tang-mm`[236]

[231] https://github.com/canonical/sphinx-docs-starter-pack/pull/496
[232] https://github.com/canonical/sphinx-docs-starter-pack/issues/509
[233] https://github.com/canonical/sphinx-docs-starter-pack/pull/516
[234] https://github.com/medubelko
[235] https://github.com/minaelee
[236] https://github.com/tang-mm

3.3.6. Contribute

The Sphinx Stack is a shared foundation for Sphinx documentation projects. It combines Sphinx configuration, styling, extensions, and quality checks to help projects build and maintain documentation.

Contributing to the Sphinx Stack helps improve the documentation of all its users. The Documentation Practice team performs most of the work, but all contributors are welcome.

Scope of contribution

Good candidates for contribution include:

- Bug fixes for build errors, broken links or configuration issues
- Improvements to defaults, extensions, workflows or style rules that benefit all users
- Dependency updates for security, compatibility or better tooling
- Documentation improvements such as new guides, corrected references or clearer tutorials

Changes that are specific to a single project, introduce unnecessary complexity, or conflict with existing workflows may not be suitable. If unsure, discuss the change in a GitHub issue before starting work.

Standards and expectations

Before contributing, review the following requirements and guidelines.

Ubuntu Code of Conduct

When contributing, you must abide by the Ubuntu Code of Conduct[237]. Projects governed by Canonical expect good conduct and excellence from every member.

Canonical Contributor License Agreement

Contributions can only be accepted from contributors who have signed the Canonical Contributor License Agreement (CLA)[238].

By signing the CLA, you grant Canonical permission to use your contribution. You retain ownership of any work you contribute.

Review the terms of the agreement before signing it or committing anything. If you agree and sign it, your work can be incorporated into the repository.

[237] https://ubuntu.com/community/ethos/code-of-conduct
[238] https://ubuntu.com/legal/contributors

Open source license

The Sphinx Stack is licensed under GPL-3.0[239].

The documentation is licensed under CC-BY-SA-3.0[240].

Ways to contribute

- *Contribute to development* (page 154)
- *Contribute to documentation* (page 160)

Report an issue

If you find a bug, identify missing functionality or want to propose an improvement, search the project's existing GitHub issues first before creating a new one.

Because the Sphinx Stack is a template repository, its code and documentation are maintained in separate repositories. Be sure to open any issues in the correct repository:

- Sphinx Stack issues[241]
- Documentation issues[242]

Get help

If you're unsure whether a change belongs in the Sphinx Stack, open an issue and discuss it with the maintainers.

Contributors of all experience levels are welcome.

Contribute to development

This guide describes how to contribute code changes to the Sphinx Stack[243].

Development contributions improve the shared configuration, extensions, themes, workflows and tooling that projects use to build documentation with the Sphinx Stack.

It requires familiarity with Git, GitHub, and the Sphinx documentation generator.

Before starting, review the *contribution requirements* (page 153).

[239] https://github.com/canonical/sphinx-stack/blob/main/LICENSE
[240] https://github.com/canonical/sphinx-stack-docs/blob/main/LICENSE
[241] https://github.com/canonical/sphinx-stack/issues
[242] https://github.com/canonical/sphinx-stack-docs/issues
[243] https://github.com/canonical/sphinx-stack

Set up your work environment

Create a personal fork[244] of the repository on GitHub.

Next, clone your fork onto your local machine:

With SSH

If you authenticate your GitHub account with SSH[245], run:

```
git clone git@github.com:<username>/sphinx-stack
cd sphinx-stack
git remote add upstream git@github.com:canonical/sphinx-stack
git fetch upstream
```

With HTTPS

If you don't authenticate with SSH, clone with HTTPS[246] instead:

```
git clone https://github.com/<username>/sphinx-stack
cd sphinx-stack
git remote add upstream https://github.com/canonical/sphinx-stack
git fetch upstream
```

Install the dependencies and build the documentation to verify your environment:

```
cd docs
make install
make html
```

If these commands complete without error, your environment is ready.

Plan your contribution

All significant development work should be associated with an issue.

Before starting work, check the existing GitHub issues[247] to see whether the task is already being discussed or tracked. Comment on the issue to have it assigned to you.

If no issue exists for your task, create one[248] before you begin.

[244] https://github.com/canonical/sphinx-stack/fork
[245] https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account
[246] https://docs.github.com/en/get-started/git-basics/about-remote-repositories#cloning-with-https-urls
[247] https://github.com/canonical/sphinx-stack/issues
[248] https://github.com/canonical/sphinx-stack/issues/new

Simple tasks

For simple, self-contained tasks, such as bug fixes or tweaks to the default configuration, check existing GitHub issues[249]. If no suitable issue exists, open one[250] and discuss the change you'd like to make with a maintainer.

Complex tasks

For complex tasks, such as introducing new features, changing workflows or adding stricter checks, describe your proposal in the issue thread.

Include the implementation plan, expected impact on projects using the Sphinx Stack, and any required tests and documentation updates. This helps the team decide whether the change aligns with the project's goals.

Contribute a change

Create a branch

Before you begin your work, sync your local copy of the code and create a new branch:

```
git fetch upstream
git checkout -b <new-branch-name>
```

Format your branch name as `<ticket-id>-<description>`. For example, if you're working on GitHub issue #235, and it's about adding a text sanitizer, you'd name your branch `issue-235-add-text-sanitizer`. Keep the name under 80 characters.

Develop

Keep the Sphinx Stack minimal by default. Changes should improve the experience of all downstream projects, rather than solve problems specific to one documentation project.

Optional features are best implemented by the projects that use the Sphinx Stack rather than added to the shared foundation.

Test

Verify your work by building the documentation and running quality checks.

```
cd docs
make html
```

Run the available quality checks:

[249] https://github.com/canonical/sphinx-stack/issues
[250] https://github.com/canonical/sphinx-stack/issues/new

```
make spelling      # Check spelling
make linkcheck     # Validate links
make woke          # Check inclusive language
make lint-md       # Check Markdown style
make vale          # Check style guide compliance (optional)
```

To preview locally with live reload at `http://127.0.0.1:8000`, run:

```
make run
```

Document your work

Document any changes that affect users.

For small changes, update the existing documentation that describes the affected functionality. This may include how-to guides, reference pages or other related content.

For a new feature or a nontrivial change to an existing one, add documentation explaining its usage and specifications. For guidance on contributing documentation, refer to *Contribute to documentation* (page 160).

All changes and fixes should be recorded in the changelog[251].

Commit a change

Register your changes with a Git commit:

```
git add -A
git commit
```

Use the Conventional Commits[252] format.

```
feat: add text sanitizer
```

Keep commits focused and separate unrelated changes or components into different commits. Review similar commits with `git log --oneline <filename>` if you are unsure which commit type to use.

Commit messages should be short and describe the purpose of the change.

[251] https://github.com/canonical/sphinx-stack/blob/main/CHANGELOG.md
[252] https://www.conventionalcommits.org/en/v1.0.0/

Complex commits

If a commit contains multiple unrelated changes, split it into smaller commits. For each commit, select the highest-ranked type that fits:

- ci
- build
- feat
- fix
- perf
- refactor
- style
- test
- docs
- chore

Sign your commits

All commits must be cryptographically signed. You can sign a commit by adding `-S` to the `git commit` command:

```
git commit -S -m "feat: add text sanitizer"
```

Signed commits display a "Verified" badge in GitHub. To configure Git to sign commits automatically, run:

```
git config --global commit.gpgsign true
```

For more information, see GitHub's documentation on signing commits[253].

Submit your contribution

Send for review

When you've committed your draft and you're ready to have it reviewed, push it to your fork:

```
git push -u origin <branch-name>
```

Open a pull request (PR) on GitHub. Give the PR a short, descriptive title using the Conventional Commits[254] format. For single-commit branches, GitHub generates this automatically from the commit message.

Include the following details in your PR:

- **Description**: Problem solved, features added or bugs fixed
- **Relevant issues**: Link related issues and PRs[255]

[253] https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits
[254] https://www.conventionalcommits.org/en/v1.0.0/
[255] https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/autolinked-references-and-urls

- **Testing**: How reviewers can verify the change or test the fix
- **Reversibility**: Reasoning and steps for decisions that are difficult to reverse

PRs are typically reviewed within a week.

Address quality concerns

Before the PR is merged, it must pass all automated checks that validate builds, dependencies and documentation quality.

If there are any issues in your branch that your local testing didn't catch, then the automatic checks will fail. To address these issues, review the logs in the failed checks. The error messages in the logs will have remedies and hints for what needs fixing.

Address review feedback

When the maintainers review the PR, they may suggest improvements or request changes. Address them in follow-up commits to your branch, the same way you committed and pushed changes while drafting. Commit locally rather than editing through the GitHub UI to avoid synchronization conflicts.

Before requesting review, rebase your branch on the latest changes from the base branch. Once review has started, avoid rebasing unless necessary so reviewers can follow the existing review history.

Reviewers commonly request:

- Wording, terminology or formatting changes
- Consistency with existing patterns
- Theme compatibility, tested in both light and dark modes

Wrap up the review

Once all suggestions are addressed, maintainers will approve and merge the PR.

After approval, **do not** force-push to your branch.

Once the PR is merged, your work is complete.

Get help

Open source contribution can be challenging. Even experienced developers encounter uncertainty when working on unfamiliar areas.

If you are unsure about a change or need guidance, ask the issue creator or a maintainer in the GitHub issue[256].

[256] https://github.com/canonical/sphinx-stack/issues

Contribute to documentation

This guide describes how to contribute changes to the Sphinx Stack documentation[257], including fixing issues, improving existing content and adding new guides.

Contributing requires familiarity with Git, GitHub and basic terminal commands.

Before starting, review the *contribution requirements* (page 153).

Set up your work environment

Create a personal fork[258] of the repository on GitHub.

Next, clone your fork onto your local machine:

With SSH

If you authenticate your GitHub account with SSH[259], run:

```
git clone git@github.com:<username>/sphinx-stack-docs
cd sphinx-stack-docs
git remote add upstream git@github.com:canonical/sphinx-stack-docs
git fetch upstream
```

With HTTPS

If you don't authenticate with SSH, clone with HTTPS[260] instead:

```
git clone https://github.com/<username>/sphinx-stack-docs
cd sphinx-stack-docs
git remote add upstream https://github.com/canonical/sphinx-stack-docs
git fetch upstream
```

Inside the project directory, install the dependencies and verify the build:

```
cd docs
make install
make html
```

If these commands complete without error, your environment is ready.

[257] https://github.com/canonical/sphinx-stack-docs
[258] https://github.com/canonical/sphinx-stack-docs/fork
[259] https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account
[260] https://docs.github.com/en/get-started/git-basics/about-remote-repositories#cloning-with-https-urls

Plan your contribution

Before making changes, consider whether the content already exists, where the change fits in the documentation structure and whether it follows the project's documentation patterns.

Significant work should be tied to an existing GitHub issue[261]. Before you start, comment on the issue to have it assigned to you.

Simple tasks

For simple changes that are small and self-contained, like fixing spelling, grammar, or punctuation, check the existing GitHub issues[262]. If none exists, open one[263] and state your interest in working on it.

Complex tasks

For complex changes, like a page rewrite or a new page, describe your proposal in the issue thread, including the page's Diátaxis[264] category and structure.

Make your changes

Create a branch

Before you begin your work, sync your local copy of the code and create a new branch:

```
git fetch upstream
git checkout -b <new-branch-name>
```

Format your branch name as `<ticket-id>-<description>`. For example, if you're working on GitHub issue #235, and it's about adding a text sanitizer, you'd name your branch `issue-235-add-text-sanitizer`. Keep the name under 80 characters.

Write documentation

All documentation lives in the `docs` directory. The directory structure, file names and `index.rst` files define how content is organized and displayed.

When adding or updating content:

- Place content in the appropriate Diátaxis category
- Follow existing terminology, style and formatting patterns
- Add labels and cross-references where appropriate
- Avoid duplicating information that already exists elsewhere

[261] https://github.com/canonical/sphinx-stack-docs/issues
[262] https://github.com/canonical/sphinx-stack-docs/issues
[263] https://github.com/canonical/sphinx-stack-docs/issues/new
[264] https://diataxis.fr

For small changes, update existing pages. For new features or major changes, add new content where users need to discover or reference it.

If you're adding a page, rewriting a page or making changes across multiple files, document the change in the *Documentation* section of the release notes.

Test your changes

When you're ready to check your draft, build the documentation locally before submitting:

```
cd docs
make html
```

Then check your document sources for problems:

```
make spelling      # Check spelling
make linkcheck     # Validate links
make woke          # Check inclusive language
make lint-md       # Check Markdown style
make vale          # Check style guide compliance (optional)
```

Preview your changes in a web browser. Make sure the elements you've changed render as expected. Double-check nested elements, such as content inside tabs and admonitions.

To preview locally with live reload at `http://127.0.0.1:8000`, run:

```
make run
```

Commit a change

Register the changes to your branch with a Git commit:

```
git add -A
git commit
```

Use the Conventional Commits[265] format. Most documentation changes should use the *docs* commit type.

```
docs: add initial setup guide
```

Keep the message short so other contributors and the project maintainers can see the gist of what you did.

Commit early and often. It's normal to make multiple commits for a single piece of work, especially when you come back to review it later.

Use separate commits for unrelated changes or different components.

[265] https://www.conventionalcommits.org/en/v1.0.0/

Sign your commits

All commits must be cryptographically signed. You can sign a commit by adding `-S` to the `git commit` command:

```
git commit -S -m "docs: add initial setup guide"
```

Signed commits display a "Verified" badge in GitHub. To configure Git to sign commits automatically, run:

```
git config --global commit.gpgsign true
```

For more information, see GitHub's documentation on signing commits[266].

Submit your contribution

Send for review

When you've committed your draft and you're ready to have it reviewed, push it to your fork:

```
git push -u origin <branch-name>
```

Open a pull request (PR) on GitHub. Give the PR a short, descriptive title using the Conventional Commits[267] format. For single-commit branches, GitHub may generate this automatically from the commit message.

Your PR should include the following details:

- **Title**: Short, descriptive summary
- **Description**: Problem solved, features added or bugs fixed
- **Relevant issues**: Link related issues and PRs[268]

PRs are typically reviewed within a week.

Address quality concerns

Before the PR is merged, it must pass all automatic checks. If there are any issues that your local testing didn't catch, the automatic checks will fail.

To address these issues, review the logs in the failed checks. The error messages will have remedies and hints for what needs fixing.

[266] https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits
[267] https://www.conventionalcommits.org/en/v1.0.0/
[268] https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/autolinked-references-and-urls

Address review feedback

When the maintainers review the PR, they may suggest improvements or request changes. Address them in follow-up commits to your branch, the same way you committed and pushed changes while drafting. Commit locally rather than editing through the GitHub UI to avoid synchronization conflicts.

Before requesting review, rebase your branch on the latest changes from the base branch. Once review has started, avoid rebasing unless necessary so reviewers can follow the existing review history.

Reviewers commonly request:

- Wording, terminology or formatting changes
- Consistency with existing patterns
- Proper reST or MyST markup
- Cross-references that use proper reST or MyST syntax
- Minimal examples before listing options, with verification steps

Wrap up the review

Once all suggestions are addressed, the maintainers will approve and merge the PR.

After approval, **do not** force-push to your branch, so the maintainers can clearly see any additional changes.

Once the PR is merged, your work is complete.

Get help

Open source contribution can be difficult. Even the most experienced writers encounter uncertainty when working on unfamiliar areas.

If you are unsure about a change or need guidance, ask the issue creator or a maintainer in the GitHub issue[269].

[269] https://github.com/canonical/sphinx-stack-docs/issues