Canonical

Documentation starter pack

Canonical

Contents

1 Quickstart guide **3**

2 Initial setup **5**
- 2.1 Enable the starter pack . . . 5
- 2.2 Customise the setup . . . 7
- 2.3 Integrate the Makefiles . . . 10
- 2.4 Set up Read the Docs . . . 10

3 Update the documentation **12**
- 3.1 Install prerequisites . . . 12
- 3.2 Build and preview . . . 13
- 3.3 Get guidance . . . 15

4 Automatic checks **16**
- 4.1 Accessibility check . . . 16
- 4.2 Inclusive language check . . . 17
- 4.3 Link check . . . 18
- 4.4 Spelling check . . . 18
- 4.5 Style guide linting . . . 19
- 4.6 Install prerequisite software . . . 20

5 Starter pack examples **21**

The documentation starter pack helps you to quickly set up, build, and publish documentation with Sphinx.

It contains common styling and configuration through the Canonical Sphinx[1] extension, supports both reST (reStructuredText) and Markdown, and includes automatic documentation checks.

The *Quickstart guide* (page 3) gives you a basic idea of how to start. Detailed information is available for *setting up your documentation* (page 5) and for *updating it* (page 12).

You can also find detailed information about the *Automatic checks* (page 16) included in the starter pack, and a *list of projects* (page 21) that use the starter pack.

[1] https://github.com/canonical/canonical-sphinx

1. Quickstart guide

The following steps will guide you through setting up your documentation using the starter pack and building an initial set of documentation.

For more detailed information, see the other sections of the starter pack documentation.

1. Download the `init.sh`[2] file from the starter pack repository and place it into the directory where you want to set up your documentation.
2. Run the script and specify where you want the files for the documentation framework to be placed (either in the current directory or in a sub-directory). Your own documentation files will need to be placed in the same directory (or sub-directories of it) later.

   See *Enable the starter pack* (page 5) for detailed information on what the script does.
3. Enter the documentation folder (the folder you specified when running the script) and build the documentation with the following command:

   ```
   make run
   ```

   This command creates a virtual environment, installs the Python dependencies, builds the documentation, and serves it on `http://127.0.0.1:8000/`.

   See *Build and preview* (page 13) for detailed information on different commands for building and viewing the documentation.
4. Keep this session running to rebuild the documentation automatically whenever a file is saved, and open `http://127.0.0.1:8000/` in a web browser to see the locally built and hosted HTML.
5. To add a new page to the documentation, create a new document called `reference.rst`, insert the following reST-formatted `Reference` heading, and then save the file:

   ```
   Reference
   =========
   ```

   Note

   This Quickstart guide uses reST, but if you prefer to use Markdown, you can create a `reference.md` file with the following content instead:

   ```
   # Reference
   ```
6. Open `index.rst`.

   At the bottom of this file, add a 3-space-indented line containing `Reference <reference>` to the end of the `toctree` section, and then save the file. This is the navigation title for the new document and its filename without the extension.

   The `toctree` section will now look like this:

[2] https://raw.githubusercontent.com/canonical/sphinx-docs-starter-pack/use-canonical-sphinx-extension/init.sh

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

   self
   Reference <reference>
```

> **Note**
>
> You can leave out the navigation title to use the document title instead. This means that in this example, you could also just type `reference` instead of `Reference <reference>`.

7. Check `http://127.0.0.1:8000/`.

   The documentation will now show **Reference** added to the navigation, and selecting the link in the navigation will open the new `reference.rst` document.

See *Get guidance* (page 15) for links to more detailed information about reST and Markdown/MyST.

2. Initial setup

This section is intended for repository administrators. It shows how to initially set up your documentation using the starter pack. After this is done, documentation contributors can follow the instructions in *Update the documentation* (page 12).

Important

After setting up your repository with the starter pack, you should track the changes made to the starter pack.

Changes to the look and feel, as well as common functionality, will be automatically available through updates to the Canonical Sphinx[3] extension.

Changes to files that are part of the starter pack, for example, *Automatic checks* (page 16), might require you to manually update your repository with the required files. See the starter pack's change log[4] for the most relevant (and of course all breaking) changes.

2.1. Enable the starter pack

You can use the starter pack to set up your documentation in one of the following ways:

- As a standalone documentation project in a new repository
- In a dedicated documentation folder in an existing code repository

The starter pack provides a shell script that updates and copies the starter pack files to the selected location.

Note

The recommended way of setting up the starter pack is to use the initialisation script. This method is automatically tested for the two scenarios mentioned above.

However, if you prefer to copy the files manually instead of running the script, this is also possible. Make sure to check the comments in the script in this case, because you will need to update the configuration in several files (in addition to copying the files to the correct locations). Also see *Behind the scenes* (page 6).

[3] https://github.com/canonical/canonical-sphinx
[4] https://github.com/canonical/sphinx-docs-starter-pack/wiki/Change-log

2.1.1. Run the initialisation script

The initialisation script will download all required files, update them, and copy them to the selected location.

1. Download `init.sh` from the starter pack repository: `https://raw.githubusercontent.com/canonical/sphinx-docs-starter-pack/use-canonical-sphinx-extension/init.sh`
2. Put the script into the root folder of the Git repository in which you want to enable the starter pack.

   This can be a new, empty repository or your code repository.
3. Make the script executable:

   ```
   chmod u+x init.sh
   ```
4. Run the script:

   ```
   ./init.sh
   ```
5. When prompted, enter the installation directory:
   - For standalone repositories, enter a full stop (`.`).
   - For documentation in code repositories, enter the name of the documentation folder (for example, `docs`).

     If the folder does not yet exist, it is created. If the folder exists already, some of its files might be overwritten by starter pack files, so make sure you have a backup.
6. Check the output for any warnings or errors.

You can now delete the `init.sh` file.

2.1.2. Behind the scenes

We recommend using the initialisation script to enable the starter pack. Here's a summary of what it does:

1. Prompt for the directory that will contain the documentation.
2. Clone the starter pack repository into a temporary directory.
3. Update several of the configuration files to adapt paths to the documentation directory.
4. Create the documentation directory.
5. Copy the starter pack files into the documentation directory.
6. Move some required files into the root folder of the repository (the GitHub workflow file and a `.wokeignore` file), unless they exist already.
7. Remove the temporary directory.

2.2. Customise the setup

The starter pack is configured in a way that makes sense for most projects. However, you must customise some settings for your project, like the project name.

In addition, there are some settings that you can customise if you wish so. And of course, you can add your own configuration as well.

2.2.1. Required customisation

You must check and update some of the configuration to adapt the documentation to your project.

Update the project information

Edit the `conf.py` file and update the configuration in the `Project information` section. See the comments in the file for more information about each setting.

You can adapt settings in the rest of the file as well, but this isn't required.

Open Graph configuration

When you post a link to your documentation somewhere (for example, on Mattermost or Discourse), it might be shown with a preview. This preview is configured through Open Graph[5].

If you don't know yet where your documentation will be hosted, you can leave the URL empty. If you do, specify the hosting URL. You can leave the defaults for the website name and the preview image or specify your own.

Configure the header

By default, the header contains your product tag, product name (taken from the `project` setting in the `conf.py` file), a link to your product page, and a drop-down menu for "More resources".

In many cases, this default setup is sufficient, but you should always check it.

You can change any of those links or add further links to the "More resources" drop-down by editing the `.sphinx/_templates/header.html` file. For example, you might want to add links to announcements, tutorials, getting started guides, or videos that are not part of the documentation.

[5] https://ogp.me/

2.2.2. Optional customisation

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

Deactivate the feedback button

By default, the starter pack includes a feedback button at the top of each page in the documentation. This button redirects users to your GitHub issues page, and populates an issue for them with details of the page they were on when they clicked the button.

If your project does not use GitHub issues, set the `github_issues` variable in the `conf.py` file to an empty value to disable both the feedback button and the issue link in the footer. If you want to deactivate only the feedback button, but keep the link in the footer, set `disable_feedback_button` in the `conf.py` file to `True`.

Configure the contributor display

By default, the starter pack will display a list of contributors at the bottom of each page. This requires the GitHub URL and folder to be configured.

If you want to turn this contributor listing off, you can do so by setting the `display_contributors` variable in the `conf.py` file to `False`.

To configure that only recent contributors are displayed, you can set the `display_contributors_since` variable. It takes any Linux date format (for example, a full date, or an expression like "3 months").

Add redirects

If you rename a source file, its URL will change. To prevent broken links, you should add a redirect from the old URL to the new URL in this case.

You can add redirects in the `redirects` variable in the `conf.py` file.

Configure included extensions

The starter pack includes a set of extensions that are useful for all documentation sets. They are pre-configured as needed, but you can customise their configuration in the `conf.py` file.

The following extensions are always included:

- `myst_parser` [6]
- `sphinxcontrib.jquery` [7]

The following extensions are included with the `[full]` optional install of canonical-sphinx:

- `sphinx-design` [8]

[6] https://myst-parser.readthedocs.io/en/latest/
[7] https://github.com/sphinx-contrib/jquery/
[8] https://sphinx-design.readthedocs.io/en/latest/

- `sphinx_copybutton` [9]
- `sphinx_tabs.tabs` [10]
- `sphinx_reredirects` [11]
- `sphinxext.opengraph` [12]
- `canonical-sphinx-extensions` [13] (`youtube-links`, `related-links`, `custom-rst-roles`, and `terminal-output`)
- `notfound.extension` [14]

None of the extensions referenced in this section need to be added to the `extensions` variable in `conf.py`. If the extensions need specific Python packages, add those to the `requirements.txt` file.

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 `conf.py` file. 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[15] 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 the `conf.py` file for more information.

Another example for page-specific configuration is the `hide-toc` field (provided by Furo[16]), which can be used to hide the page-internal table of content. See Hiding Contents sidebar[17].

[9] https://sphinx-copybutton.readthedocs.io/en/latest/
[10] https://sphinx-tabs.readthedocs.io/en/latest/
[11] https://documatt.gitlab.io/sphinx-reredirects/
[12] https://sphinxext-opengraph.readthedocs.io/en/latest/
[13] https://github.com/canonical/canonical-sphinx-extensions
[14] https://sphinx-notfound-page.readthedocs.io/en/latest/
[15] https://www.sphinx-doc.org/en/master/usage/restructuredtext/field-lists.html
[16] https://pradyunsg.me/furo/quickstart/
[17] https://pradyunsg.me/furo/customisation/toc/

2.2.3. Add your own configuration

To add custom configuration for your project, see the `Additions to default configuration` and `Additional configuration` sections in the `conf.py` file. These can be used to extend or override the common configuration, or to define additional configuration that is not covered by the common `conf.py` file.

The following links can help you with additional configuration:

- Sphinx configuration[18]
- Sphinx extensions[19]
- Furo documentation[20] (Furo is the Sphinx theme we use as our base)

If you need additional Python packages for any custom processing you do in your documentation, add them to the `.sphinx/requirements.txt` file.

2.3. Integrate the Makefiles

The starter pack contains two Makefiles: `Makefile` and `Makefile.sp`.

`Makefile.sp` implements the targets provided by the starter pack. You should keep it up-to-date with recent changes to the starter pack; therefore, avoid doing updates to the file. (If you need updates, consider contributing them to the starter pack!)

You can use `Makefile` to add custom targets or different target names.

If you want to integrate the starter pack targets into the main Makefile of your project, you can do so with a command similar to the following:

```
doc-%:
	cd docs && $(MAKE) -f Makefile.sp sp-$* ALLFILES='*.md **/*.md'
```

This example will create targets prefixed with `doc-` (for example, `doc-html` and `doc-serve`). When calling these targets, they switch to the documentation folder (`docs` in this case) and run the corresponding `sp-*` targets from `Makefile.sp`. In addition, the `ALLFILES` variable is overridden with a different set of files (this is needed for the *Inclusive language check* (page 17)).

2.4. Set up Read the Docs

For Canonical-specific information on how to set up your documentation on Read the Docs, see the Read the Docs at Canonical[21] and How to publish documentation on Read the Docs[22] guides.

In general, after enabling the starter pack for your documentation, follow these steps to build and publish your documentation on Read the Docs:

1. Make sure your documentation *builds without errors or warnings* (page 13).
2. Log into Read the Docs.

[18] https://www.sphinx-doc.org/en/master/usage/configuration.html
[19] https://www.sphinx-doc.org/en/master/usage/extensions/index.html
[20] https://pradyunsg.me/furo/quickstart/
[21] https://library.canonical.com/documentation/read-the-docs
[22] https://library.canonical.com/documentation/publish-on-read-the-docs

3. Use the manual import[23] to create a project.
4. If your documentation is not on the root level but in a documentation folder, you must specify the path to the `.readthedocs.yaml` file for your build. You do this by navigating to *Admin* > *Settings* and specifying the path under "Path for `.readthedocs.yaml`".

After this initial setup, your documentation should build successfully. If you get any errors, check the build log for indications on what the problem is.

2.4.1. Configure the webhook

If you have administrator privileges for the GitHub repository that you are adding, the integration webhook (which is responsible for automatically building the documentation when the repository changes) is created automatically.

If you don't have administrator privileges, the webhook must be set up by someone who does. See How to manually configure a Git repository integration[24] if you want to set it up manually.

2.4.2. Make your documentation public

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

To make the documentation public, you must configure the privacy level for each version of the documentation separately. You can do this by navigating to the *Versions* tab and changing the *Privacy Level* for each version.

2.4.3. Enable PR previews

To make Read the Docs automatically build your documentation when a pull request is opened or updated on GitHub, enable PR reviews for your project.

To do so, navigate to *Admin* > *Settings* and select *Build pull requests for this project*.

Read the Docs will then automatically build the documentation for each pull request, and the link to the output will be available as one of the checks in the pull request.

[23] https://readthedocs.com/dashboard/import/manual/
[24] https://docs.readthedocs.io/en/stable/guides/setup/git-repo-manual.html

3. Update the documentation

This section is intended for documentation contributors and covers how to work with the documentation after the repository has been set up with the starter pack as described in *Enable the starter pack* (page 5).

You'll find information on how to build and preview the documentation, and some pointers on what you need to know about the documentation framework and where to get help with it.

3.1. Install prerequisites

The documentation framework that the starter pack 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.

3.1.1. Install prerequisite software

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
```

3.1.2. Python environment

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

If you want to install them manually, you can run the following command:

```
make install
```

This command creates a virtual environment (`.sphinx/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:

```
make clean
```

Note

- By default, the starter pack 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`

3.2. Build and preview

The starter pack provides make commands to build and view the documentation. All these commands will automatically set up the Python environment if it isn't set up yet.

3.2.1. 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 `_build/` directory in your documentation 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 *Run a clean build* (page 13).

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

3.2.2. Run a clean build

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

```
make clean-doc html
```

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

```
make clean html
```

3.2.3. View the documentation

To view the documentation output, run the following command:

```
make serve
```

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

3.2.4. Live view

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 on `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.

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 a clean build* (page 13) before finalising your changes.

3.2.5. 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.

3.3. Get guidance

The starter pack uses Sphinx[25] as the documentation framework. It supports markup in both reStructuredText[26] and Markdown[27] with MyST[28].

It's outside of the scope of the starter pack to teach you how to use these tools to create documentation, but you can check the *Quickstart guide* (page 3) for a very brief introduction. For more detailed information and syntax guides, see the linked documents.

To make it easier for you to get started, and to keep our documentation consistent, the following style guides give recommendations and conventions for using reST and Markdown/MyST:

- reStructuredText style guide[29]
- MyST style guide[30]

The starter pack also contains cheat sheets for both markup languages that allow to easily copy and paste the markup that you want. See the `doc-cheat-sheet.rst` and `doc-cheat-sheet-myst.md` files.

3.3.1. Other resources

Canonical documentation uses Diátaxis[31].

The following documentation repository contains an example for how to implement this structure:

- Example product documentation[32]
- Example product documentation repository[33]

[25] https://www.sphinx-doc.org/
[26] https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html
[27] https://commonmark.org/
[28] https://myst-parser.readthedocs.io/
[29] https://canonical-documentation-with-sphinx-and-readthedocscom.readthedocs-hosted.com/style-guide/
[30] https://canonical-documentation-with-sphinx-and-readthedocscom.readthedocs-hosted.com/style-guide-myst/
[31] https://diataxis.fr/
[32] https://canonical-example-product-documentation.readthedocs-hosted.com/
[33] https://github.com/canonical/example-product-documentation

4. Automatic checks

The starter pack comes with several automatic checks that you can (and should!) run on your documentation before committing and pushing changes.

The following checks are available:

4.1. Accessibility check

The accessibility check uses Pa11y[34] to check for accessibility issues in the documentation output.

It is configured to use the Web Content Accessibility Guidelines (WCAG) 2.2[35], requiring Level AA conformance[36].

> **Note**
>
> The accessibility check is not yet required, since there are some issues within the starter pack itself. We are working on resolving those.
>
> For now, the check is implemented as non-blocking as part of our GitHub workflow.

4.1.1. Install prerequisite software

`Pa11y` must be installed through `npm`:

```
sudo apt install npm
```

To install `Pa11y`:

```
make pa11y-install
```

4.1.2. Run the accessibility check

Look for accessibility issues in rendered documentation:

```
make pa11y
```

4.1.3. Configure the accessibility check

The `pa11y.json` file in the `.sphinx` directory provides basic defaults.

To browse the available settings and options, see `Pa11y`'s README[37] on GitHub.

[34] https://pa11y.org/
[35] https://www.w3.org/TR/WCAG22/
[36] https://www.w3.org/WAI/WCAG2AA-Conformance
[37] https://github.com/pa11y/pa11y#command-line-configuration

4.2. Inclusive language check

The inclusive language check uses woke[38] to check for violations of inclusive language guidelines.

4.2.1. Install prerequisite software

To install `woke`, you need `snap`:

```
sudo apt install snapd
```

To install `woke`:

```
make woke-install
```

4.2.2. Run the inclusive language check

Ensure the documentation uses inclusive language:

```
make woke
```

4.2.3. Configure the inclusive language check

By default, the inclusive language check is applied only to reST files located under the documentation directory (usually `docs`). To check Markdown files, for example, or to use a location other than the `docs` sub-tree, you must override the `ALLFILES` variable in `Makefile.sp` (see *Integrate the Makefiles* (page 10)).

You can find more information about available options in the woke User Guide[39].

4.2.4. Inclusive language check exemptions

Sometimes, you might need to use some non-inclusive words. In such cases, create check exemptions for them.

See the woke documentation[40] for how to do this. The following sections provide some examples.

Exempt a word

To exempt an individual word, place a custom `none` role (defined in the `canonical-sphinx-extensions` Sphinx extension) anywhere on the line containing the word in question. The role syntax is:

```
:none:`wokeignore:rule=<SOME_WORD>,`
```

For instance:

```
This is your text. The word in question is here: whitelist. More text.
:none:`wokeignore:rule=whitelist,`
```

[38] https://github.com/get-woke/woke
[39] https://docs.getwoke.tech/usage/#file-globs
[40] https://docs.getwoke.tech/ignore

To exempt an element of a URL, use the standard reST method of placing links at the bottom of the page (or in a separate file) and place a comment line immediately above the URL line. The comment syntax is:

```
.. wokeignore:rule=<SOME_WORD>
```

Here is an example where a URL element contains the string “master”:

```
.. LINKS
.. wokeignore:rule=master
.. _link definition: https://some-external-site.io/master/some-page.html
```

You can now refer to the label `link definition_` in the body of the text.

Exempt an entire file

A more drastic solution is to make an exemption for the contents of an entire file. For example, to exempt file `docs/foo/bar.rst`, add the following line to the file `.wokeignore`:

```
foo/bar.rst
```

4.3. Link check

The link check uses Sphinx to access the links in the documentation output and validate whether they are working.

4.3.1. Run the link check

Validate links within the documentation:

```
make linkcheck
```

4.3.2. Configure the link check

If you have links in the documentation that you don’t want to be checked (for example, because they are local links or give random errors even though they work), you can add them to the `linkcheck_ignore` variable in the `conf.py` file.

4.4. Spelling check

The spelling check uses `pyspelling` to check the spelling in your documentation. It ignores code (both code blocks and inline code) and URLs (but it does check the link text).

4.4.1. Run the spelling check

Ensure there are no spelling errors in the documentation:

```
make spelling
```

If you only want to check the existing output and do not want to build the HTML again, run the spelling check separately:

```
make spellcheck
```

4.4.2. Configure the spelling check

The spelling check uses `pyspelling`, which in turn relies on `aspell`. Its configuration is located in the `.sphinx/spellingcheck.yaml` file.

The starter pack includes a common list of words that should be excluded from the check (`.sphinx/.wordlist.txt`). You shouldn't edit this file, because it is maintained and updated centrally and contains words that apply across all projects. To add custom exceptions for your project, add them to the `.custom_wordlist.txt` file.

If you need to add systematic exceptions for specific HTML tags or CSS classes (for example, all image captions or H2 headings), you can do this in the `.sphinx/spellingcheck.yaml` file. You can configure which HTML elements to exclude under `pyspelling.filters.html`.

4.4.3. Exclude specific terms

Sometimes, you need to use a term in a specific context that should usually fail the spelling check. (For example, you might need to refer to a product called `ABC Docs`, but you do not want to add `docs` to the word list because it isn't a valid word.)

In this case, you can use the `:spellexception:` role. See More useful markup[41] in the reST style guide (also available in MyST).

4.5. Style guide linting

The starter pack includes a method to run the Vale[42] documentation linter configured with the Vale rules for the current style guide[43].

4.5.1. Run the style guide linting

Check documentation with Vale:

```
make vale
```

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

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

> **Note**
>
> Running Vale against a directory will also run against subfolders.

You can use wildcards to run against all files matching a string, or an extension.

For example, to run against all `.md` files within a folder:

[41] https://canonical-documentation-with-sphinx-and-readthedocscom.readthedocs-hosted.com/style-guide/#more-useful-markup
[42] https://vale.sh/
[43] https://github.com/canonical/praecepta

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

To match, for example, `doc_1.md` and `doc_2.md`:

```
make vale TARGET=doc*
```

4.5.2. Exempt paragraphs

To disable Vale linting within individual files, specific markup can be used.

For Markdown:

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

This text will be ignored by Vale.

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

For reST:

```
.. vale off

This text will be ignored by Vale.

.. vale on
```

4.6. Install prerequisite software

Some of the tools used by the automatic checks might not be available by default on your system. To install them, you need `snap` and `npm`:

```
sudo apt install npm snapd
```

To install the validation tools:

```
make woke-install
make pa11y-install
```

5. Starter pack examples

The following documentation sets use the starter pack and can serve as implementation examples.

If you are using the starter pack, feel free to open a PR against this page and add your project!

Documentation	Using the extension?
LXD[44]	No
MicroCloud[45]	No

[44] https://documentation.ubuntu.com/lxd/en/latest/
[45] https://canonical-microcloud.readthedocs-hosted.com/en/latest/