Sphinx and Read the Docs

Canonical Group Ltd

Apr 29, 2024

CONTENTS

The guide for setting up your documentation with Sphinx and Read the Docs has moved to the Canonical Reference Library:

- Read the Docs at Canonical
- How to publish documentation on Read the Docs

Also see the following information:

- Example product documentation and Example product documentation repository
- Sphinx documentation starter pack documentation and Sphinx documentation starter pack repository

CHAPTER

ONE

STYLE GUIDES

1.1 reStructuredText style guide

The documentation files use reStructuredText (reST) syntax.

See the following sections for syntax help and conventions.

Note: This style guide assumes that you are using the Sphinx documentation starter pack. Some of the mentioned syntax requires Sphinx extensions (which are enabled in the starter pack).

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

1.1.1 Headings

Input	Description

`Title`
`=====`
	Page title and H1 heading

`Heading` `-------`	H2 heading
`Heading` `~~~~~~~`	H3 heading
`Heading` `^^^^^^^`	H4 heading

`Heading`
`.......`
	H5 heading

Underlines must be the same length as the title or heading.

Adhere to the following conventions:

- Do not use consecutive headings without intervening text.
- Be consistent with the characters you use for each level. Use the ones specified above.
- Do not skip levels (for example, do not follow an H2 heading with an H4 heading).
- Use sentence style for headings (capitalise only the first word).

1.1.2 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**

Adhere to the following conventions:

- Use italics sparingly. Common uses for italics are titles and names (for example, when referring to a section title that you cannot link to, or when introducing the name for a concept).
- Use bold sparingly. Avoid using bold for emphasis and rather rewrite the sentence to get your point across.

1.1.3 Code blocks

To start a code block, either end the introductory paragraph with two colons (`::`) and indent the following code block, or explicitly start a code block with `.. code::`. In both cases, the code block must be surrounded by empty lines.

When explicitly starting a code block, you can specify the code language to enforce a specific lexer, but in many cases, the default lexer works just fine.

For a list of supported languages and their respective lexers, see the official Pygments documentation.

Input	Output

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

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

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

Terminal output

Showing a terminal view can be useful to show the output of a specific command or series of commands, where it is important to see the difference between input and output. In addition, including a terminal view can help break up a long text and make it easier to consume, which is especially useful when documenting command-line-only products.

To include a terminal view, use the following directive:

Input	Output

`.. terminal::`
`   :input: command number one`
`   :user: root`
`   :host: vm`
 	
`   output line one`
`   output line two`
`   :input: another command`
`   more output`
	`root@vm:~# command number one   output line`
	`oneoutput line two       root@vm:~# another`
	`command more output`

Input is specified as the `:input:` option (or prefixed with `:input:` as part of the main content of the directive). Output is the main content of the directive.

To override the prompt (`user@host:~$` by default), specify the `:user:` and/or `:host:` options. To make the terminal scroll horizontally instead of wrapping long lines, add `:scroll:`.

1.1.4 Links

Link markup depends on whether you need an external URL or a page in the same documentation set.

External links

For external links, use one of the following methods.

Link inline:
Define occasional links directly within the surrounding text. To make the link text show up in code-style (which excludes it from the spelling check), use the `:literalref:` role.

Input	Output
`` `Canonical website <https://canonical.com/>`_ ``	Canonical website
`` :literalref:`ubuntu.com` ``	`ubuntu.com`
`` :literalref:`xyzcommand <https://example.com>` ``	`xyzcommand`

You can also use a URL as is (`https://example.com`), but that might cause spellchecker errors.

Tip: To prevent a URL from appearing as a link, add an escaped space character (`https:\ //`). The space won't be rendered:

Input	Output
`https:\ //canonical.com/`	

Define the links at the bottom of the page:
To keep the text readable, group the link definitions below.

Input	Output	Description
`` `Canonical website`_ ``	Canonical website	Using the below defined link

`.. LINKS`
`.. _Canonical website:␣`
`↪https://canonical.com/`
	n/a	Defining links at the bottom

Define the links in a shared file:
To keep the text readable and links maintainable, put all link definitions in a file named `reuse/links.txt` to include it in a custom `rst_epilog` directive (see the Sphinx documentation).

Listing 1:

```
custom_rst_epilog = """
    .. include:: reuse/links.txt
    """
```

Input	Output
`` `Canonical website`_ ``	Canonical website

Related 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 `&#32;`, and include the value in quotes if Sphinx complains about the metadata value because it starts with `[`.

To add a link to a Discourse topic, configure the Discourse instance in the `custom_conf.py` file. Then add the following field at the top of the page (where `12345` is the ID of the Discourse topic):

```
:discourse: 12345
```

YouTube links

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

You can reference pages and targets in this documentation set, and also in other documentation sets using Intersphinx.

Referencing a section

To reference a section within the documentation (either on the same page or on another page), add a target to that section and reference that target.

You can add targets at any place in the documentation. However, if there is no heading or title for the targeted element, you must specify a link text.

Input	Output	Description
`.. _target_I`		Adds the target `target_ID`. **Note:** When defining the target, you must prefix it with an underscore. Do not use the starting underscore when referencing the target.
`:ref:`a_s`	*Referencing a section*	References a target that has a title.
`:ref:`Lin text <a_random`	*Link text*	References a target and specifies a title.
`:ref:`sta`	Sphin example	You can also reference targets in other doc sets.

Adhere to the following conventions:

- Never use external links to reference a section in the same doc set or a doc set that is linked with Intersphinx. It would likely cause a broken link in the future.
- Override the link text only when it is necessary. If you can use the referenced title as link text, do so, because the text will then update automatically if the title changes.
- Never "override" the link text with the same text that would be generated automatically.

Referencing a page

If a documentation page does not have a target, you can still reference it by using the `:doc:` role with the file name and path.

Input	Output
`:doc:`index``	*Sphinx and Read the Docs*
`:doc:`Link text <index>``	*Link text*
`:doc:`starter-pack:how-to/index``	How-to guides
`:doc:`Link text <starter-pack:how-to/index>``	Link text

Adhere to the following conventions:

- Only use the `:doc:` role when you cannot use the `:ref:` role, thus only if there is no target 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.
- Override the link text only when it is necessary. If you can use the document title as link text, do so, because the text will then update automatically if the title changes.
- Never "override" the link text with the same text that would be generated automatically.

1.1.5 Navigation

Every documentation page must be included as a sub-page to another page in the navigation.

This is achieved with the toctree 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 `custom_excludes` variable in the `custom_conf.py` file.

1.1.6 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

Output

```
1. Step 1

   - Item 1

     * Sub-item
   - Item 2

```

(continues on next page)

(continued from previous page)

```
     i. Sub-step 1
     #. Sub-step 2
#. Step 2

   a. Sub-step 1

      - Item
   #. Sub-step 2
```

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
    b. Sub-step 2

Adhere to the following conventions:

- 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

1.1.7 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 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 for reference.

```
+-----------------------+------------+
| Header 1              | Header 2   |
+=======================+============+
| Cell 1                | Cell 2     |
|                       |            |
| 2nd paragraph cell 1  |            |
+-----------------------+------------+
| Cell 3                | Cell 4     |
+-----------------------+------------+
```

List tables

See list tables for reference.

```
.. list-table::
   :header-rows: 1

   * - Header 1
     - Header 2
   * - Cell 1

       2nd paragraph cell 1
     - Cell 2
   * - Cell 3
     - Cell 4
```

1.1.8 Notes

Input	Output

`.. note::`
`   A note.`
	Note: A note.

`.. tip::`
`   A tip.`
	Tip: A tip.

`.. important::`
`   Important information`
	Important: Important information

`.. caution::`
`   This might damage your hardware!`
	Caution: This might damage your hardware!

Adhere to the following conventions:

- Use notes sparingly.
- Only use the following note types: `note`, `tip`, `important`, `caution`
- Only use a caution if there is a clear hazard of hardware damage or data loss.

1.1.9 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

Adhere to the following conventions:

- 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 for information about how to word the alt text.

1.1.10 Reuse

A big advantage of reST in comparison to plain Markdown is that it allows to reuse content.

Substitution

To reuse sentences and entire paragraphs that have little markup or special formatting, define substitutions for them in two possible ways.

Globally, in a file named `reuse/substitutions.txt` that is included in a custom `rst_epilog` directive (see the Sphinx documentation):

Listing 2:

```
custom_rst_epilog = """
    .. include:: reuse/substitutions.txt
    """
```

Listing 3:

```
.. |version_number| replace:: 0.1.0

.. |rest_text| replace:: *Multi-line* text
                         that uses basic **markup**.

.. |site_link| replace:: Website link
.. _site_link: https://example.com
```

Locally, putting the same directives in any reST file:

Listing 4:

```
.. |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

Tip: Use substitution names that hint at the included content (for example, `note_not_supported` instead of `note_substitution`).

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 required (if the same text occurs more than once). Using only `:start-line:` and `:end-line:` is error-prone though.

You cannot put any targets into the content that is being reused (because references to this target would be ambiguous then). You can, however, put a target 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.

Input	Output

`.. include:: index.rst`
`   :start-after: Also see the following information:`
`   :end-before: and `Sphinx documentation starter pack repository`_`
	• Example product documentation and Example product documentation repository
	• Sphinx documentation starter pack documentation

Adhere to the following conventions:

- Files that only contain text that is reused somewhere else should be placed in the `reuse` folder 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.

1.1.11 Tabs

The recommended way of creating tabs is to use the Sphinx tabs extension, which remembers the selected tab (also when navigating to other pages).

Input	Output

`.. tabs::`
 	
`   .. group-tab:: Tab 1`
 	
`      Content Tab 1`
 	
`   .. group-tab:: Tab 2`
 	
`      Content Tab 2`
	Tab 1
	Tab 2
	Content Tab 1
	Content Tab 2

Alternatively, if you use tabs only occasionally and don't want to include an additional extension for them, you can use the basic tabs that the Sphinx design extension provides.

Note: The Sphinx design tabs sync within a page, but if you navigate to another page, the selection is lost.

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

1.1.12 Glossary

You can define glossary terms in any file. Ideally, all terms should be collected in one glossary file though, and they can then be referenced from any file.

Input	Output

```
.. glossary::

   example term
     Definition of the example term.
```
	example term
	Definition of the example term.

`` :term:`example term` ``	*example term*

1.1.13 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:`PurposelyWrc`		Explicitly exempt a term from the spelling check.

1.2 MyST style guide

The documentation files use a mixture of Markdown and MyST syntax.

See the following sections for syntax help and conventions.

Note: This style guide assumes that you are using the Sphinx documentation starter pack. Some of the mentioned syntax requires Sphinx extensions (which are enabled in the starter pack).

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

1.2.1 Headings

Input	Description
`# Title`	Page title and H1 heading
`## Heading`	H2 heading
`### Heading`	H3 heading
`#### Heading`	H4 heading
...	Further headings

Adhere to the following conventions:

- Do not use consecutive headings without intervening text.
- Do not skip levels (for example, do not follow an H2 heading with an H4 heading).
- Use sentence style for headings (capitalise only the first word).

1.2.2 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**

Adhere to the following conventions:

- Use italics sparingly. Common uses for italics are titles and names (for example, when referring to a section title that you cannot link to, or when introducing the name for a concept).
- Use bold sparingly. Avoid using bold for emphasis and rather rewrite the sentence to get your point across.

1.2.3 Code blocks

Start and end a code block with three back ticks:

~~~
```
~~~

You can specify the code language after the back ticks to enforce a specific lexer, but in many cases, the default lexer works just fine.

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

Showing a terminal view can be useful to show the output of a specific command or series of commands, where it is important to see the difference between input and output. In addition, including a terminal view can help break up a long text and make it easier to consume, which is especially useful when documenting command-line-only products.

To show a terminal view, use the following directive:

Input

~~~~~
```{terminal}
:input: command number one
:user: root
:host: vm

output line one
output line two
:input: another command
more output
```
~~~~~

Output

~~~~~
root@vm:~# command number one   output line
oneoutput line two       root@vm:~# another
command more output
~~~~~

Input is specified as the `:input:` option (or prefixed with `:input:` as part of the main content of the directive). Output is the main content of the directive.

To override the prompt (`user@host:~$` by default), specify the `:user:` and/or `:host:` options. To make the terminal scroll horizontally instead of wrapping long lines, add `:scroll:`.

1.2.4 Links

How to link depends on if you are linking to an external URL or to another page in the documentation.

External links

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
`https://canonical.com`	
``[`https://canonical.com`](https://canonical.com)``	`https://canonical.com`

To display a URL as text and prevent it from being linked, add a `<span></span>`:

Input	Output
`https:/<span></span>/canonical.com`	

Related 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/canonical-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 `&#32;`, and include the value in quotes if Sphinx complains about the metadata value because it starts with `[`.

To add a link to a Discourse topic, configure the Discourse instance in the `custom_conf.py` file. Then add the following field at the top of the page (where `12345` is the ID of the Discourse topic):

```
discourse: 12345
```

YouTube links

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

For internal references, both Markdown and MyST syntax are supported. In most cases, you should use MyST syntax though, because it resolves the link text automatically and gives an indication of the link in GitHub rendering.

Referencing a section

To reference a section within the documentation (either on the same page or on another page), add a target to that section and reference that target.

You can add targets at any place in the documentation. However, if there is no heading or title for the targeted element, you must specify a link text.

Input	Output	Output on GitHub	Description
`(target_ID)=`		(target_ID)=	Adds the target `target_ID`.
`` {ref}`a_section_target_m ``	*Referencing a section*	{ref}`a_section_target_my`	References a target that has a title.
`` {ref}`link text <a_random_target_myst>` ``	*link text*	{ref}`link text <a_random_target_myst>`	References a target and specifies a title.
`` {ref}`starter-pack:home` ``	Sphinx example	{ref}`starter-pack:home`	You can also reference targets in other doc sets.
`` [`xyz`](a_random_target_ ``	*`xyz`*	*`xyz`* (link is broken)	Use Markdown syntax if you need markup on the link text.

Adhere to the following conventions:

- Never use external links to reference a section in the same doc set or a doc set that is linked with Intersphinx. It would likely cause a broken link in the future.
- Override the link text only when it is necessary. If you can use the section title as link text, do so, because the text will then update automatically if the title changes.
- Never "override" the link text with the same text that would be generated automatically.

Referencing a page

If a documentation page does not have a target, you can still reference it by using the `{doc}` role with the file name and path. Use MyST syntax to automatically extract the link text. When overriding the link text, use Markdown syntax.

Input	Output	Output on GitHub	Status
`` {doc}`index` ``	*Sphinx and Read the Docs*	{doc}`index`	Preferred.
`[](index)`	*Sphinx and Read the Docs*		Do not use.
`[Index page](index)`	*Index page*	*Index page*	Preferred when overriding the link text.
`` {doc}`Index page <index>` ``	*Index page*	{doc}`Index page <index>`	Alternative when overriding the link text.

Adhere to the following conventions:

- Only use the {doc} role when you cannot use the {ref} role, thus only if there is no target 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.
- Override the link text only when it is necessary. If you can use the document title as link text, do so, because the text will then update automatically if the title changes.
- Never "override" the link text with the same text that would be generated automatically.

1.2.5 Navigation

Every documentation page must be included as a sub-page to another page in the navigation.

This is achieved with the toctree 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 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 custom_excludes variable in the custom_conf.py file.

1.2.6 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

Adhere to the following conventions:

- 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

1.2.7 Tables

You can use standard Markdown tables. However, using the reST list table syntax is usually much easier. See the Sphinx documentation 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 for reference.

~~~
```{list-table}
   :header-rows: 1

* - Header 1
  - Header 2
* - Cell 1

    2nd paragraph cell 1
  - Cell 2
* - Cell 3
  - Cell 4
```
~~~

1.2.8 Notes

Input	Output
```` ```{note} ```` ```` A note. ```` ```` ``` ````	**Note:** A note.
```` ```{tip} ```` ```` A tip. ```` ```` ``` ````	**Tip:** A tip.
```` ```{important} ```` ```` Important information ```` ```` ``` ````	**Important:** Important information.
```` ```{caution} ```` ```` This might damage your hardware! ```` ```` ``` ````	**Caution:** This might dam-age your hard-ware!

Adhere to the following conventions:

- Use notes sparingly.
- Only use the following note types: `note`, `tip`, `important`, `caution`
- Only use a caution if there is a clear hazard of hardware damage or data loss.

1.2.9 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

Adhere to the following conventions:

- 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 for information about how to word the alt text.

1.2.10 Reuse

A big advantage of MyST in comparison to plain Markdown is that it allows to reuse content.

Substitution

To reuse sentences or paragraphs that have little markup and special formatting, use substitutions.

Substitutions can be defined in the following locations:

- Globally, in a file named `reuse/substitutions.yaml` that is loaded into the `myst_substitutions` variable in `custom_conf.py`:

  Listing 5:

  ```
  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())
  ```

  Listing 6:

  ```
  # 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
                         ```"

  ---
  ~~~

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
`{{advanced_reuse_key}}`	This is a substitution that includes a code block: `code block`

Adhere to the following convention:

- Substitutions do not work on GitHub. Therefore, 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 required (if the same text occurs more than once). Using only `:start-line:` and `:end-line:` is error-prone though.

You cannot put any targets into the content that is being reused (because references to this target would be ambiguous then). You can, however, put a target right before including the file.

By combining file inclusion and substitutions, you can even replace parts of the included text.

Input	Output
`% Include parts of the content from` `% file [style-guide.rst](style-guide.rst)` ` ```{include} style-guide.rst` ` :start-after: "Adhere to the following conventions:"` ` :end-before: "- Do not skip levels"` ` ``` `	• Do not use consecutive headings without intervening text. • Be consistent with the characters you use for each level. Use the ones specified above.

Adhere to the following convention:

- File inclusion does not work on GitHub. Therefore, 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` folder 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.

1.2.11 Tabs

The recommended way of creating tabs is to use the Sphinx tabs extension, which remembers the selected tab (also when navigating to other pages).

Input

~~~
````{tabs}

```{group-tab} Tab 1

Content Tab 1
```

```{group-tab} Tab 2

Content Tab 2
```

````
~~~

Output

Tab 1
Tab 2
Content Tab 1
Content Tab 2

Alternatively, if you use tabs only occasionally and don't want to include an additional extension for them, you can use the basic tabs that the Sphinx design extension provides.

Input

~~~
````{tab-set}

```{tab-item} Tab 1
:sync: key1

Content Tab 1
```

```{tab-item} Tab 2
:sync: key2

Content Tab 2
```

````
~~~

Output

Tab 1

Content Tab 1

Tab 2

Content Tab 2

1.2.12 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

1.2.13 Glossary

You can define glossary terms in any file. Ideally, all terms should be collected in one glossary file though, and 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*

1.2.14 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.

INDEX