‹ (/documentation/01-wishlist/)

Code documentation lesson (/documentation/): Popular tools and solutions

› (/documentation/04-sphinx/)

Overview

Teaching: 10 min
Exercises: 0 min

Questions

- What tools are out there?
- What are their pros and cons?

Objectives

- Choose the right tool for the right reason.

What tools and solutions are out there?

- Comments in the source code
- README files in the source tree
- Wikis
- LaTeX/PDF
- Doxygen
- reStructuredText and Markdown
- HTML static site generators

Comments in the source code

- Advantages
  - Good for programmers
  - Version controlled alongside code
  - Can be used to auto-generate documentation for functions/classes
- Disadvantage
  - Probably not enough for users

README files in the source tree

- Advantage
  - Versioned (goes with the code development)
  - It is often good enough to have a `README.md` or `README.rst` along with your code/script
- Disadvantage
  - You need a terminal or GitHub/GitLab browser to read them
  - Sometimes users have no access to the source tree
- If you use README files, use either RST (http://docutils.sourceforge.net/rst.html) or Markdown (https://commonmark.org/help/) markup

Wikis

- Popular solutions (but many others exist):
  - MediaWiki (https://www.mediawiki.org)
  - Dokuwiki (https://www.dokuwiki.org)
- Advantage
  - Barrier to write and edit is low
- Disadvantages

  - Typically disconnected from source code repository (**reproducibility**)
  - Difficult to serve multiple versions
  - Difficult to check out a specific old version
  - Typically needs to be hosted and maintained

LaTeX/PDF

- Advantage
  - Popular and familiar in the physics and mathematics community
- Disadvantages
  - PDF format is not ideal for copy-paste ability of examples
  - Possible, but not trivial to automate rebuilding documentation after every Git push

---

Doxygen

- Auto-generates API documentation
- Documented directly in the source code
- Popular in the C++ community
- Has support for C, Fortran, Python, Java, etc., see Doxygen Github Repo (https://github.com/doxygen/doxygen)
- Many keywords are understood by Doxygen: Doxygen special commands (http://www.doxygen.nl/manual/commands.html)
- Can be used to also generate higher-level ("human") documentation
- Can be deployed to GiHub/GitLab/Bitbucket Pages

---

reStructuredText and Markdown

- Two of the most popular lightweight markup languages.
- reStructuredText (RST) has more features than Markdown but the choice is a matter of taste.
- Markdown convenient for smaller documents, but for larger and more complicated documents RST may be a better option.
- There are (unfortunately) many flavors of Markdown (https://github.com/jgm/CommonMark/wiki/Markdown-Flavors).

```
# This is a section in Markdown   This is a section in RST
                                  ========================

## This is a subsection           This is a subsection
                                  --------------------

Nothing special needed for        Nothing special needed for
a normal paragraph.               a normal paragraph.

                                  ::

    This is a code block          This is a code block


**Bold** and *emphasized*.        **Bold** and *emphasized*.

A list:                           A list:
- this is an item                 - this is an item
- another item                    - another item

There is more: images,            There is more: images,
tables, links, ...                tables, links, ...
```

- We will use RST in a Sphinx example and Markdown in a GiHub pages example

Experiment with Markdown:

- Learn Markdown in 60 seconds (http://commonmark.org/help/)
- https://dillinger.io (http://dillinger.io)

- https://stackedit.io (https://stackedit.io)

HTML static site generators

There are many tools that can turn RST or Markdown into beautiful HTML pages:

- Sphinx (http://sphinx-doc.org) **<- we will exercise this**
  - Generate HTML/PDF/LaTeX from RST and Markdown.
  - Basically all Python projects use Sphinx but **Sphinx is not limited to Python**
  - Read the docs (http://readthedocs.org) hosts public Sphinx documentation for free!
- Jekyll (https://jekyllrb.com) **<- this is how this lesson material is built**
  - Generates HTML from Markdown.
  - GitHub supports this without adding extra build steps.
- pkgdown (https://pkgdown.r-lib.org/)
  - Popular in the R community
- MkDocs (https://www.mkdocs.org/)
- GitBook (https://www.gitbook.com/)
- Hugo (https://gohugo.io)
- Hexo (https://hexo.io)

There are many more …

GitHub, GitLab, and Bitbucket make it possible to serve HTML pages:

- GitHub Pages (https://pages.github.com)
- Bitbucket Pages (https://pages.bitbucket.io/)
- GitLab Pages (https://pages.gitlab.io)

Key Points

- Some popular solutions make reproducibility and maintenance of multiple code versions difficult.

‹ (/documentation/01-wishlist/)

› (/documen/04-sphinx/)

‹ (/documentation/03-tools/)

Code documentation lesson (/documentation/): Sphinx and reStructuredText

› (/documentation/05-rtd/)

Overview

Teaching: 5 min
Exercises: 15 min

Questions

- How do we get started on writing Sphinx documentation in RST?

Objectives

- Create example Sphinx documentation and learn some RST along the way.

Group exercise: Build Sphinx documentation using RST

We will take the first steps in creating documentation using Sphinx, and learn some RST syntax along the way.

- Our goal in this episode is to build HTML pages locally on our computers.
- In the next episode we will learn how to deploy the documentation to a cloud service upon every `git push`.
- Please write your questions in the collaborative HackMD document so that we can answer them and discuss them together after the group sessions.

Prerequisites: Check whether we have the software we need

Before we start, make sure that Sphinx is part of your Python installation or environment. If you use Anaconda, you are set. If you use Miniconda or virtual environments, make sure Sphinx is installed into the Miniconda or virtual environment.

Test Sphinx installation within Python:

```
$ python --version
Python 3.7.0

$ python -c "import sphinx; print(sphinx.__version__)"
2.0.1

$ python -c "import sphinx_rtd_theme"
# this should produce no output
```

Test Sphinx tool installation:

```
$ sphinx-quickstart --version
sphinx-quickstart 2.0.1
```

The the above commands produce an error instead of printing versions (any version would do) e.g. command not found or ModuleNotFoundError please follow our installation instructions (https://coderefinery.github.io/installation/python/#installing-required-packages).

Exercise 1: Generate the basic documentation template

Create a directory for the example documentation, step into it, and inside generate the basic documentation template:

```
$ mkdir doc-example
$ cd doc-example
$ sphinx-quickstart
```

The quickstart utility will ask you some questions. For this exercise, you can go with the default answers except to specify a project name, author name, and project release:

```
> Separate source and build directories (y/n) [n]: <hit enter>
> Project name: <your project name>
> Author name(s): <your name>
> Project release [0.1]: 0.1
> Project language [en]: <hit enter>
```

A couple of files and directories are created:

File/directory	Contents
conf.py	Documentation configuration file
index.rst	Documentation master file
_build/	Directory where docs are built
_templates/	Your own HTML templates
_static/	Static files (images, styles, etc.) copied to output directory on build
Makefile & make.bat	Makefiles to build documentation using make

`Makefile` and `make.bat` (for Windows) are build scripts that wrap the sphinx commands, but we will be doing it explicitly.

Let's have a look at the `index.rst` file, which is the main file of your documentation:

```
.. myproject documentation master file, created by
   sphinx-quickstart on Mon Oct 21 21:46:06 2019.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Welcome to myproject's documentation!
=====================================

.. toctree::
   :maxdepth: 2
   :caption: Contents:



Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

- We will not use the `Indices and tables` section now, so remove it and everything below.
- The top four lines, starting with `..`, are a comment.
- The next lines are the table of contents. We can add content below:

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   feature-a
```

A common gotcha with directives is that **the first line of the content must be indented to the same level as the options (i.e., :maxdepth)**.

`feature-a` refers to a file `feature-a.rst`. Let's create it:

```
Feature A
=========

Subsection
----------

Exciting documentation in here.
Let's make a list (empty surrounding lines required):

- item 1

  - nested item 1
  - nested item 2

- item 2
- item 3
```

We now build the site:

```
$ ls

_build  _static  _templates  conf.py  feature-a.rst  index.rst

$ sphinx-build . _build

Running Sphinx v1.5.1
loading pickled environment... done
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 1 source files that are out of date
updating environment: 1 added, 1 changed, 0 removed
reading sources... [100%] index
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] index
generating indices... genindex
writing additional pages... search
copying static files... done
copying extra files... done
dumping search index in English (code: en) ... done
dumping object inventory... done
build succeeded.

$ ls _build

_sources  _static  feature-a.html  genindex.html  index.html  objects.inv search.htm
l  searchindex.js
```

Now open the file `_build/index.html` in your browser by:

Linux users type:

```
$ xdg-open _build/index.html
```

macOS users type:

```
$ open _build/index.html
```

Windows users type:

```
$ start _build/index.html
```

Others:

enter `file:///home/user/doc-example/_build/index.html` in your browser (adapting the path to your case).

Hopefully you can now see a website. If so, then you are able to build Sphinx pages locally. This is useful to check how things look before pushing changes to GitHub or elsewhere.

Note that you can change the styling by editing `conf.py` and changing the value `html_theme` (for instance you can set it to `sphinx_rtd_theme` to have the Read the Docs look).

Exercise 2: Add content to your example documentation

1. Add a entry below feature-a labeled *feature-b* to the `index.rst` file.
2. Create a file `feature-b.rst` in the same directory as your `feature-a.rst` file.
3. Add some content to feature-b, rebuild with `sphinx-build` , and refresh the browser to look at the results (Help (http://docutils.sourceforge.net/docs/ref/rst/directives.html)).

Experiment with the following RST syntax:

- *Emphasized text* and **bold text**
- Headings

```
Level 1
=======

Level 2
-------

Level 3
^^^^^^^

Level 4
"""""""
```

- An image: `.. image:: image.png`
- `A link <http://www.google.com>`_
- Numbered lists (can be automatic using `#`)

```
1. item 1
2. item 2
#. item 3
#. item 4
```

- Simple tables

```
====== ======
No.    Prime
====== ======
1      No
2      Yes
3      Yes
4      No
====== ======
```

- Code block using special marker `::`

```
The following is a code block::

  def hello():
      print("Hello world")
```

- Code block specifying syntax highlighting for other language than Python

```
.. code-block:: c

   #include <stdio.h>
   int main()
   {
       printf("Hello, World!");
       return 0;
   }
```

- You could include the contents of an external file using `literalinclude` directive, as follows:

```
.. literalinclude:: filename
```

- It is possible to combine `literalinclude` with code highlighting, line numbering, and even line highlighting.
- We can also use jupyter notebooks (*.ipynb) with sphinx. It requires `nbsphinx` extension to be installed. See nbsphinx documentation (http://nbsphinx.readthedocs.io/en/latest/) for more information

```
.. toctree::
 :maxdepth: 2
 :caption: Contents:

 feature-a
 <python_notebook_name>.ipynb
```

Rendering (LaTeX) math equations

There are two different ways to display mathematical equations within Sphinx: `pngmath` and `MathJax`. While `pngmath` displays an equation as an image, `MathJax` is using scalable vector graphics (quality remains the same after zooming). For this reason, we strongly encourage you to use `MathJax` for your mathematical equations.

To enable `MathJax` in Sphinx, you need first to add `sphinx.ext.mathjax` to the list of extensions in `conf.py`:

```
extensions = ['sphinx.ext.mathjax']
```

The following shows how to inline mathematics within a text:

```
This is an inline equation embedded :math:`a^2 + b^2 = c^2` in text.
```

An equation and equation array:

```
.. math::
   :label: myequation

   a^2 + b^2 = c^2

.. math::
   :label: myarray

   \begin{eqnarray}
     x^2 & : x < 0 \\
     x^3 & : x \ge 0 \\
   \end{eqnarray}
```

These equations can then be referenced using `:eq:`myequation`` and `:eq:`myarray``.

Where to find more

- For more RST functionality, see the Sphinx documentation (http://www.sphinx-doc.org/en/stable/rest.html) and the quick-reference (http://docutils.sourceforge.net/docs/user/rst/quickref.html).
- For Sphinx additions to standard RST, see Sphinx Markup Constructs (http://www.sphinx-doc.org/en/1.7/markup/index.html).
- https://docs.python-guide.org/writing/documentation/ (https://docs.python-guide.org/writing/documentation/)

Key Points

- Sphinx and RST are relatively lightweight options for writing documentation.

‹
(/documentation
/03-
tools/)

›
(/documen
/05-
rtd/)

‹ (/documentation/04-sphinx/)

Code documentation lesson (/documentation/): Deploying Sphinx documentation to Read the Docs

› (/documentation/06-gh-pages/)

Overview

Teaching: 0 min
Exercises: 20 min

Questions

- How do Python projects deploy their documentation?
- Can we use their solutions for projects which do not use Python?

Objectives

- Create a basic workflow which you can take home and adapt for your project.

Read the Docs (https://readthedocs.org)

- Runs Sphinx and converts RST or Markdown to HTML and PDF and hosts them for you
- Equations and images no problem
- Layout can be styled
- Many projects use Read the Docs (https://readthedocs.org) as their main site
- It is no problem to serve using your own URL `http://myproject.org` instead of `http://myproject.readthedocs.io`

Typical Read the Docs workflow

- Host source code with documentation sources on a public Git repository.
- Each time you `git push` to the repository, a `post-receive` hook triggers Read the Docs to rebuild the documentation.
- Read the Docs then clones the repository, runs Sphinx, and rebuilds HTML and PDF.
- No problem to build several branches (versions) of your documentation.

Exercise: Deploy Sphinx documentation to Read the Docs

In this exercise we will make a copy of an example repository (https://github.com/coderefinery/word-count/) on GitHub and deploy it to Read the Docs. The example project contains a script for counting the frequency distribution of words in a given file and some documentation generated using Sphinx. For bigger projects, we will have much more source files.

We will use GitHub for this exercise but it will also work with any Git repository with public read access.

1. In the first step, we will make a copy of the example repository and then clone the newly created repository to our laptop.
2. In the second step, we will enable the project on Read the Docs, then commit and push some changes and check that the documentation is automatically rebuilt.

Step 1: Go to the word-count project template (https://github.com/coderefinery/word-count/generate) and copy it to your namespace

Clone the repository

The repository contains following two folders, among few other files and folders:

- **source** folder contains the source code
- **doc** folder contains the Sphinx documentation

The doc folder contains the Sphinx configuration file (`conf.py`) and the index file (`index.rst`) and some contents (other RST files). The `conf.py` file has been adjusted to be able to autogenerate documentation from sources.

Build HTML pages locally

Inside the cloned repository, build the documentation and verify the result in your browser:

```
$ sphinx-build doc _build
```

Test HTML pages links

Inside the cloned repository, check the integrity of all external links:

```
$ sphinx-build doc -W -b linkcheck -d _build/doctrees _build/html
```

Step 2: Enable the project on Read the Docs (https://readthedocs.org)

Import a project to Read the Docs by connecting to GitHub

- Log into Read the Docs (https://readthedocs.org) and visit your dashboard (https://readthedocs.org/dashboard/)
- Click “Import a Project”
- Select “Connect to GitHub”, and choose the word-count repository
- Rename the project to user-word-count (replacing “user” with your GitHub username: we need a unique project name)
- Click “Next”

Verify the result

That’s it! Your site should now be live on http://user-word-count.readthedocs.io (replace project name).

Verify refreshing the documentation

Finally, make some changes to your documentation

- Add documentation related to other functions
- Prerequisites and how to use the program
- Rules for contribution
- Some example results (figures, tables, …)

- Commit and push them, and verify that the documentation website refreshes after your changes (can take few seconds or a minute)

📌 Do not add the generated build directory to your repository

The `_build` directory is generated locally with the command `sphinx-build doc _build` and allows you to check the content locally but it should not be part of the Git repository. We recommend to add `_build` to `.gitignore` to prevent you from accidentally adding files below `_build` to the Git repository.

📌 Running your own sphinx server

We recommend to use Read the Docs to host your documentation but if you prefer, you can host your own Sphinx server. If you want to know more about it, look at:

- https://docs.readthedocs.io/en/latest/install.html (https://docs.readthedocs.io/en/latest/install.html)
- https://pypi.org/project/sphinx-autobuild/ (https://pypi.org/project/sphinx-autobuild/)
- https://pypi.org/project/sphinx-server/ (https://pypi.org/project/sphinx-server/)

📌 Migrating your own documentation to Sphinx/ Read the Docs

- First convert your documentation to RST using Pandoc (https://pandoc.org)
- Create a file `index.rst` which lists all other RST files and provides the table of contents.
- Add a `conf.py` file. You can generate a starting point for `conf.py` and `index.rst` with `sphinx-quickstart`, or you can take the examples in this lesson as inspiration.
- Test building the documentation locally with `sphinx-build`.
- Once this works, enable the project on Read the Docs and try to push a change to your documentation.

‹ (/documentation/04-sphinx/)

› (/documen/06-gh-pages/)

‹ (/documentation/05-rtd/)

Code documentation lesson (/documentation/): Hosting websites/homepages on GitHub Pages

› (/documentation/07-discussion/)

Overview

Teaching: 0 min
Exercises: 20 min
Questions

- How to serve a website/homepage using GitHub

Hosting websites/homepages on GitHub Pages

You can host your personal homepage or group webpage or project website on GitHub using GitHub Pages (https://pages.github.com/).

GitLab (https://about.gitlab.com/features/pages/) and Bitbucket (https://confluence.atlassian.com/bitbucket/publishing-a-website-on-bitbucket-cloud-221449776.html) also offer a very similar solution.

Unless you need user authentication or a sophisticated database behind your website, GitHub Pages (https://pages.github.com/) can be a very nice alternative to running your own web servers.

This is how all https://coderefinery.org (https://coderefinery.org) material is hosted.

---

organizational website
https://github.com/myorganization/myorganization.github.io/ (master)
↓
https://myorganization.github.io/

personal homepage
https://github.com/myuser/myuser.github.io/ (master)
↓
https://myuser.github.io

project website
https://github.com/myorganization/myproject/ (gh-pages)
↓
https://myorganization.github.io/myproject/

https://github.com/myuser/myproject/ (gh-pages)
↓
https://myuser.github.io/myproject/

Exercise

- Deploy own website reusing a template:
  - Follow https://pages.github.com/ (https://pages.github.com/)
  - Select “Project site”
  - Select “Choose a theme” (for instance “Minimal”)
  - Click “Select theme”
  - Adjust the README.md and commit
  - Browse your page on `http://username.github.io/repository` (adjust “username” and “repository”)
- Make a change to the repository after the webpage has been deployed for the first time
- Verify that the change shows up on the website a minute or two later

The documentation for GitHub Pages is very good so no need for us to duplicate screenshots: https://pages.github.com/ (https://pages.github.com/)

Real-life examples

- Research Software Hour
  - Source: https://raw.githubusercontent.com/ResearchSoftwareHour/researchsoftwarehour.github.io/master/about.md (https://raw.githubusercontent.com/ResearchSoftwareHour/researchsoftwarehour.github.io/master/about.md)
  - Result: https://researchsoftwarehour.github.io/about/ (https://researchsoftwarehour.github.io/about/)
- This lesson
  - Source: https://raw.githubusercontent.com/coderefinery/documentation/gh-pages/_episodes/06-gh-pages.md (https://raw.githubusercontent.com/coderefinery/documentation/gh-pages/_episodes/06-gh-pages.md)
  - Result: this page

Discussion

- You can use HTML directly or another static site generator if you prefer to not use the default Jekyll (https://jekyllrb.com/).
- It is no problem to use a custom domain instead of `*.github.io`.

‹ (/documentation/05-rtd/)

› (/documen/07-discussion/