

---

# reStructuredText Cheat Sheet

**Mats Gustafsson**

**Nov 11, 2025**



## GUIDES

1	reStructuredText cheat sheet	1
2	Indices and tables	9
	Index	11



## RESTRUCTUREDTEXT CHEAT SHEET

This section summarizes commonly used reStructuredText (RST) and Sphinx syntax for editing and maintaining documentation.

### 1.1 Common Conventions

- Use `code-block:: bash` for shell commands.
- Use `code-block:: python` for code examples.
- Inline code or filenames: `example.py`
- Bold and italics: **bold**, *italic*
- Lists: - Unordered: - or \* - Ordered: 1. 2. etc.
- Avoid overusing raw HTML; Sphinx supports nearly all formatting needs natively.

**Rule of thumb:** aim for 2–5 index entries per RST file, more only for very technical chapters (like `usage.rst`).

### 1.2 Headings

Our heading hierarchy follows this exact order:

- Top Level (e.g. document title) =
- Second Level -
- Third Level +
- Fourth Level #
- Fifth Level “

Always keep the underline **at least as long** as the heading text.

### 1.3 Cross-references

References connect sections or terms so they can be linked from elsewhere.

#### Internal section reference

1. Define a label before the heading:

```
.. _my-section-label:
```

```
My Section Title  
-----
```

2. Refer to it from anywhere using:

```
See :ref:`my-section-label` for details.
```

File reference

```
:doc:`../path/to/other_file`
```

Inline hyperlink

```
The `Sphinx documentation <https://www.sphinx-doc.org/>`_ is  
↵excellent.
```

## 1.4 Index entries

Use `.. index::` to add searchable keywords.

Examples:

```
.. index::  
   single: Docker; build container  
   pair: direction of arrival; DoA  
   triple: machine learning; classification; examples
```

Place index entries **immediately below** the relevant section title for accurate linking.

Index only what's useful

- Do not index every heading or parameter.
- Index **concepts people might look up later** such as tools, algorithms, file formats, configuration options, and important scripts.

Use specific keywords

- Prefer concrete terms: CFAR-DoA or Hailo-8L instead of code or setup.

Group related terms

- Use `;` to build hierarchy:

```
.. index::  
   single: audio; conversion  
   single: audio; resampling
```

Use pairs/triples for synonyms

- If a concept has multiple names, use `pair` or `triple` so it appears in all relevant places in the index:

```
.. index::  
   pair: direction of arrival; DoA
```

Index tools, not every option

- For commands or scripts, add one index entry for the tool.
- Avoid indexing every CLI flag—search handles that.

## 1.5 Tables

Use **grid tables** for clarity and column alignment.

Example:

```
+-----+-----+-----+
| Column A      | Column B      | Column C      |
+=====+=====+=====+
| Item 1        | Description 1  | Value 1       |
+-----+-----+-----+
| Item 2        | Description 2  | Value 2       |
+-----+-----+-----+
```

Tips:

- Align + and | vertically for clean rendering.
- Use = in the header separator row.
- Use monospace font (code-block) when showing RST examples.

## 1.6 Editing Box-Drawing Diagrams in Emacs

When editing Unicode box-drawing diagrams in **Emacs** or **Doom Emacs**, make sure your buffer is in UTF-8 encoding and that you use a monospace font for alignment.

### 1.6.1 Check or enforce UTF-8

Most modern Emacs configurations (including Doom) default to UTF-8, but you can verify or set it manually:

```
(setq-default buffer-file-coding-system 'utf-8-unix)
```

If a file opens in another encoding, fix it interactively:

```
C-x RET f utf-8 RET
```

### 1.6.2 Insert box-drawing characters

You can insert any Unicode character by name or code point:

```
C-x 8 RET BOX DRAWINGS LIGHT HORIZONTAL RET
C-x 8 RET 2500 RET
```

This inserts ─.

### 1.6.3 Common characters

Sym- bol	Code	Insert command
—	U+2500	U+2502 C-x 8 RET 2500 RET C-x 8 RET 2502 RET C-x 8 RET 250C
┌┐	U+250C	U+2510 RET C-x 8 RET 2510 RET C-x 8 RET 2514 RET C-x 8 RET 2518
└┘	U+2514	U+2518 RET C-x 8 RET 252C RET C-x 8 RET 2534 RET C-x 8 RET 253C
┐┌	U+252C	U+2534 RET
┘┘	U+253C	

Alternatively, run:

```
M-x insert-char RET box draw RET
```

and select from the interactive menu.

### 1.6.4 Quick editing tips

- Use **spaces** (not tabs) for alignment.
- Ensure you are in a **monospace** buffer. In Doom Emacs, you can toggle variable pitch fonts off with `SPC t v`.
- Copy and reuse template boxes instead of redrawing them each time.

### 1.6.5 Template box (copy and fill in)

```
Example process step
```

This makes it easy to maintain consistent visual flow diagrams directly inside your `.rst` files without any external graphics tools.

### 1.6.6 Reusable Box Templates (Yasnippet or Tempo)

If you frequently add diagrams, you can automate box creation using **Yasnippet** (bundled with Doom Emacs) or the built-in **tempo** snippet system.

### 1.6.7 Yasnippet setup

Yasnippet allows you to expand a keyword into a pre-defined text pattern.

1. Open your snippets folder (default for Doom):

```
~/doom.d/snippets/text-mode/
```

Create it if it doesn't exist.

2. Add a new snippet file, for example `box`:

```
# -*- mode: snippet -*-
# name: Box diagram template
# key: box
```

(continues on next page)



(continued from previous page)

```
# --
  ${1:Your process step}
$0
```

3. Reload snippets:

```
M-x yas-reload-all
```

4. Type **box** in any buffer and press **TAB** → a box appears with an editable field.

### 1.6.8 Tempo setup (alternative)

If you prefer Emacs' built-in **tempo templates**, add this to your Doom config:

```
(require 'tempo)

(tempo-define-template
 "box"
 '("
  " p " | \n"
  " \n")
 "box"
 "Insert a Unicode box diagram.")
```

Then restart Emacs or evaluate the code (**M-x eval-buffer**). Now you can type:

```
M-x tempo-template-box
```

and insert a ready-made box anywhere.

### 1.6.9 Customization tips

- You can create variants (e.g., **flowbox** or **doublebox**) with wider lines.
- Add these snippets to version control in your Doom config for easy sharing.
- Yasnippet placeholders like **\${1:Text}** support tab-cycling between editable fields.
- All snippets support UTF-8 box characters directly—no special input method required.

These small helpers make it easy to document process flows or architecture diagrams right from within Emacs while keeping the style consistent across *.rst* files.

## 1.7 Notes, Warnings, and Tips

Admonitions help highlight important notes:

```
.. note::
    This is a note.

.. warning::
```

(continues on next page)

(continued from previous page)

```
This is a warning.  
.. tip::  
    This is a tip.
```

They render with icons and borders in HTML output.

## 1.8 Images

Images are inserted with the `.. image::` or `.. figure::` directives.

Basic image:

```
.. image:: _static/example.png  
:alt: Example image  
:width: 400px  
:align: center
```

With caption (use ```figure```):

```
.. figure:: _static/diagram.png  
:alt: System architecture  
:width: 80%  
:align: center  
  
System architecture overview.
```

Notes:

- Images are usually stored in the `_static` directory.
- Use `:width:` in pixels or percent for scaling.
- `:align: center` or `:align: right` controls placement.
- Always include `:alt:` text for accessibility.

## 1.9 Code Blocks

Code blocks are used to show code or console examples with proper syntax highlighting.

Generic code block:

```
.. code-block:: python  
    def hello(name):  
        print(f"Hello, {name}!")  
  
.. code-block:: bash  
    echo "Hello, world!"
```

Key points:

- The language name after `code-block::` controls highlighting (e.g. `python`, `bash`, `json`, `ini`).
- Indentation is **required** — the content must be indented under the directive.

- Inline code uses double backticks: `print("Hi")`.
- For literal, unformatted text (no highlighting), use `::` at the end of a paragraph:

Example::

This text is shown exactly as typed.  
Useful for quick terminal snippets or pseudo-code.

**Tips:**

- Use `code-block:: rst` when demonstrating reStructuredText syntax.
- Keep lines under ~80 characters to avoid rendering issues in narrow layouts.
- To highlight specific lines, add `:emphasize-lines::`

```
.. code-block:: python
   :emphasize-lines: 2

   def main():
       print("Important line!")
```



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## INDEX

### A

admonitions, 5

### C

code blocks, 6

cross-references  
    syntax, 1

### D

Doom Emacs  
    snippets, 4  
    Unicode, 3

### E

Emacs  
    box-drawing, 3  
    Yasnippet, 4

### F

figures, 6

### G

grid tables, 3

### H

headings, 1

### I

images  
    inserting, 6  
index entries, 1  
indexing  
    syntax, 2

### K

keywords  
    documentation, 2

### L

literal blocks, 6

### R

references  
    internal links, 1  
reStructuredText  
    diagrams editing, 3  
    format, 1

### S

Sphinx  
    cross-references, 1  
    syntax highlighting, 6

### T

tables, 1  
    syntax, 3  
templates  
    box-drawing, 4