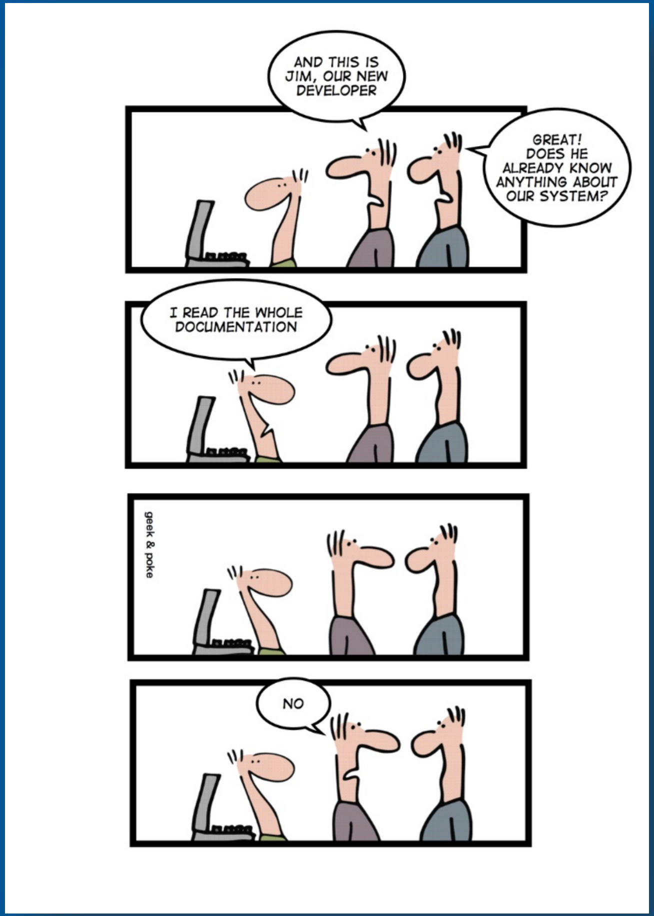
Continuous Documentation for Users, Developers and Maintainers

Platform for Advanced Scientific Computing (PASC19)
June 14, 2019

Tobias Frust

Helmholtz-Zentrum Dresden-Rossendorf (HZDR)

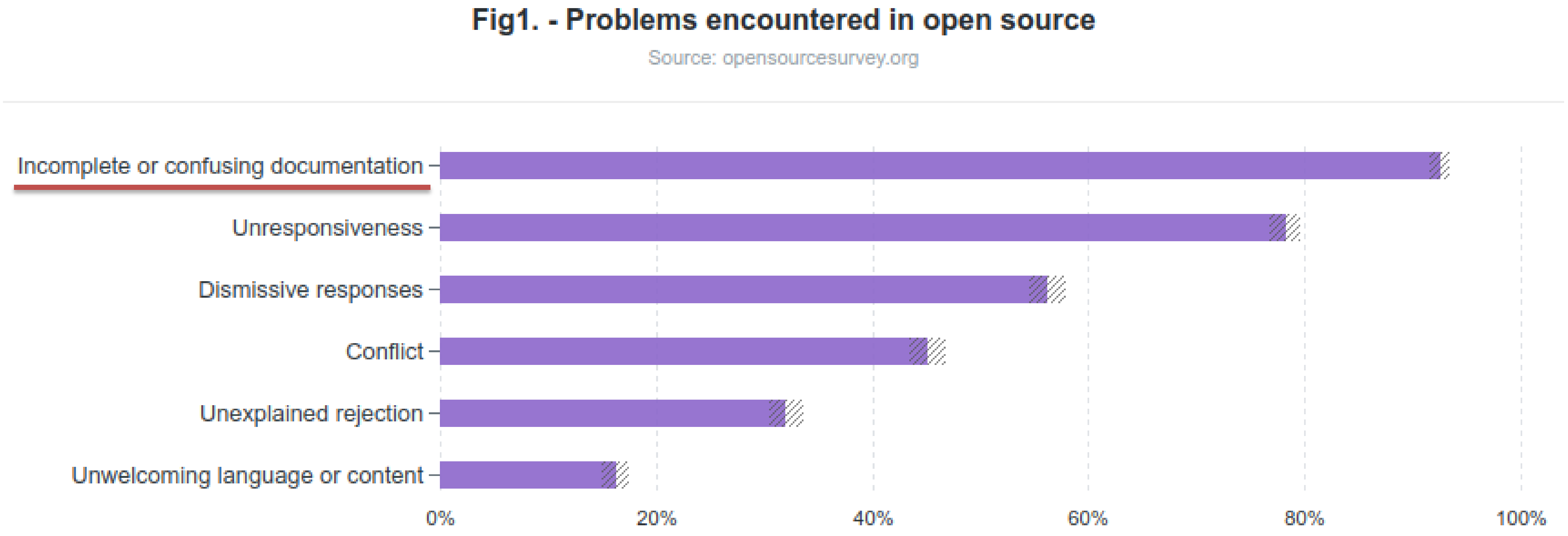

AND THIS IS JIM, OUR NEW DEVELOPER
GREAT! DOES HE ALREADY KNOW ANYTHING ABOUT OUR SYSTEM?
I READ THE WHOLE DOCUMENTATION
geek & poke
NO

[1]

DOI 10.5281/zenodo.3247324

[1] From https://geekandpoke.typepad.com/.a/6a00d8341d3df553ef0168eabe2192970c-pi

GitHub Open Source Survey



Zlotnick, Frances, "GitHub Open Source Survey 2017". GitHub, Inc., 02-Jun-2017. and https://opensourcesurvey.org/2017/

Why documentation is important for your software project

"It doesn't matter how good your software is, because ***if the documentation is not good enough, people will not use it****.*

Even if for some reason they have to use it because they have no choice, without good documentation, they won't use it effectively or the way you'd like them to."

What nobody tells you about documentation (Daniele Procida) - https://www.divio.com/blog/documentation/

Why documentation is important for your software project

Documentation can save time and pay for itself:

- Helps to create inclusive communities
- Makes for a better software product
- Reduces cost of ownership
- Reduces the user's learning curve
- Makes users happy
- Improves Reproducibility

Incorrect, old or missing documentation can:

- Waste time
- Cause errors and destroy data
- Turn away customers/scientists
- Increase support costs
- Shorten a product's life span

Writing effective documentation can be challenging

- Documentation must be kept **up-to-date.**
- Documentation needs to be considered **continuously.**
- Features vs. Documentation: Features often win.
- Needs to be **adapted to the target audience**.

Photo by Mārtiņš Zemlickis on Unsplash

Whom is documentation for?

- For **you,** the developer himself
  - You will be using and working with your code in months
  - You want people to use your code and give credit (e.g. citation)
  - Others could be encouraged to contribute to your code
- For **others**, e.g. users, contributors
  - Easily use your code and build upon it.
- For **science**
  - Encourage Open Science.
  - Allow Reproducibility and Transparency.

Photo by Miguel Henriques on Unsplash

Components of Software Documentation

Components

Tutorials

- Learning-oriented
- For the newcomers to get started

How-To Guides

- Goal-oriented
- Shows how to solve a specific problem

Explanation

- Understanding-oriented
- Provides background and context

Technical Reference

- Information-oriented
- Describes the system
- Is accurate and complete

Best Practices for Documenting Scientific Software

Mandatory Prerequisite

- Put your code AND documentation under **version control.**
- Use a **Software Management System** (e.g. GitHub, GitLab, Bitbucket, …).
- If possible, have a **public project.**

#1: Put a README file into the root of your repository

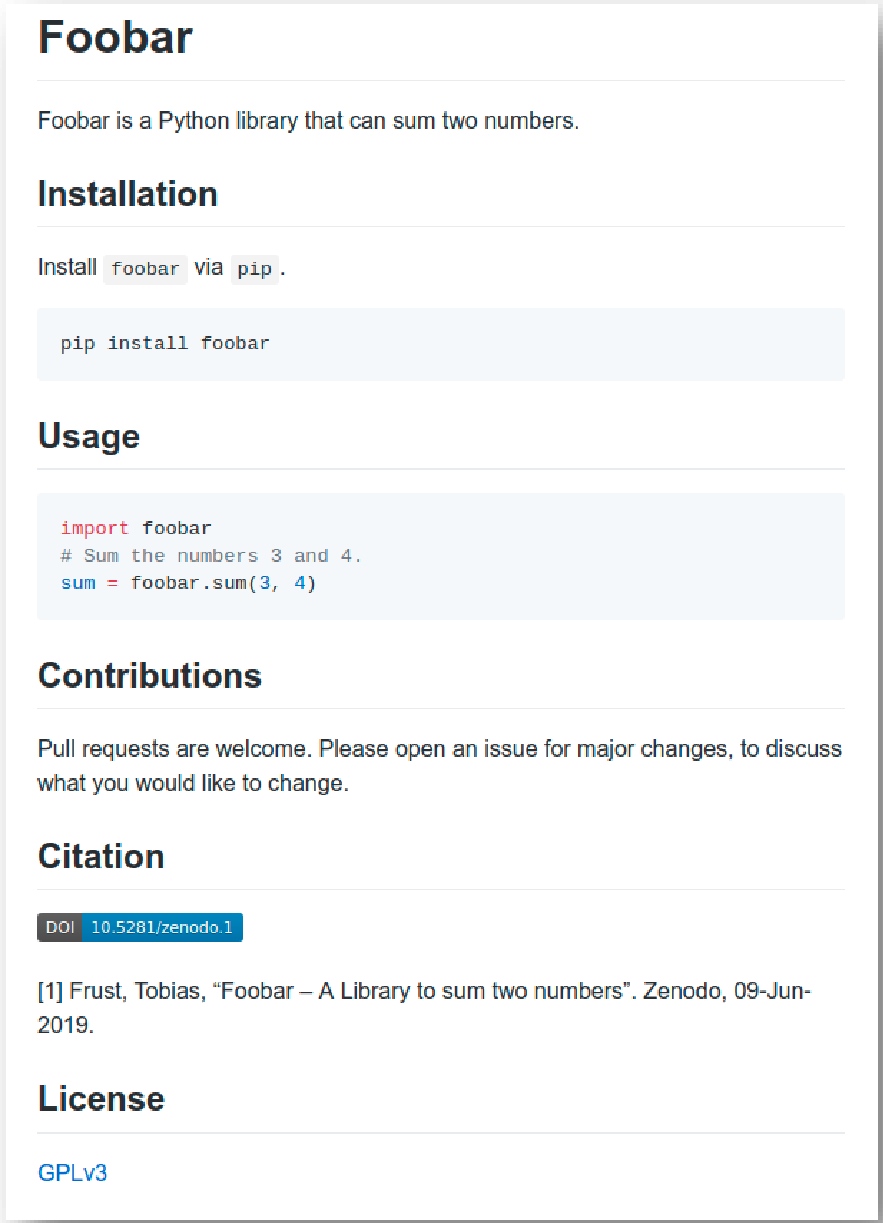
- A README is like the **homepage for your software** project.
- Store as a **text file** – readable on all operating systems
- Use **Markup Languages** (e.g. Markdown, Restructured Text)

→ Will be rendered by Code Hosting sites.

- Minimum Content:
  - Description
  - Installation instructions
  - Usage instructions
  - License Information

~~~
# Project Title

A short description.

## Installation

A step by step installation guide

```bash
Steps to install the software.
```

## Usage

Provide a short usage/quick start example.

```bash
Code example, …
```

## Contributions

Information about contribution guidelines.

## Citation

- Tell, how this software can be cited.
- Provide a DOI for each version of your software.

## License

[GPLv3](License.md)
~~~

~~~markdown
# Foobar

Foobar is a Python library that can sum two numbers.

## Installation

Install `foobar` via `pip`.

```bash
pip install foobar
```

## Usage

```python
import foobar
# Sum the numbers 3 and 4.
sum = foobar.sum(3, 4)
```

## Contributions

Pull requests are welcome. Please open an issue for major
changes, to discuss what you would like to change.

## Citation

[![DOI](https://zenodo.org/badge/DOI/10.5281/zenodo.1.svg)](
https://doi.org/10.5281/zenodo.1)

[1] Frust, Tobias, “Foobar – A Library to sum two numbers”.
Zenodo, 09-Jun-2019.

## License

[GPLv3](License.md)
~~~



Foobar

Foobar is a Python library that can sum two numbers.

Installation

Install `foobar` via `pip`.

```
pip install foobar
```

Usage

```python
import foobar
# Sum the numbers 3 and 4.
sum = foobar.sum(3, 4)
```

Contributions

Pull requests are welcome. Please open an issue for major changes, to discuss what you would like to change.

Citation

DOI 10.5281/zenodo.1

[1] Frust, Tobias, “Foobar – A Library to sum two numbers”. Zenodo, 09-Jun-2019.

License

GPLv3

#2: Include a Quick-Start Guide

Goal: Allow people to immediately start playing with your tool. Guide them through the first steps.

Introduction

- What is this Software used for?

Requirements

- e.g. Operating System, dependencies.

Installation

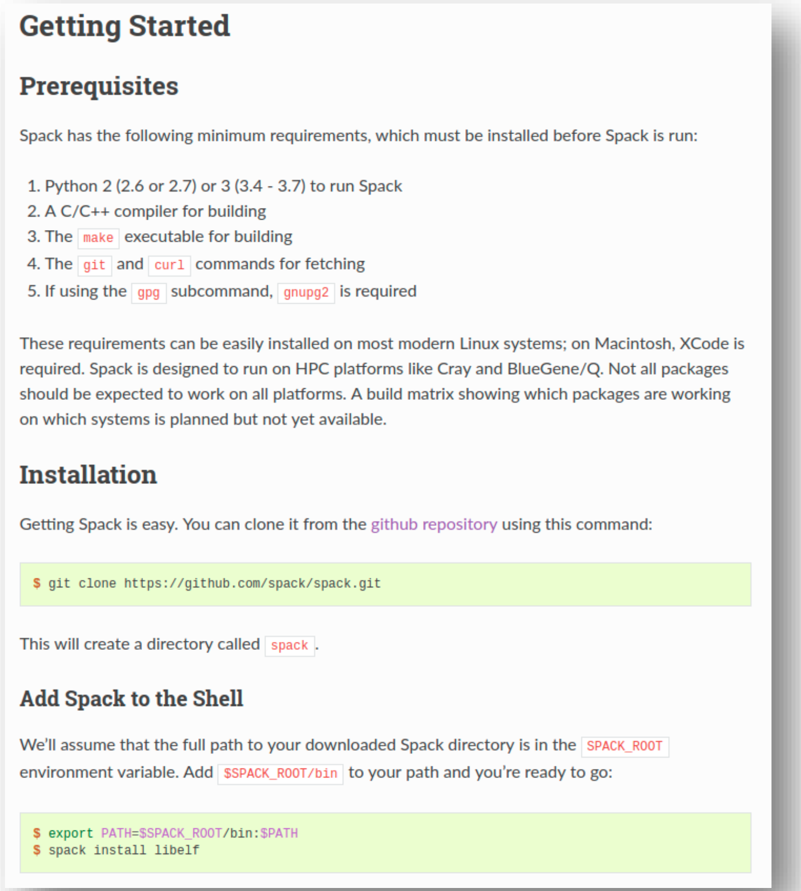
- Describe how to install the software step by step.

Usage

- Provide a motivating example covering the general concepts.
- Balance between simplicity and complexity.

Referential Information

- Link to the detailed user and installation guide.
- Link to other follow-up material.



Getting Started

Prerequisites

Spack has the following minimum requirements, which must be installed before Spack is run:

1. Python 2 (2.6 or 2.7) or 3 (3.4 - 3.7) to run Spack
2. A C/C++ compiler for building
3. The `make` executable for building
4. The `git` and `curl` commands for fetching
5. If using the `gpg` subcommand, `gnupg2` is required

These requirements can be easily installed on most modern Linux systems; on Macintosh, XCode is required. Spack is designed to run on HPC platforms like Cray and BlueGene/Q. Not all packages should be expected to work on all platforms. A build matrix showing which packages are working on which systems is planned but not yet available.

Installation

Getting Spack is easy. You can clone it from the github repository using this command:

```
$ git clone https://github.com/spack/spack.git
```

This will create a directory called `spack`.

Add Spack to the Shell

We'll assume that the full path to your downloaded Spack directory is in the `SPACK_ROOT` environment variable. Add `$SPACK_ROOT/bin` to your path and you're ready to go:

```
$ export PATH=$SPACK_ROOT/bin:$PATH
$ spack install libelf
```

Examples:

Numpy - https://docs.scipy.org/doc/numpy/user/quickstart.html

Scipy - https://www.scipy.org/getting-started.html

Spack - https://spack.readthedocs.io/en/latest/getting_started.html

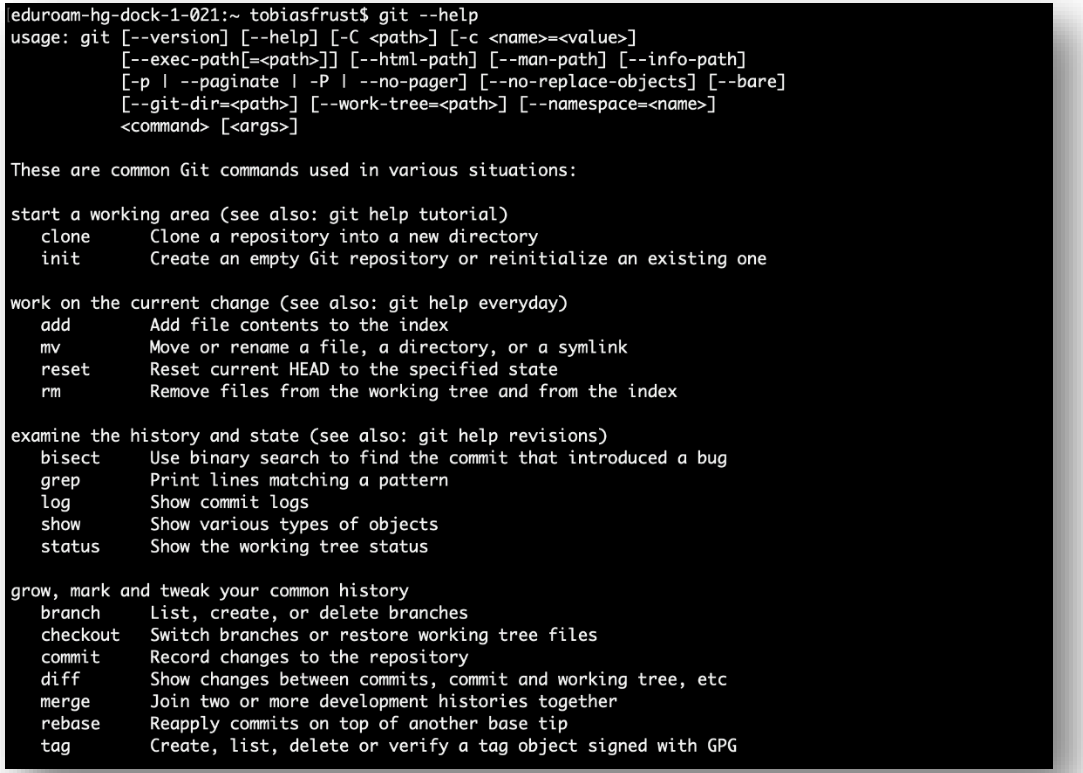
#3: Include Examples

- *Showing is better than telling.*
- Include examples going beyond simple instructions.
- Have enough examples to show the functionality of your software.
- Examples can be a good starting point for first user attempts.

- Examples:
  - Keras – 35 examples including a README (https://github.com/keras-team/keras/tree/master/examples)
  - Matplotlib (https://matplotlib.org/examples/)

#4: Provide a help command for your Command Line Interface

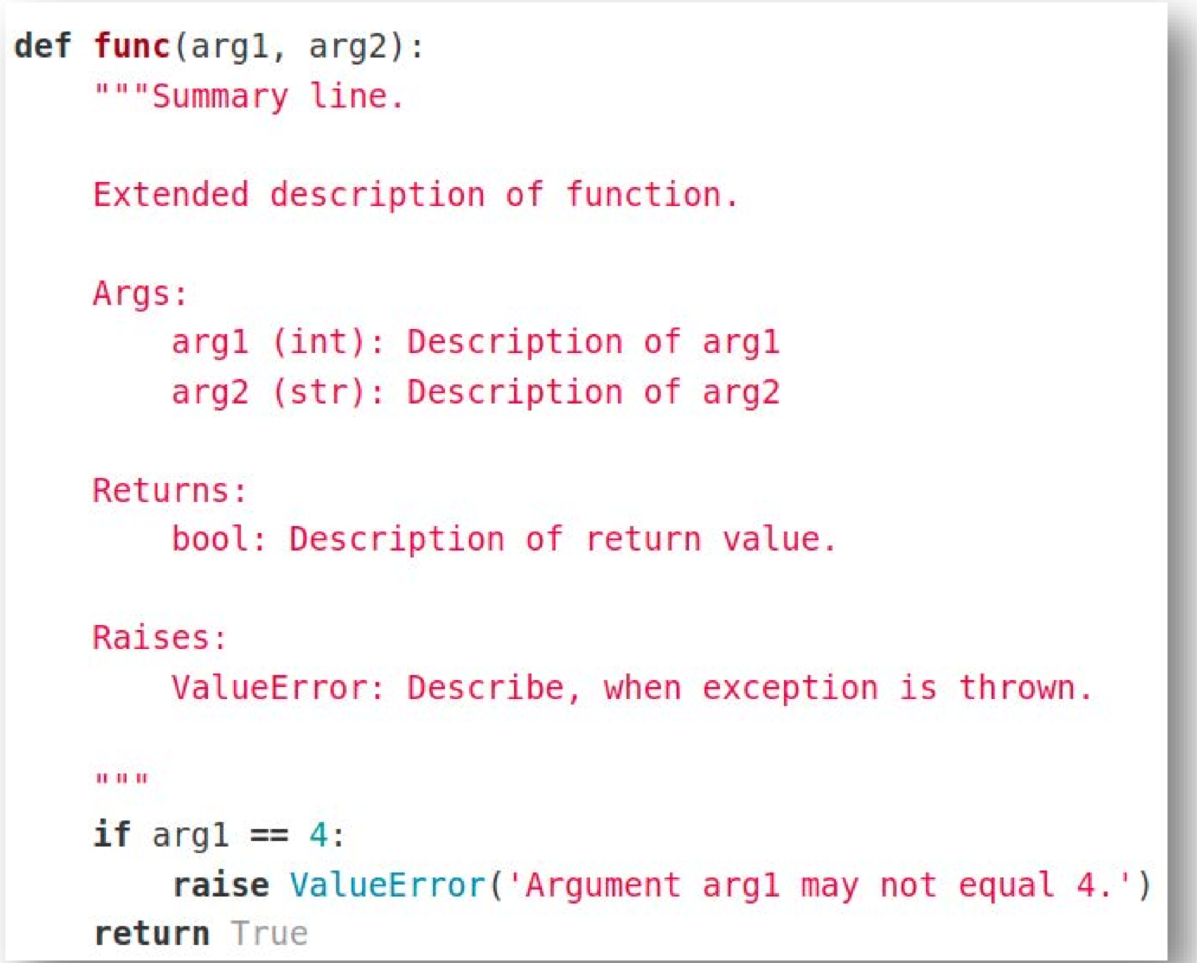
- Scientific software often ships with a Command Line Interface (CLI).
- Good for development effort; but hard to figure out what it does.
- Document CLI with a `help` command (`-h`/`--help`)
- Include:
  - Usage information.
  - Subcommands (if applicable).
  - Describe options/arguments and Environment variables.
  - Maybe add examples.
- Example: Click for Python, Boost Program Options, …



```
eduroam-hg-dock-1-021:~ tobiasfrust$ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG
```

#5: Document your entire Application Programming Interface (API)

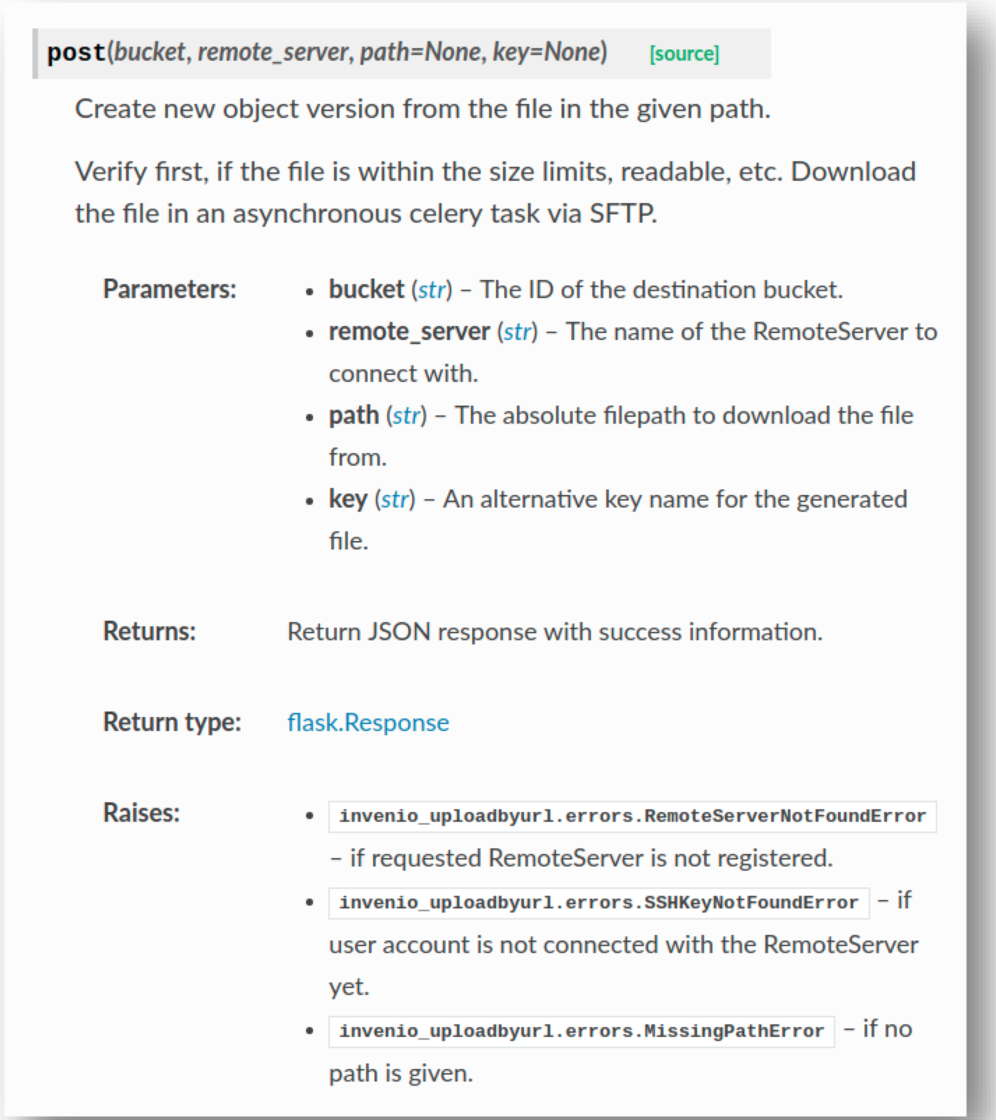
- The API is how people interact with your code.
- Use a consistent style that is understood by documentation tools ([#6]).
  - e.g. Google Style Guide (https://google.github.io/styleguide/)
- For functions, define:
  - Short **description.**
  - **Input/Output** parameters with type.
  - **Errors**, that can be raised.
- Classes should define:
  - **Attributes** and their type.
  - Describe **methods.**



```python
def func(arg1, arg2):
    """Summary line.

    Extended description of function.

    Args:
        arg1 (int): Description of arg1
        arg2 (str): Description of arg2

    Returns:
        bool: Description of return value.

    Raises:
        ValueError: Describe, when exception is thrown.

    """
    if arg1 == 4:
        raise ValueError('Argument arg1 may not equal 4.')
    return True
```

#6: Use Automated documentation tools

- *Let documentation write itself - at least parts of it!*
- Tools help to
  - Create **beautiful documentation** in multiple output formats.
  - Extract **documentation** directly from the source code.
  - Create extensive **cross-referencing.**
  - Generate detailed **API documentation**.
- **Python:** Sphinx (sphinx-doc.org)
- **C++:** Doxygen (doxygen.nl)
- **R:** Roxygen (https://github.com/klutometis/roxygen)
- **Java:** Javadoc



post(*bucket*, *remote_server*, *path=None*, *key=None*) [source]

Create new object version from the file in the given path.

Verify first, if the file is within the size limits, readable, etc. Download the file in an asynchronous celery task via SFTP.

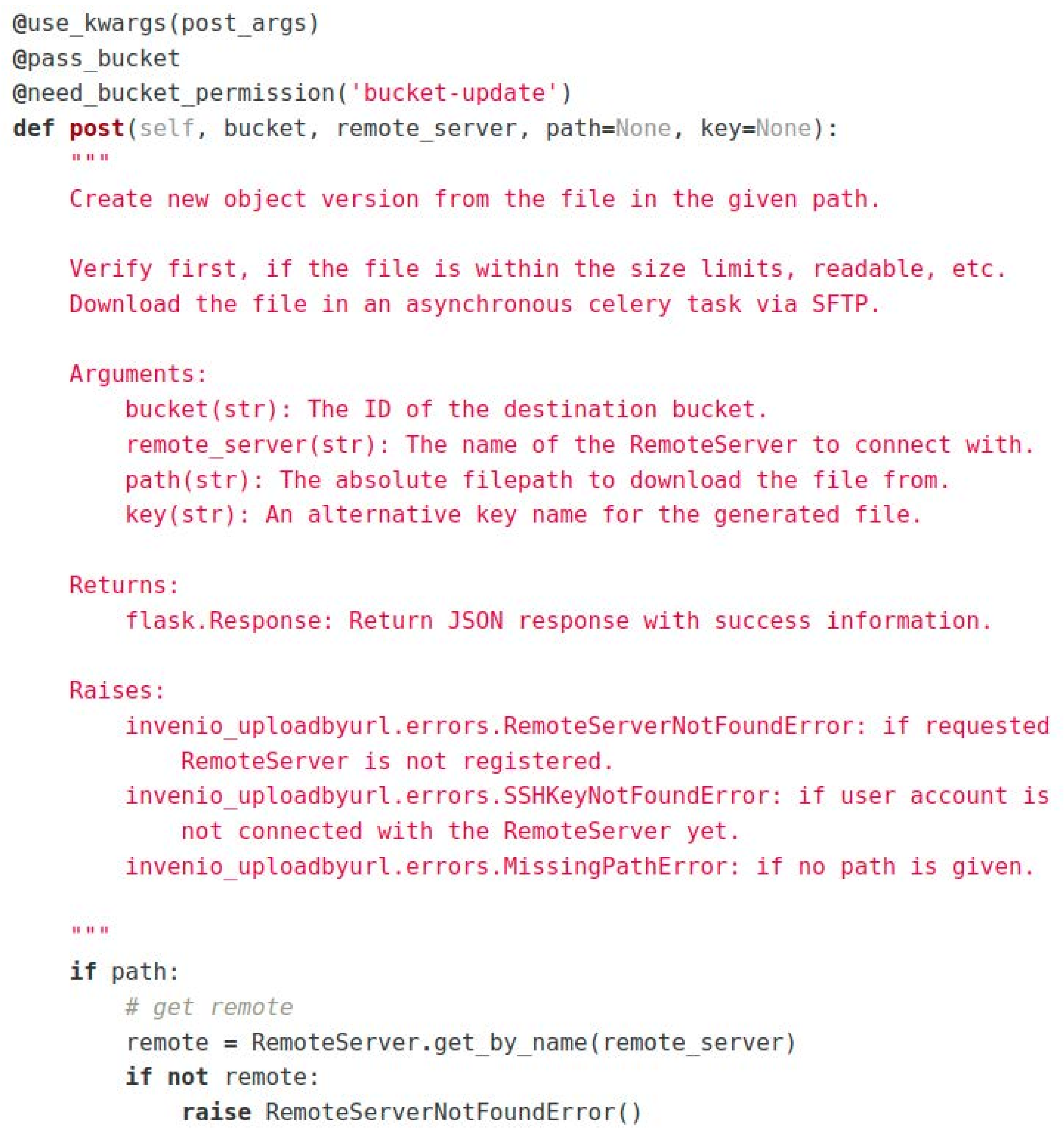
Parameters:
- **bucket** (*str*) – The ID of the destination bucket.
- **remote_server** (*str*) – The name of the RemoteServer to connect with.
- **path** (*str*) – The absolute filepath to download the file from.
- **key** (*str*) – An alternative key name for the generated file.

Returns: Return JSON response with success information.

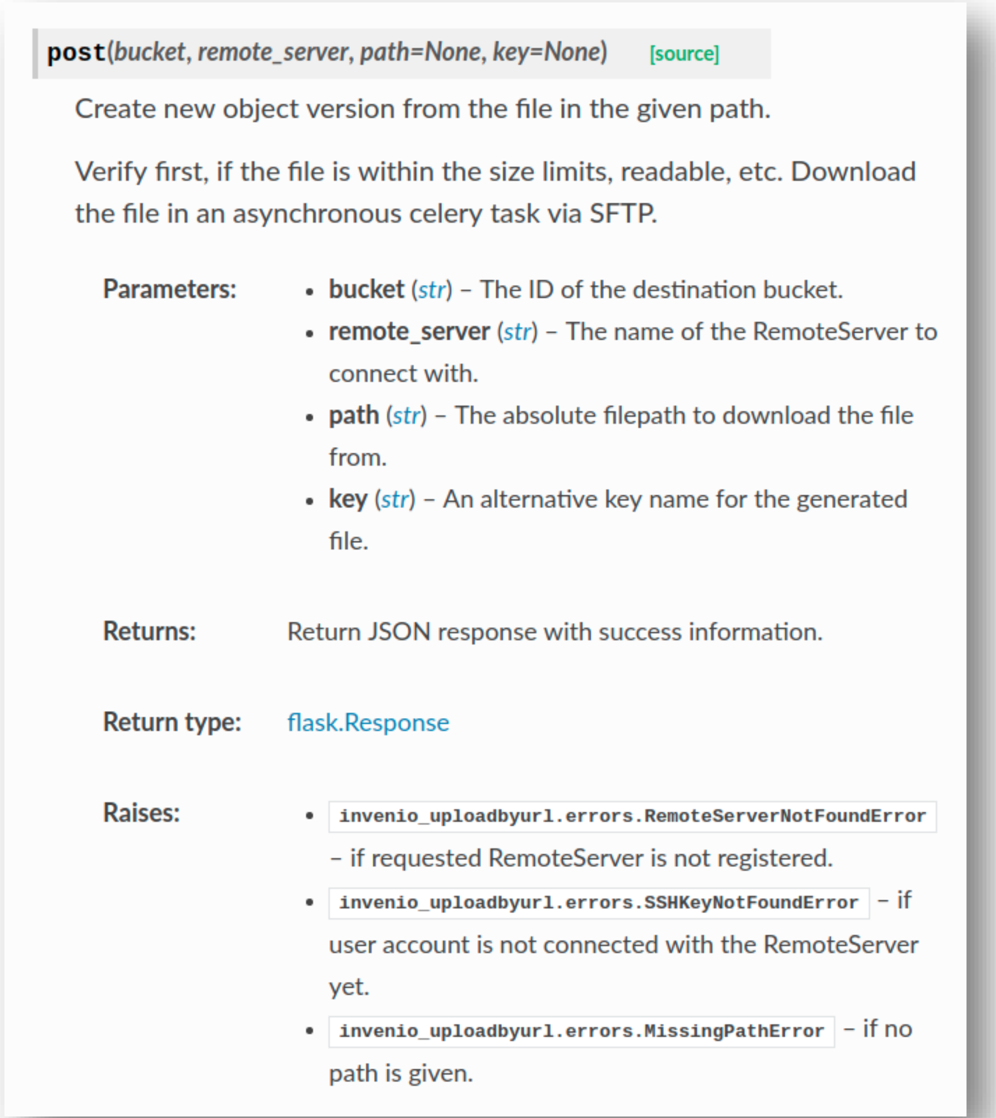
Return type: flask.Response

Raises:
- `invenio_uploadbyurl.errors.RemoteServerNotFoundError` – if requested RemoteServer is not registered.
- `invenio_uploadbyurl.errors.SSHKeyNotFoundError` – if user account is not connected with the RemoteServer yet.
- `invenio_uploadbyurl.errors.MissingPathError` – if no path is given.

#6: Use Automated documentation tools



```
@use_kwargs(post_args)
@pass_bucket
@need_bucket_permission('bucket-update')
def post(self, bucket, remote_server, path=None, key=None):
    """
    Create new object version from the file in the given path.

    Verify first, if the file is within the size limits, readable, etc.
    Download the file in an asynchronous celery task via SFTP.

    Arguments:
        bucket(str): The ID of the destination bucket.
        remote_server(str): The name of the RemoteServer to connect with.
        path(str): The absolute filepath to download the file from.
        key(str): An alternative key name for the generated file.

    Returns:
        flask.Response: Return JSON response with success information.

    Raises:
        invenio_uploadbyurl.errors.RemoteServerNotFoundError: if requested
            RemoteServer is not registered.
        invenio_uploadbyurl.errors.SSHKeyNotFoundError: if user account is
            not connected with the RemoteServer yet.
        invenio_uploadbyurl.errors.MissingPathError: if no path is given.

    """
    if path:
        # get remote
        remote = RemoteServer.get_by_name(remote_server)
        if not remote:
            raise RemoteServerNotFoundError()
```



post(*bucket*, *remote_server*, *path=None*, *key=None*) [source]

Create new object version from the file in the given path.

Verify first, if the file is within the size limits, readable, etc. Download the file in an asynchronous celery task via SFTP.

Parameters:
- **bucket** (*str*) – The ID of the destination bucket.
- **remote_server** (*str*) – The name of the RemoteServer to connect with.
- **path** (*str*) – The absolute filepath to download the file from.
- **key** (*str*) – An alternative key name for the generated file.

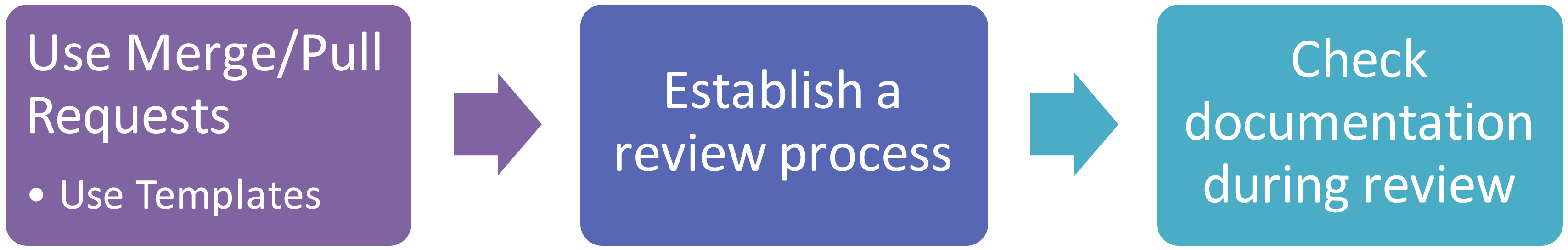
Returns: Return JSON response with success information.

Return type: flask.Response

Raises:
- `invenio_uploadbyurl.errors.RemoteServerNotFoundError` – if requested RemoteServer is not registered.
- `invenio_uploadbyurl.errors.SSHKeyNotFoundError` – if user account is not connected with the RemoteServer yet.
- `invenio_uploadbyurl.errors.MissingPathError` – if no path is given.

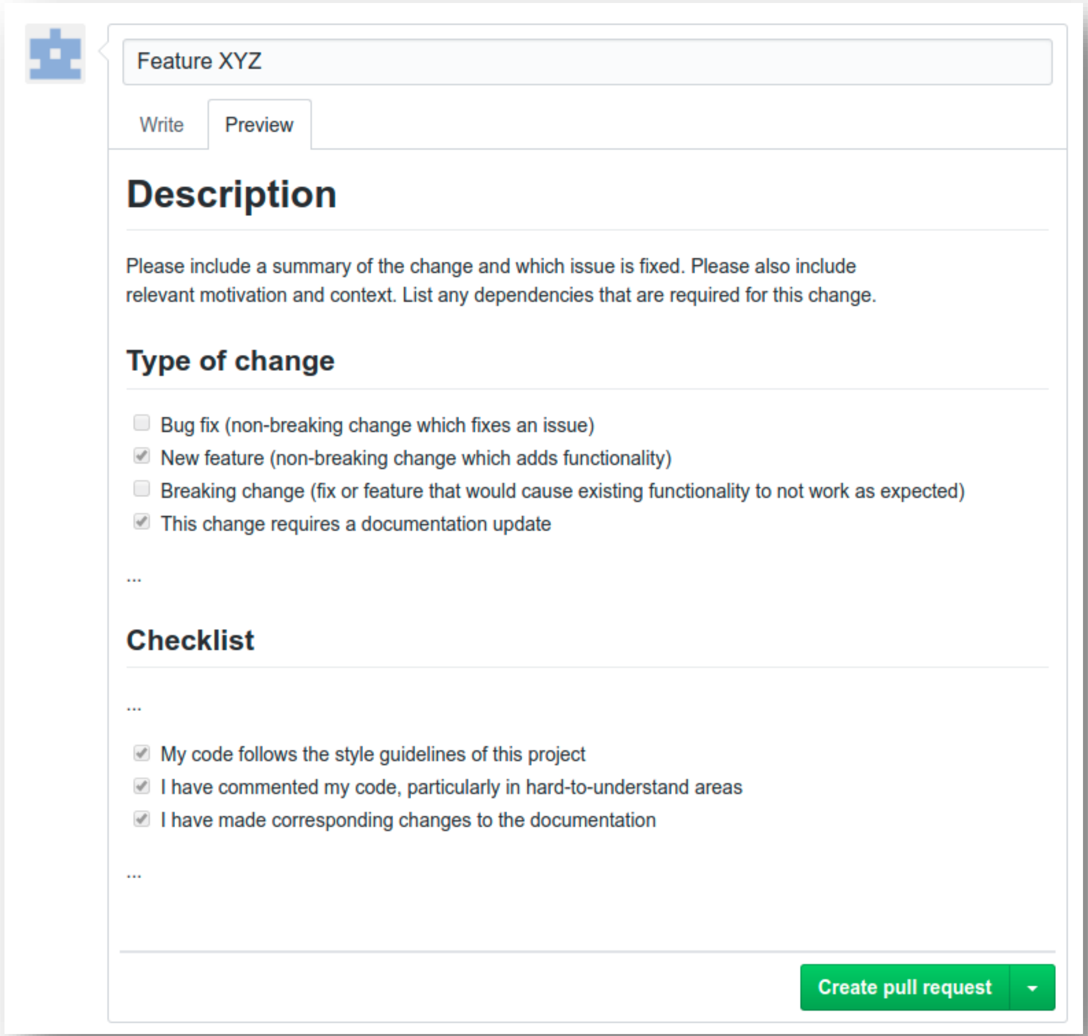
#7: Make a documentation check part of your Merge/Pull Request strategy

- Writing documentation cannot fully be automated – it's a creative process.
- Make documentation an integral part of the contribution process.



#7: Make a documentation check part of your Merge/Pull Request strategy

- Structure contributions by providing **templates**
- Make documentation part of each Merge/Pull request template
- In GitHub add a file called `PULL_REQUEST_TEMPLATE` to one of three locations:
  - The root of the project
  - `.github/` folder
  - `docs/` folder
- In GitLab, create `*.md` file inside the `.gitlab/merge_request_templates/` directory



Feature XYZ

Description

Please include a summary of the change and which issue is fixed. Please also include relevant motivation and context. List any dependencies that are required for this change.

Type of change

- [ ] Bug fix (non-breaking change which fixes an issue)
- [✓] New feature (non-breaking change which adds functionality)
- [ ] Breaking change (fix or feature that would cause existing functionality to not work as expected)
- [✓] This change requires a documentation update

...

Checklist

...

- [✓] My code follows the style guidelines of this project
- [✓] I have commented my code, particularly in hard-to-understand areas
- [✓] I have made corresponding changes to the documentation

...

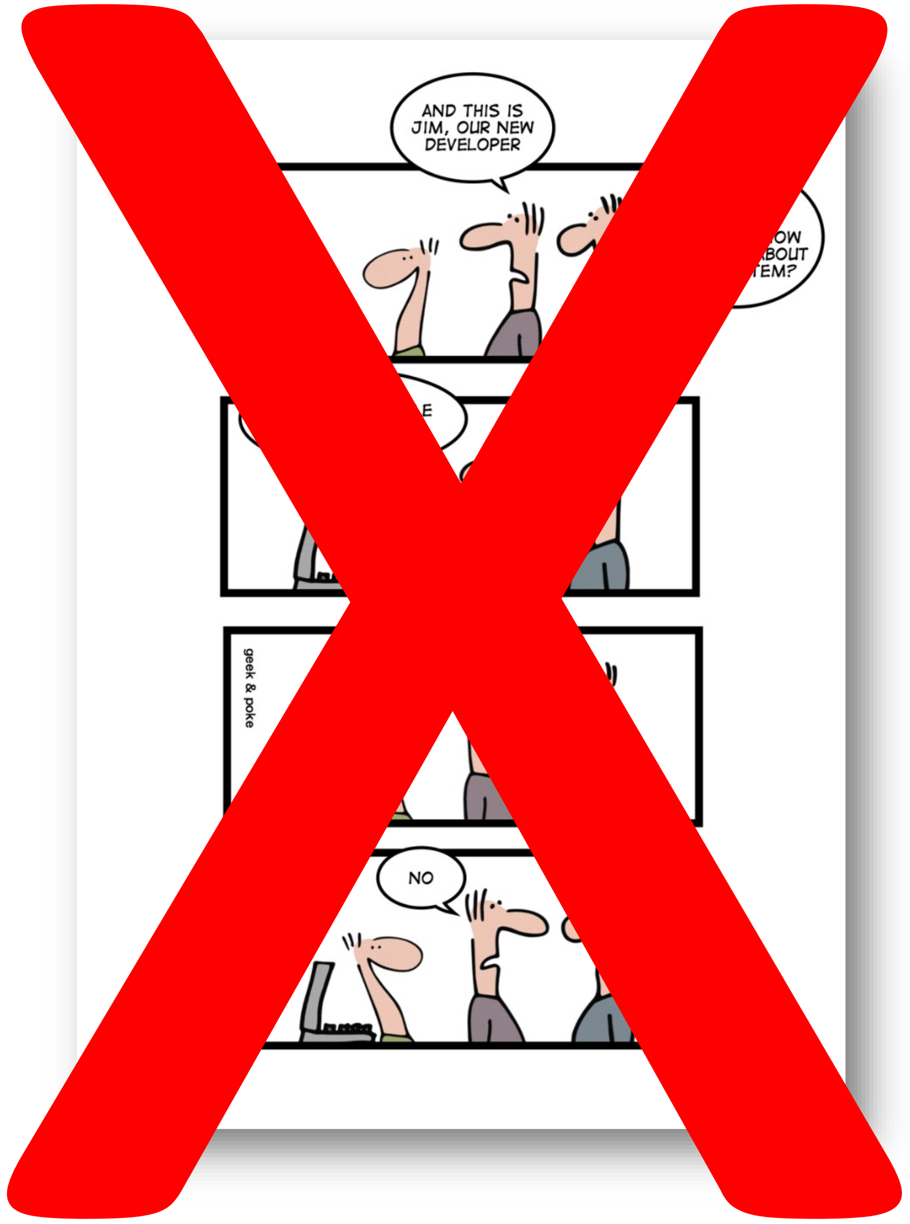
#8: Automate as much as possible

- **Automate publishing** of documentation for new releases.
  - ReadTheDocs (https://readthedocs.org)
  - GitHub/GitLab Pages (https://gitlab.pages.io – https://github.pages.com)
- Do **static analysis** (check style guide, check documentation style, …).
- Use **CI/CD** (e.g. GitLab CI, Travis, …) – Make reviewer's life easier.

Look at **Awesome Static Analysis** for tools for your programming language:
https://matthias-endler.de/awesome-static-analysis/ or
https://github.com/mre/awesome-static-analysis

Conclusion

- Consider documentation from the **very beginning**.
- Use **standards**.
- Preferably, **automate as much as possible** – you will love it once it is in place!
- Make **creating documentation more enjoyable** than boring.



References

- Zlotnick, Frances (2017), “GitHub Open Source Survey 2017”. GitHub, Inc., https://doi.org/10.5281/zenodo.806811
- Lee BD (2018) Ten simple rules for documenting scientific software. PLOS Computational Biology 14(12): e1006561. https://doi.org/10.1371/journal.pcbi.1006561
- Procida, Daniele (2017), What nobody tells you about documentation, Blog Post, https://www.divio.com/blog/documentation/
- Berkeley Library (2018), How to Write a Good Documentation, https://guides.lib.berkeley.edu/how-to-write-good-documentation

Examples

- C++: Doxygen + Breathe + Sphinx + ReadTheDocs; xtensor Repository on GitHub – Corresponding documentation.
- Sphinx Documentation- http://www.sphinx-doc.org/en/master/
- Read the Docs: Documentation; https://docs.readthedocs.io
- Doxygen Documentation; http://www.doxygen.nl
- Pull request template on GitHub; https://help.github.com/en/articles/creating-a-pull-request-template-for-your-repository
- Merge request template on GitLab; https://docs.gitlab.com/ee/user/project/description_templates.html#creating-merge-request-templates