Versioning and Stable Releases

Dushyant Mehra

Outline

- Why stable releases?
- Creating tags in Github
- Creating a stable release from a Tag
- Creating a Uploading a Package using PyPI and Twine
- Versioning
- Stable DOIs for publications using Zenodo

Why stable release?

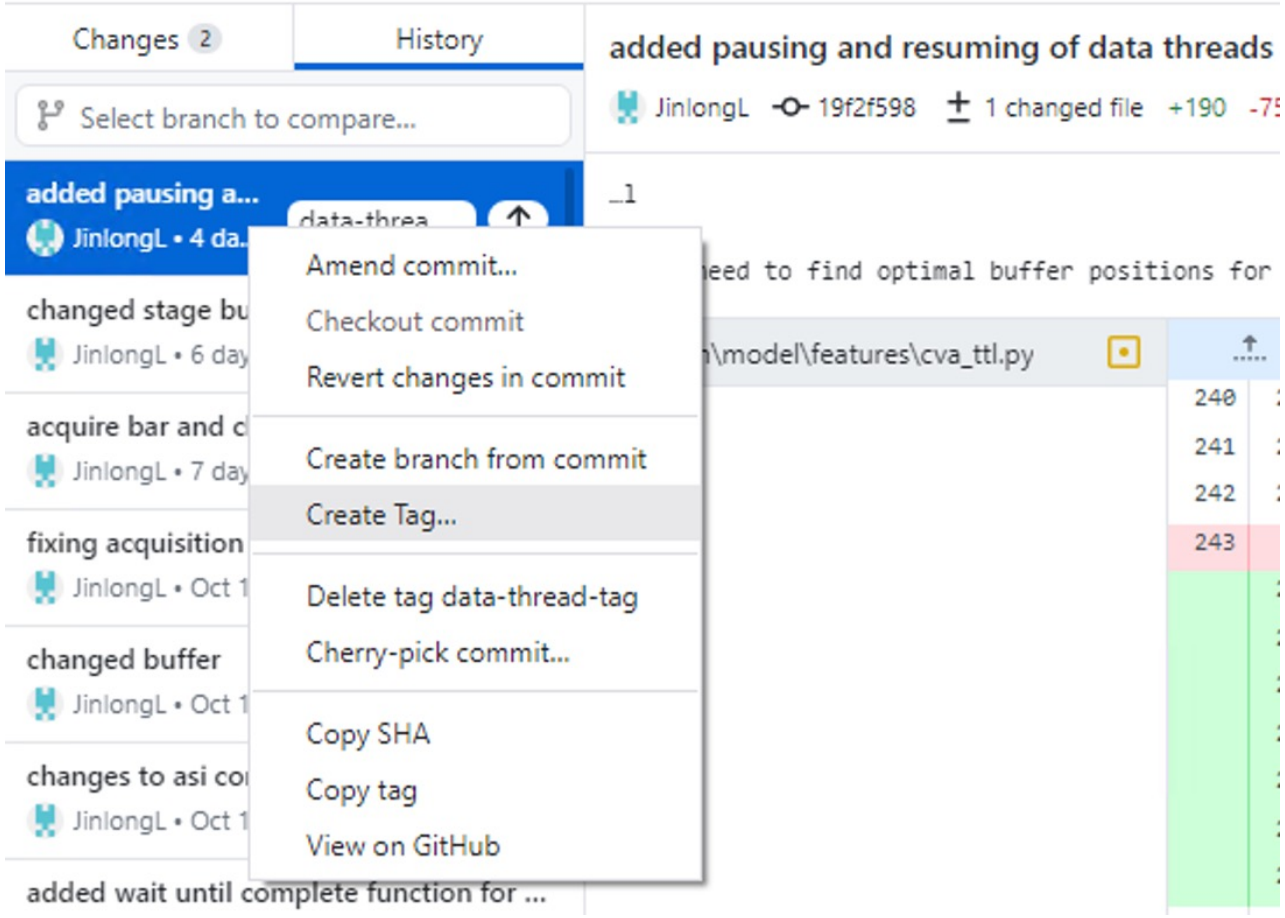
- Collaborative codebases are constantly updated and changed.
  - Having stable versions helps with managing releases
- Important for reproducibility of research for publications.
- It is important to have a specific release that ensures code is still functional
  - Ensure packages that are installed to make the

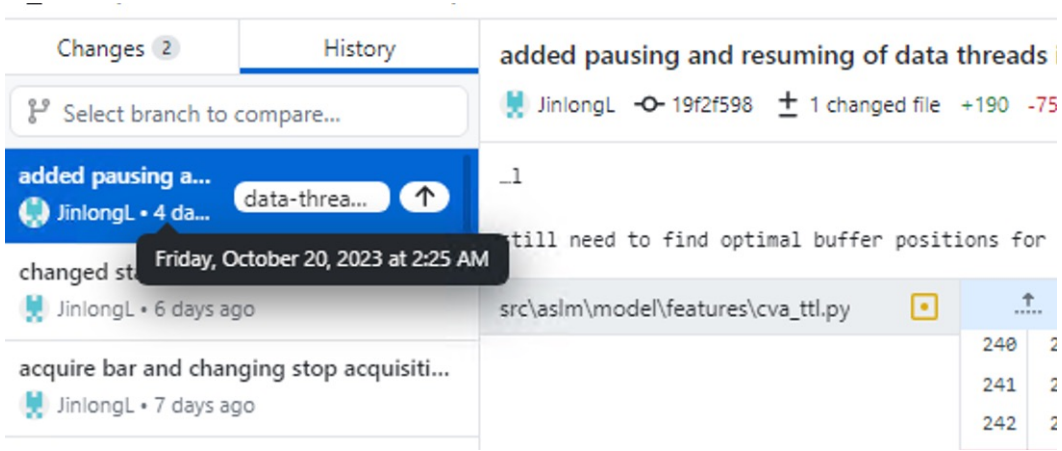
Implementing stable release

- Github Tag
- Github Stable Release
- Packaging Projects using PyPI and twine
- Github Release and stable DOI via Zenodo
- If we have time: Conda Constructor

Creating a Github Tag to a commit

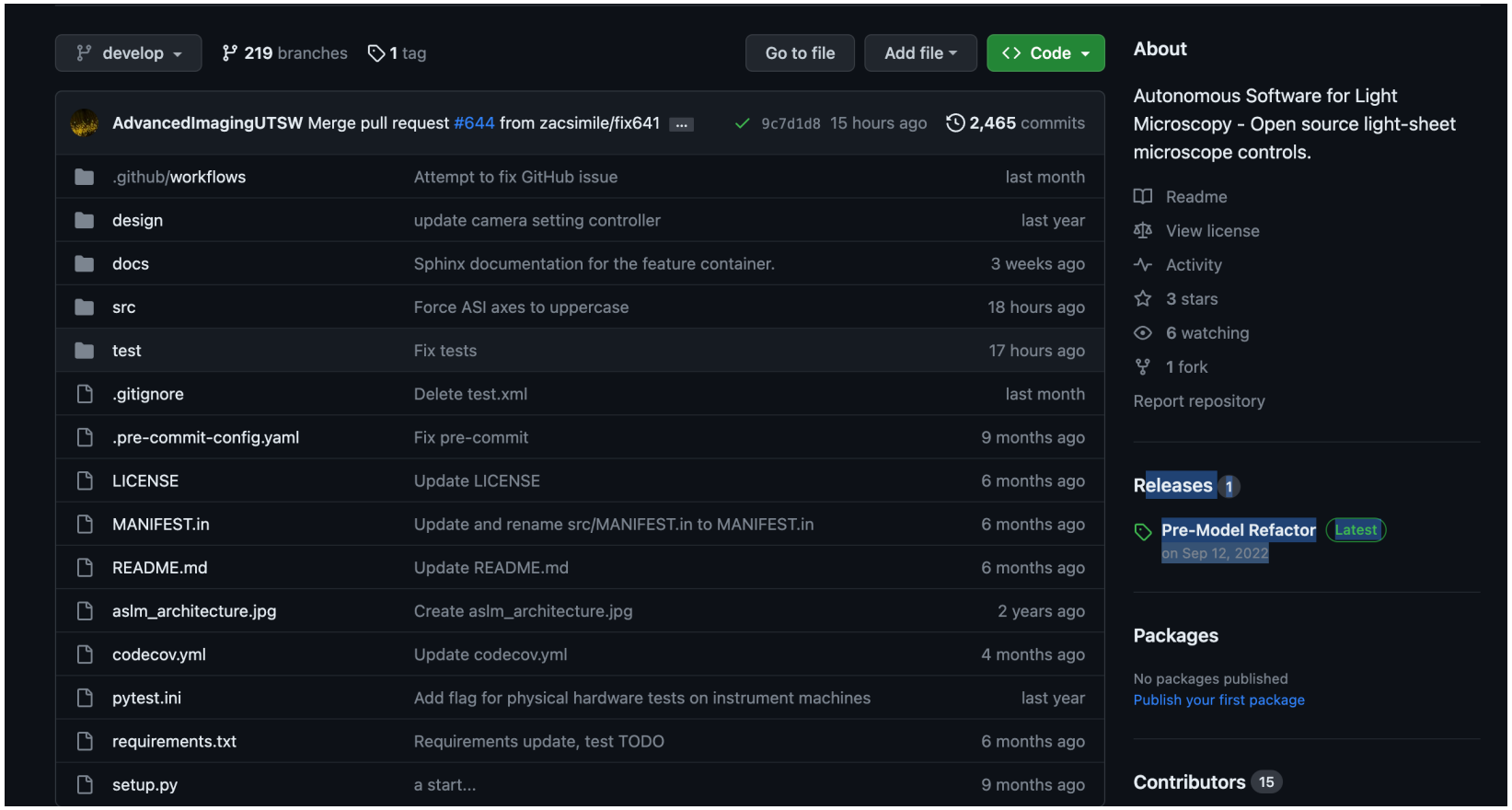
- In Github it is possible tag a to a commit





Creating a stable release in Github

- Can create a stable release from a tag on recent commit using Github



Creating a stable release using Github

Releases | Tags

Draft a new release | Find a release

Sep 12, 2022

codeCollision4

v0.0.1-alpha

f1a11da

Compare

Pre-Model Refactor

Latest

This release aims to create a stable build prior to the major refactoring of the Model. Below are the pull requests and change-log.

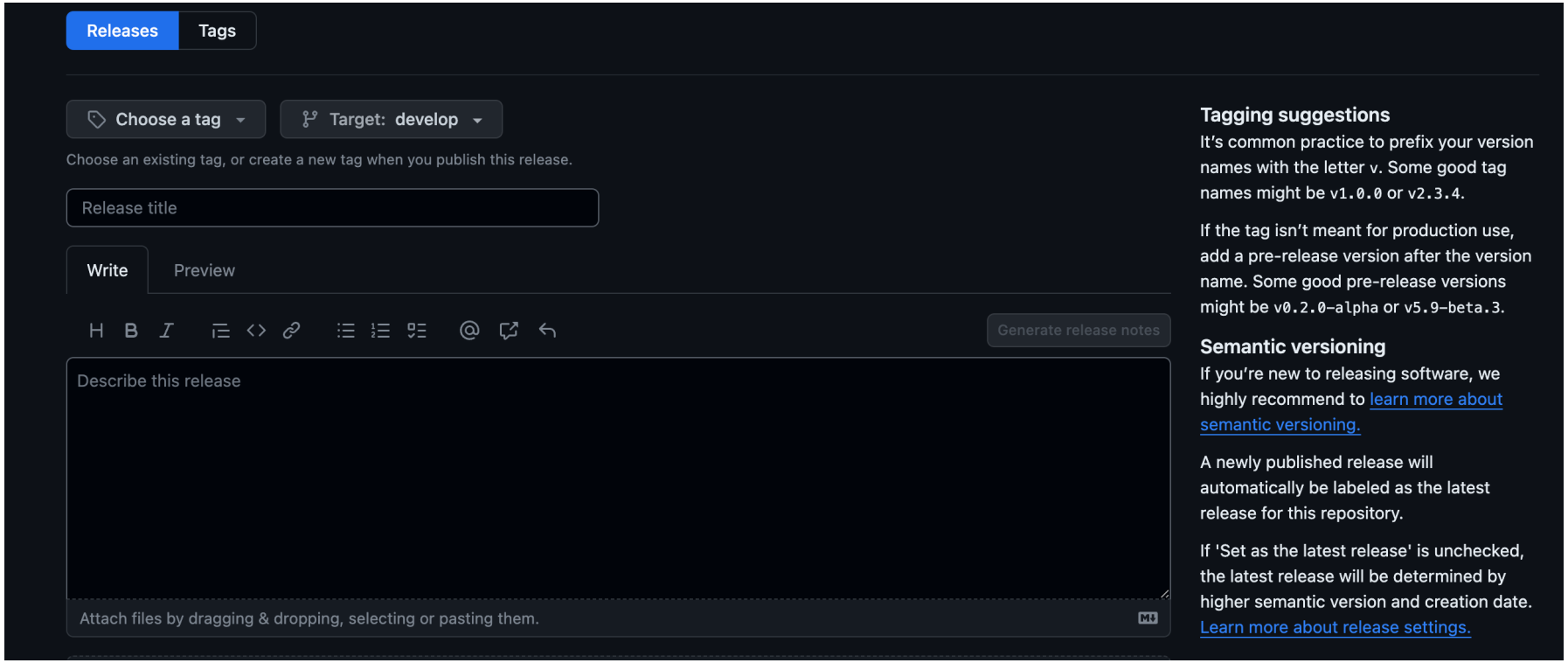
What's Changed

- Yaml refactor by @AdvancedImagingUTSW in #31
- Cam view by @codeCollision4 in #35
- Add multi_position controller by @annie-xd-wang in #40
- Camera attempt by @AdvancedImagingUTSW in #43
- Refactored experiment.yml by @annie-xd-wang in #42
- First pass with laser base by @AdvancedImagingUTSW in #44
- Model in process test merge by @AdvancedImagingUTSW in #51
- Model in process by @annie-xd-wang in #50
- Swampland 2 by @AdvancedImagingUTSW in #57
- Evaluation of Widget List by @AdvancedImagingUTSW in #60

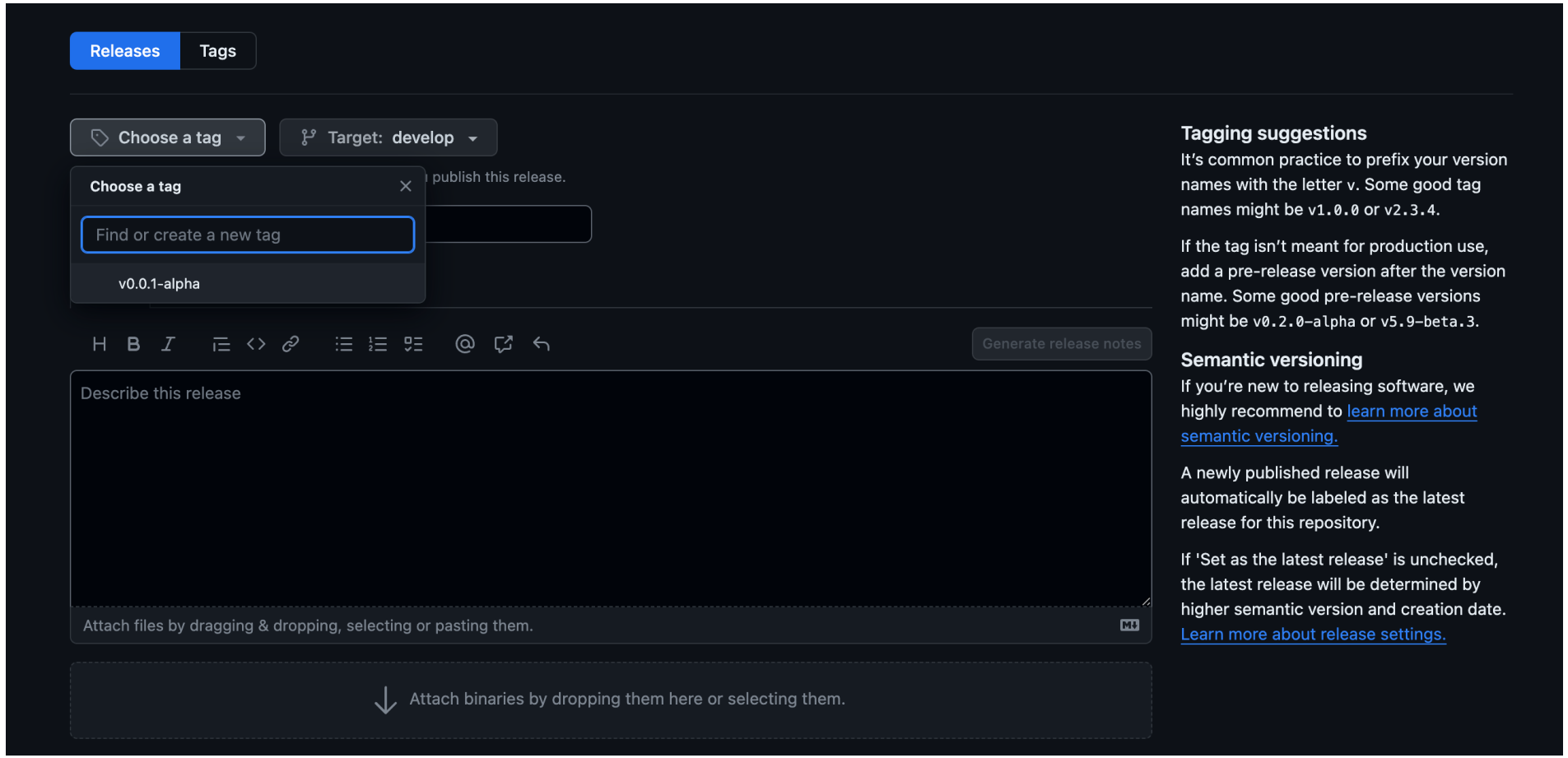
heDeanLab/ASLM/releases/latest

Drafting a stable release using Github

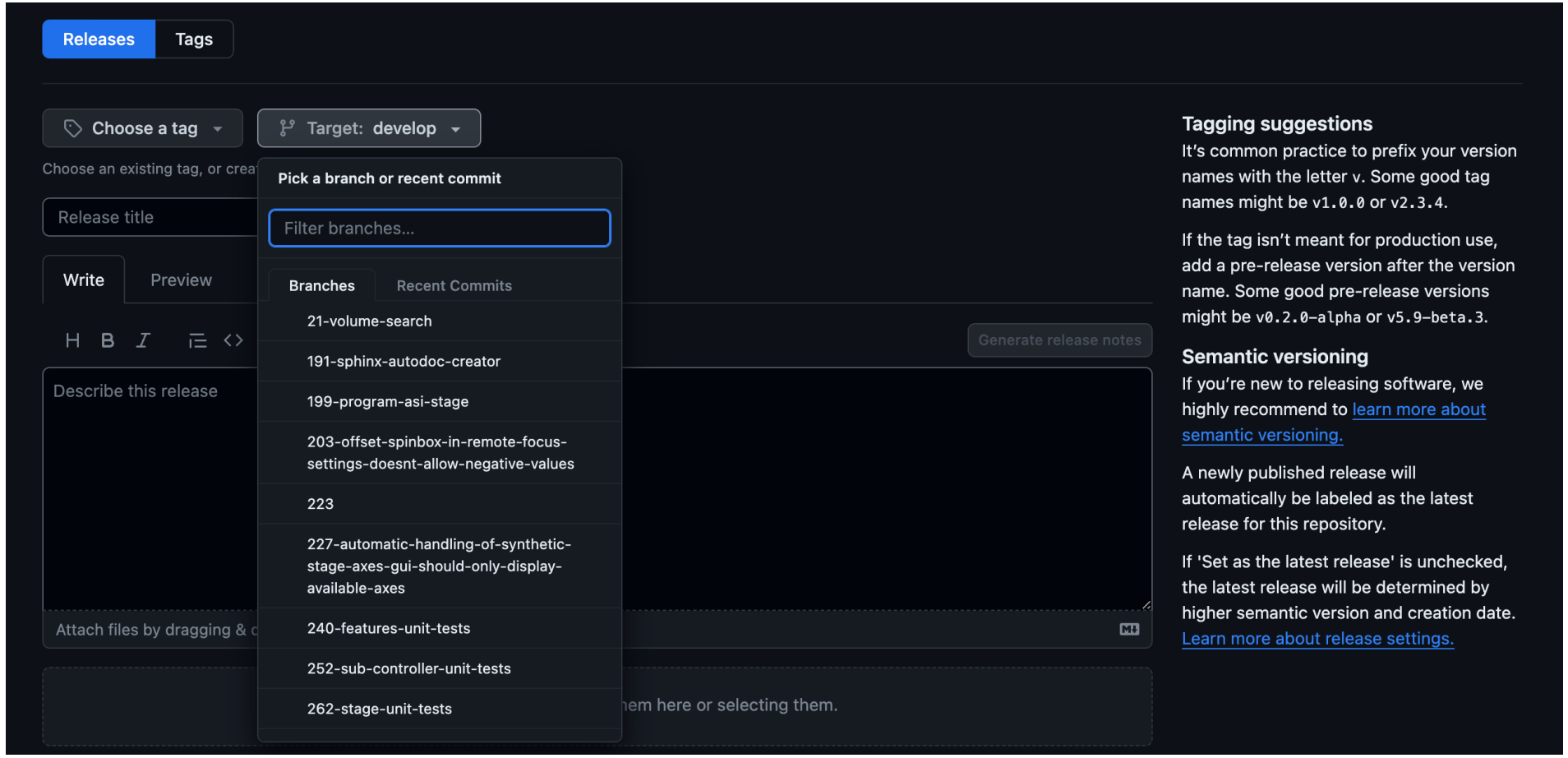
- Click draft stable release using Github



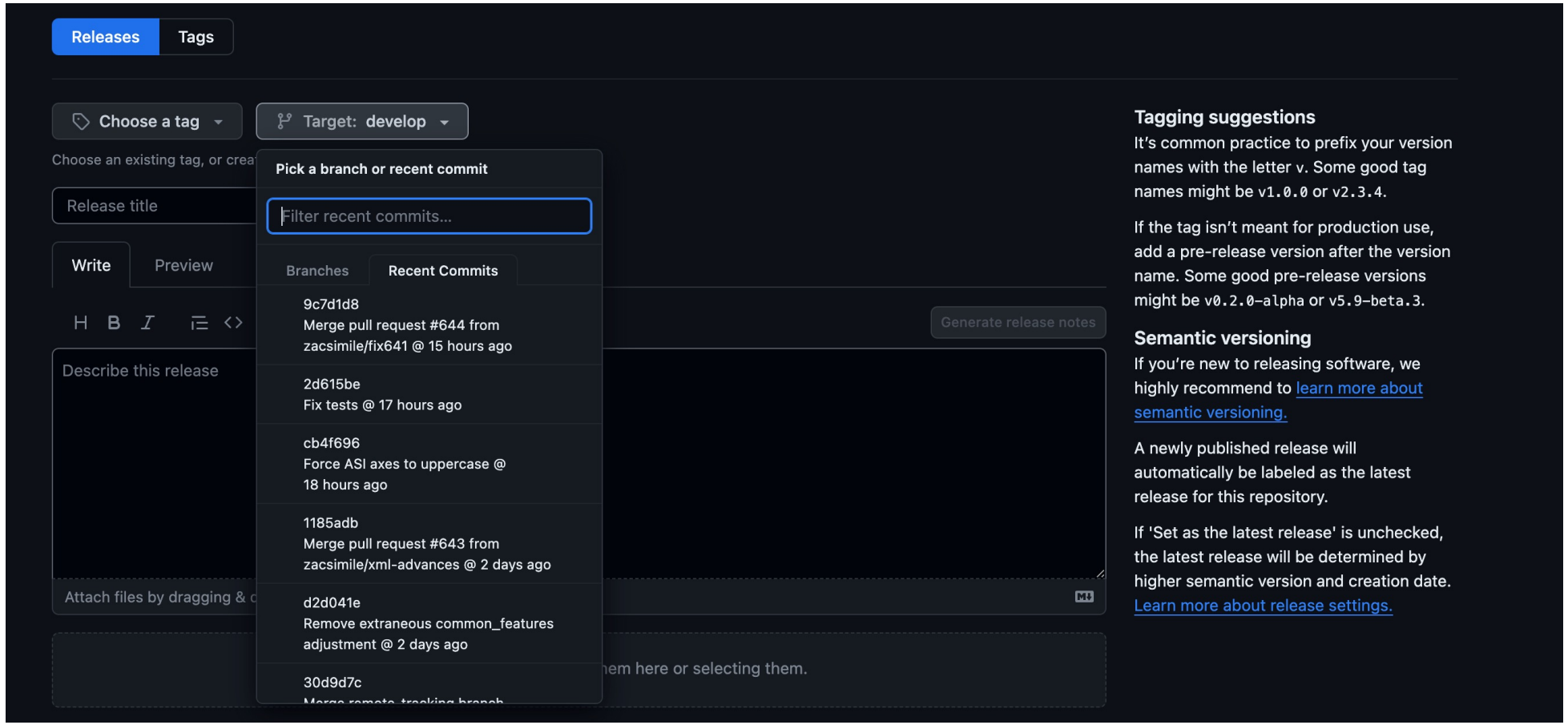
Creating a stable release from a tag using Github



Creating a stable release from a branch

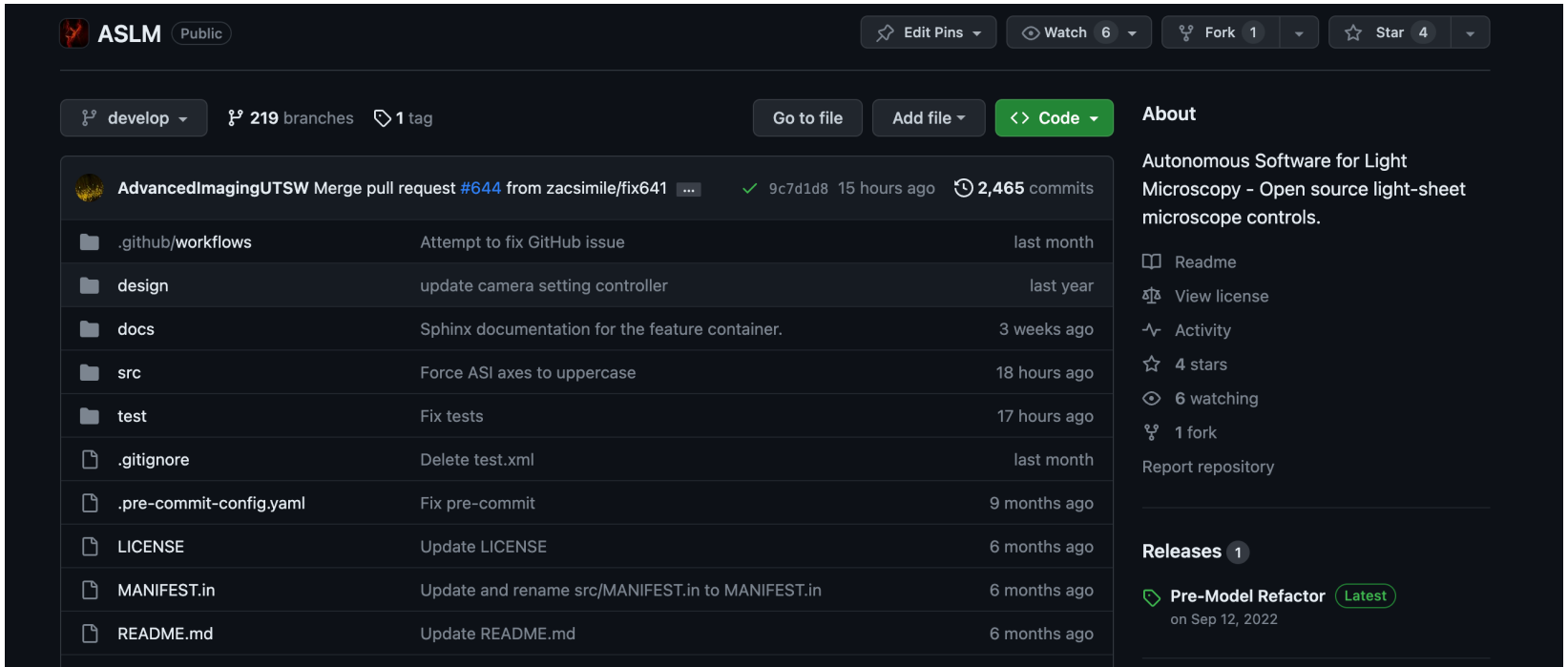


Creating a stable release from a commit



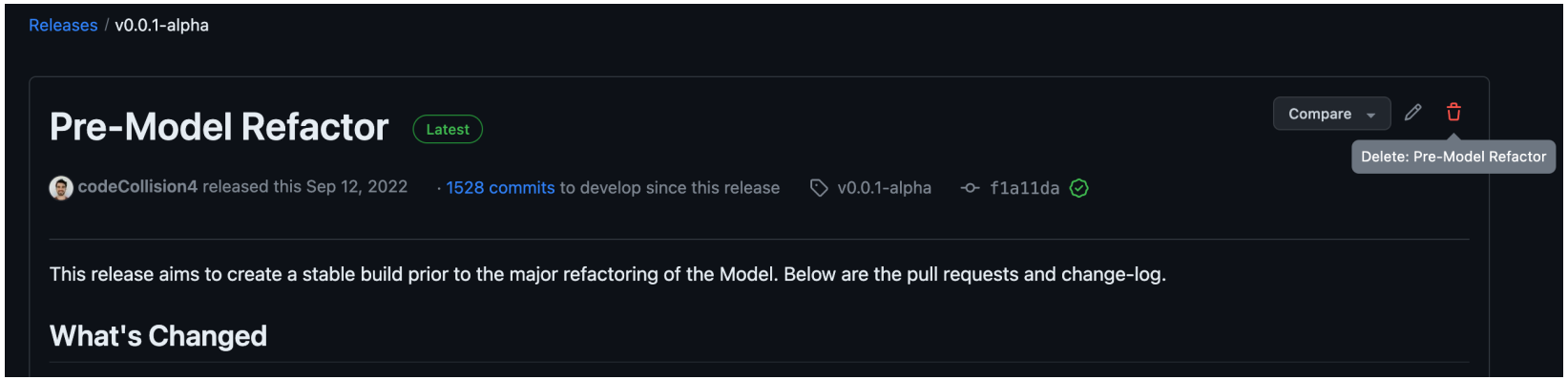
Stable Release

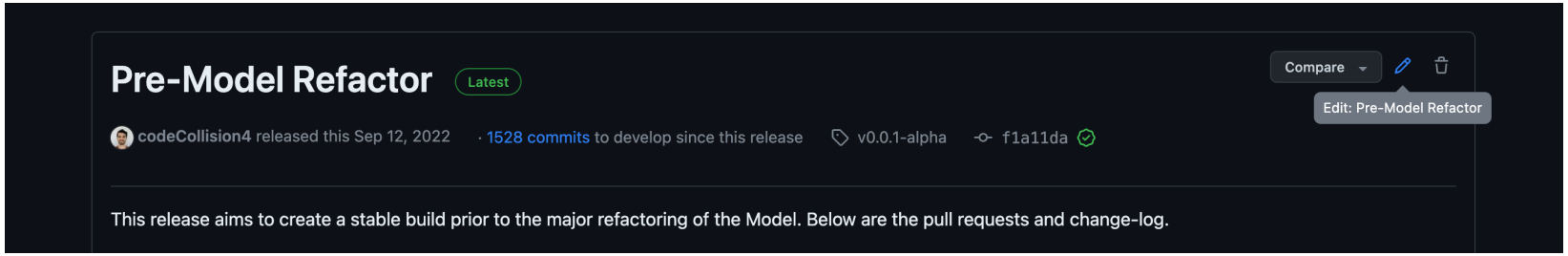
- Stable releases are shown on the main github page and will link to that branch
- Make sure versions of dependences are also included in the pyproject.toml file and operating systems are defined.
- Release will come with a link that can be used for citation



Editing or Deleting Stable Release

- It is possible to edit or delete a stable release by editing the tag or changing the fix.



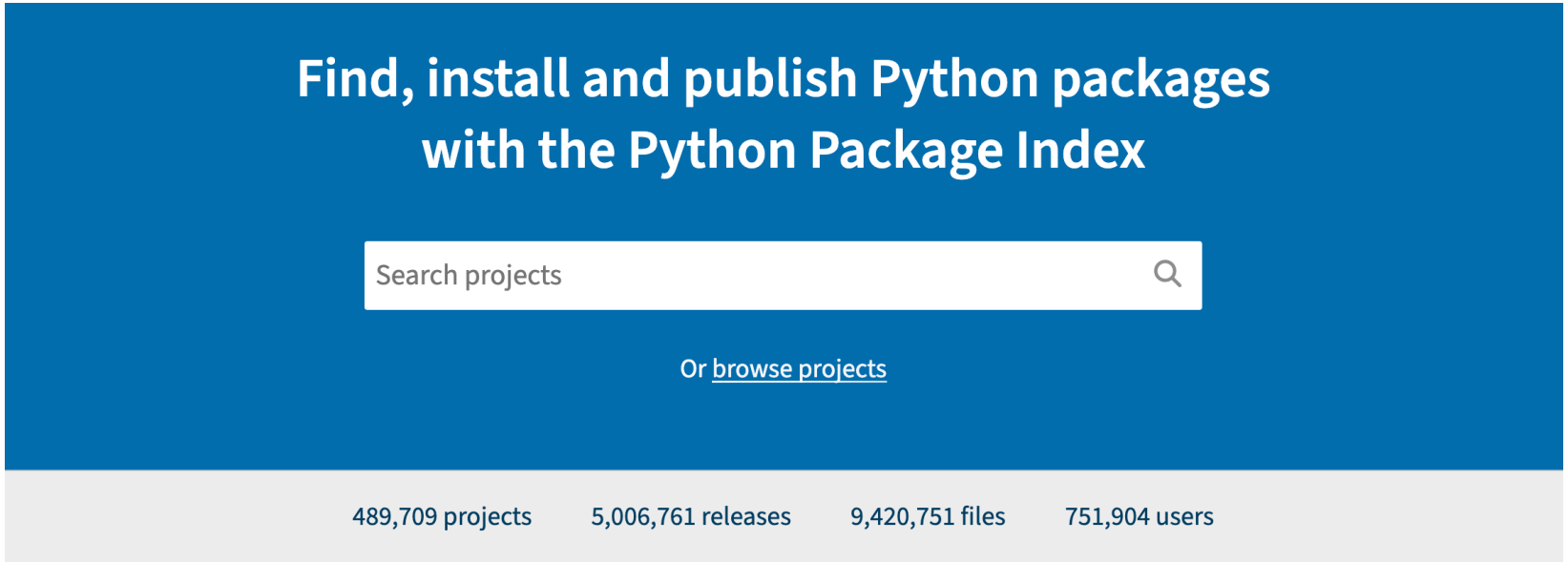


Downloading Release from Github

- To download latest version of software package, we can use
  - pip install “git+https://github.com/user/project”
- To download a stable release:
  - Pip install “git+https://github.com/user/project1.2.3”
- Include stable release as dependency in pyproject.toml:
  - Dev = [“project @ git+https//github.com/user/project1.2.3”]
- Zip files of releases can be created and can be used for faster install. For stable release:
  - Pip install “https://github.com/user/project1.2.3.zip”
- For dependency in pyproject.toml:
  - Dev = [“project @ https//github.com/user/project1.2.3.zip”]

Python Packaging Index (PyPI)

- We have used PyPI to install python packages before
- PyPI is a package management server that can can be used
- PyPI is a package management server to find install and publish packages



Publishing Packages using PyPI and Twine

- Step 1: Create a Readme.md file
- Step 2: Create a license file
- Step 3: Organize files in src and tests folder
- Step 4: buiding and versioning in pyproject.toml file
- Step 5: building wheel
- Step 6: pushing to PyPI
- Step 7: automatically publish releases using Github Actions

Step 1: Create a README.md file

- Create a README.md file that includes
  - Description of Package
  - How to install package
  - How to use package
  - Authors
- Include README.md in pyproject.toml
- Include citation in README.md for academic research

```md
# pyqt5-calc

## Simple calculator based on MVC pattern with Python and PyQt5 library.

Calculator based on tutorial from [Real Python](https://realpython.com/):  [Python and PyQt: Building a GUI
Desktop Calculator](https://realpython.com/python-pyqt-gui-calculator/). I updated the implementation to split the
application into separate [modules](https://docs.python.org/3/tutorial/modules.html) to follow the [MVC
(model-view-controller)](https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos)
pattern used in the tutorial, with those modules further placed into a package.

## Installation

For a development install, enter `pip install -e .[dev]`.
```

Step 2: Create a license file

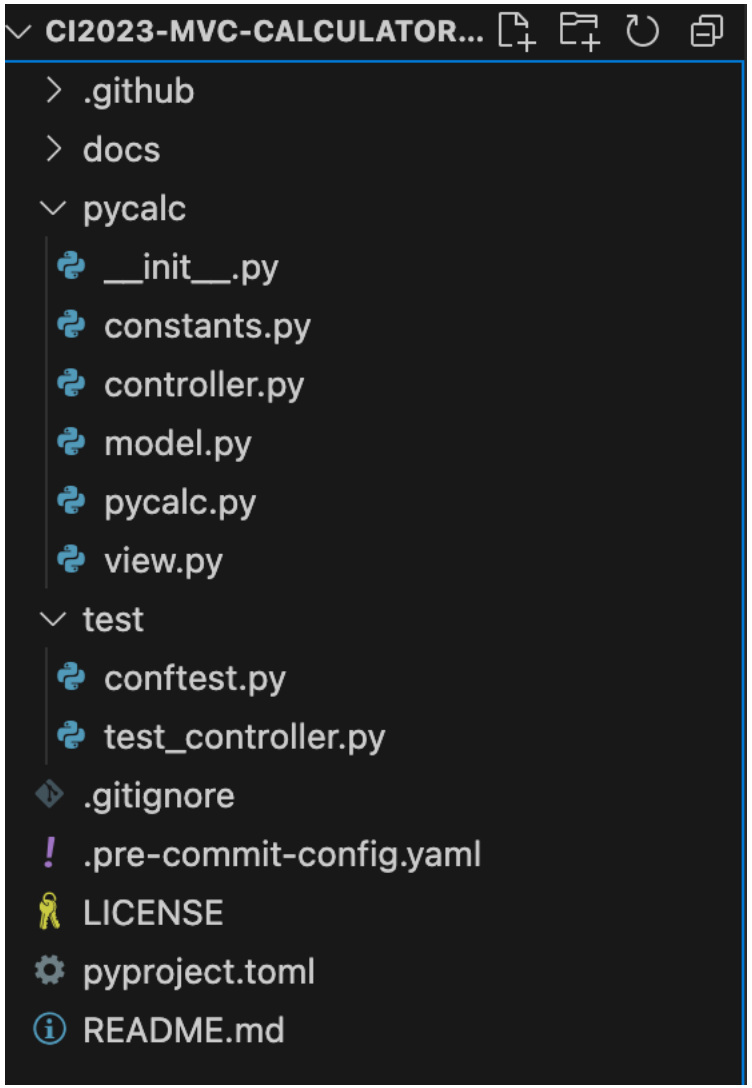
- Create a license file that includes rights for usage or redistribution
- Important for code distribution
- Use UTSW license file.
- Include license in pyproject.toml

```
MIT License

Copyright (c) 2021 TimothyDJones

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

```
[project]
name = "pycalc"
description = "An MVC calculator made in Qt."
authors = [{name = "Tim Jones"}]
readme = "README.md"
license = {file = "LICENSE.md"}
dynamic = ["version"]
```

Step 3: Organize files in src and tests folder

- Organize files into test and src folders similar to first calculator exercise
- Make sure that README.md, pyproject.toml, and license file are at the top level



Step 4: building and versioning in pyproject.toml file

- Versions of code are defined in 3 locations
  - Git Tag
  - Version field in pyproject.toml
  - __version__ in __init__.py
- By using setuptools.scm it will be possible to sync all three
- Add setuptools to build section of pyproject.toml
- Add set dynamic to version in project section of pyproject.toml

```
[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"
```

```
[project]
name = "pycalc"
description = "An MVC calculator made in Qt."
authors = [{name = "Tim Jones"}]
readme = "README.md"
license = {file = "LICENSE.md"}
dynamic = ["version"]
```

Step 5: Building wheel

- With using pyproject.toml, build is included so we can use the function to build a wheel
  - Python –m build
- Without using pyproject.toml, we would need to install build using
  - Pip install build
  - Python -m build
- Wheel file will be created in dist folder within repo
- Check wheel file by going to directory and unzipping file
  - Cd dist
  - Unzip project.whl

```
dist
|
|____ epi_models-0.1.0-py3-none-any.whl
|____ epi_models-0.1.0.tar.gz
```

Step 6: publishing wheel to PyPI using twine

- Install twine to convert packages to wheels to push to PyPI
  - Pip install twine
- Check package with twine
  - twine check dist/*
- Register for PyPI and possibly TestPyPI
  - https://pypi.org/
  - https://test.pypi.org/ (check if package works)
- Install into testPyPI
  - twine upload --repository testpypi dist/*
  - pip install -i https://test.pypi.org/pypi/
  - Wheel file will be created in dist folder within repo
- Install into PyPI
  - twine upload dist/*
  - Check if package works by using
    - pip install project

Step 7: automatically publish releases using Github Actions

- Use Github actions to automatically publish a package when a new release is setup in github
- Specify Yaml file to build package into wheel file
- Use github actions to publish package of a specific release instead of using twine.

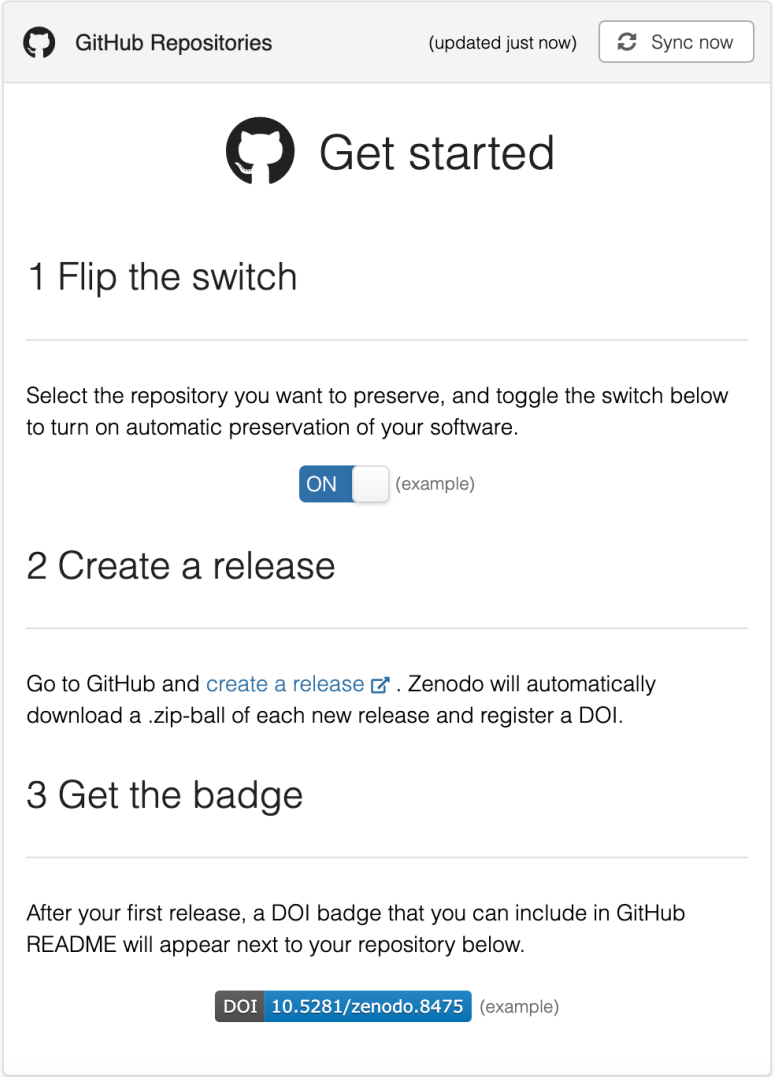
```yaml
# Provide a name for the workflow
name: Upload Python Package

# Tell GitHub when to run the action
# This will run every time a new release is published
on:
  release:
    types: [published]

jobs:
  deploy:
    # Run the workflow on the latest Ubuntu version
    runs-on: ubuntu-latest
    steps:
      # This will checkout our GitHub project and enter
      # the directory
      - uses: actions/checkout@v3
      # This will set up a Python environment
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.x'
      # Here we update pip to the latest version and
      # install 'build'. We won't need 'twine' here.
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install build
      # Here we run build to create a wheel and a
      # .tar.gz source distribution.
      - name: Build package
        run: python -m build --sdist --wheel
      # Finally, we use a pre-defined action to publish
      # our package in place of twine.
      - name: Publish package
        uses: pypa/gh-action-pypi-publish@release/v1
        with:
          user: __token__
          password: ${{ secrets.PYPI_API_TOKEN }}
```

Notes on Versioning

- Versioning is an important part of stable releases
  - Different stable release versions can alter functionality of code.
- Versions of stable releases need to be defined just like packages when listing dependencies
- Versions below 1 often signify that a repo is going through constant updates between versions and is considered unstable
  - Very little to no notification of changes before release is published
- Versions 1 or above signify a stable release constant updates between versions and is considered unstable
  - Functions which are being phased out in newer versions might be tagged with a depreciation warning so users know that this function will be removed or updated in a new released

Archiving using Github and Zenodo

- For publications, we would need a stable DOI that shows what code was used to facilitate the study
- Often, we adapt a repo to generate specific plots for a paper
- You don't want to place all of the code you used to create the plots into the repo
  - You might also have datasets you want to include that are larger than the limit allowed by github (~50 gb) or not necessary for the repo.
- Go to Zenodo Github page and login with github account
- Authorize zenodo access to github
- Follow Zenodo instructions to create a stable doi for a release in a repository



Conclusions

- Creating stable releases of code is a useful method to share codebases with others
  - Helps with reproducibility
- It is possible to create releases with github tags, releases, and creating packages on PyPI
- Create stable DOIs for publication using zenodo.

Exercise

- Build a wheel and create a stable release of codebase using PyPI and twine

Further Reading

- Creating tags and releases in github: https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository
- Creating packages with PyPI: https://carpentries-incubator.github.io/python_packaging/instructor/05-publishing.html#versions-and-releases
- Create stable releases with zenodo: https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content
- Conda Constructor: https://anaconda.org/anaconda/constructor, https://github.com/conda/constructor
- Creating packages from releases with github actions: https://docs.github.com/en/packages/managing-github-packages-using-github-actions-workflows/publishing-and-installing-a-package-with-github-actions
- Verisioning packages: https://setuptools.pypa.io/en/latest/userguide/pyproject_config.html