A Bird's Eye View of Contributing to and Maintaining Open Source Software

"Pom", Photo Credit: @mdegiacomi

MDAnalysis UGM 24 London, UK

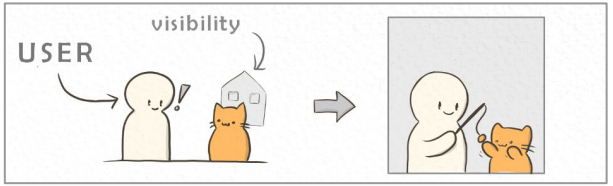
23 August 2024

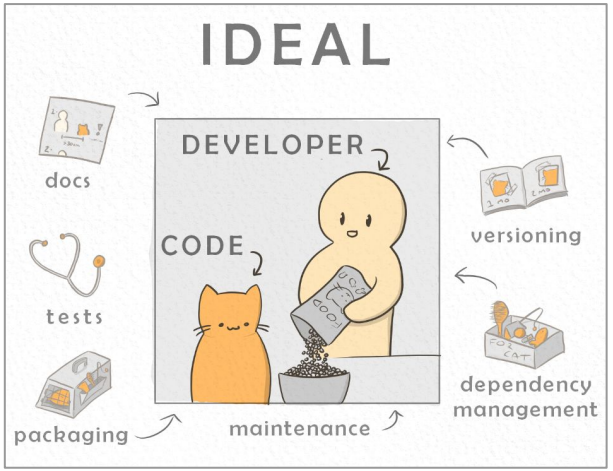
Fiona Naughton

Oliver Beckstein

Scientific code should meet the FAIR tenets

- **F**indable
- **A**ccessible
- **I**nteroperable
- **R**eusable

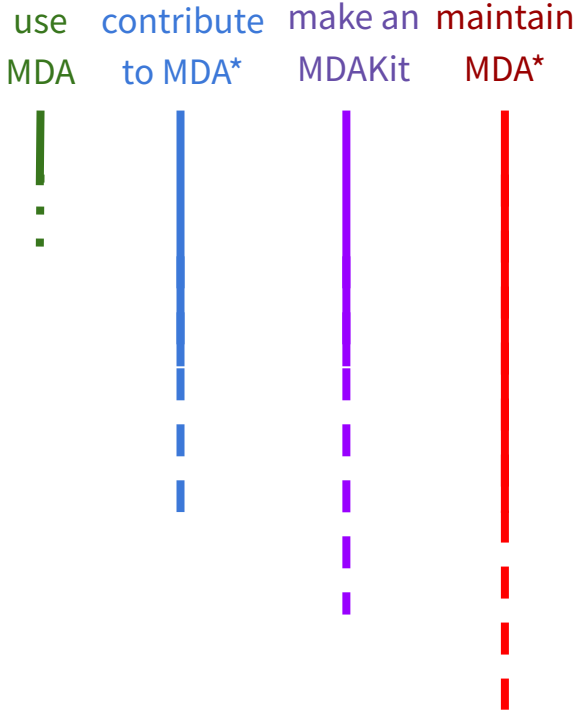




The ‘curriculum’ for FAIR code

What should I learn? In what order?

- Coding in the **relevant language** (e.g., Python, C/C++, …)
- **Version control** (git) and **distributed software development**, including (GitHub/GitLab, Pull Requests)
- **Testing** (pytest)
- **Documentation** (restructured text, sphinx)
- **Packaging** and **releases** (versioning, conda-forge, pypi)
- **Continuous integration “CI”** (GitHub actions)
- **Non-code contributions**! (User Guide, workshops/teaching material, community, …)



*or other Open Source project!

The ‘curriculum’ for FAIR code

How can I learn to do these things?

- It’s OK not to know everything, but good to know where/how to learn

MDAnalysis Contributing Guide

https://userguide.mdanalysis.org/stable/contributing.html

MDAnalysis/MolSSI Workshop

https://github.com/MDAnalysis/MDAnalysisMolSSIWorkshop-Intermediate2Day

Other Resources

MolSSI

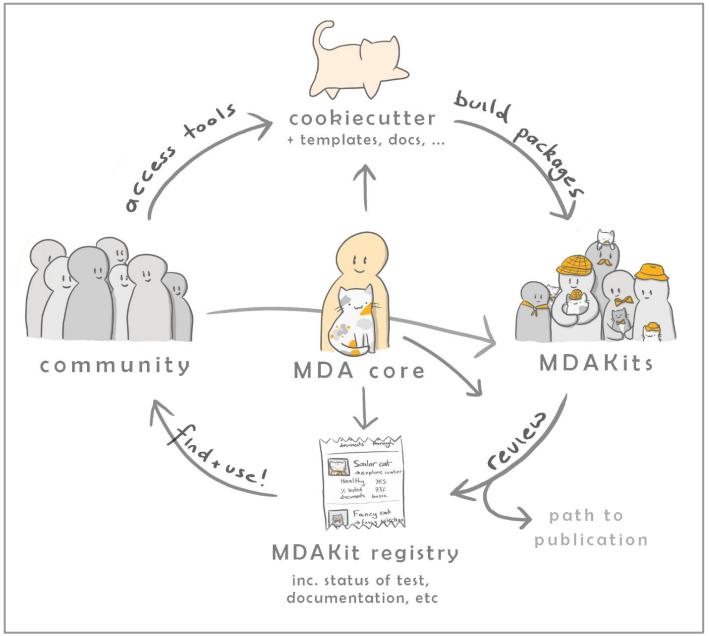
software carpentry

MDAKits (MDAnalysis toolkits)!

Encouraging and providing resources for the community to build FAIR-compliant code

https://mdakits.mdanalysis.org/

- **Cookiecutter template** provided that sets up files for tests, documentation, versioning, etc
- Toolkits that meet (minimal) requirements are promoted on the **MDAKit registry**
  - Users can check here for analysis packages
  - Developers can find users and collaborate with each other
  - Continual development of Kits is encouraged!



Slide adapted from: Irfan Alibay, https://github.com/MDAnalysis/MDAnalysisMolSSIWorkshop-Intermediate2Day/blob/jun24-ws/notebooks/MDAKitsIntro.pdf

Even with (or especially with?) a tool like the MDAKit CookieCutter to set things up, it's ok to be daunted by all of this!

Let's take a quick look through some of these concepts.

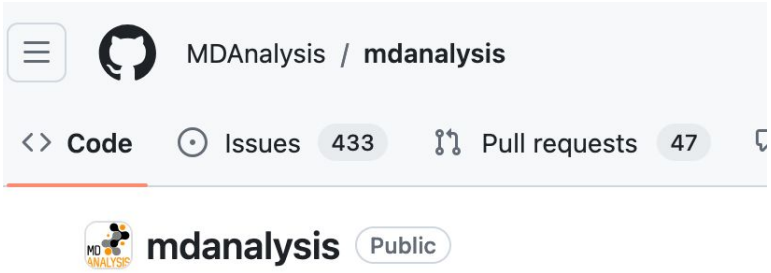
MDAKITS

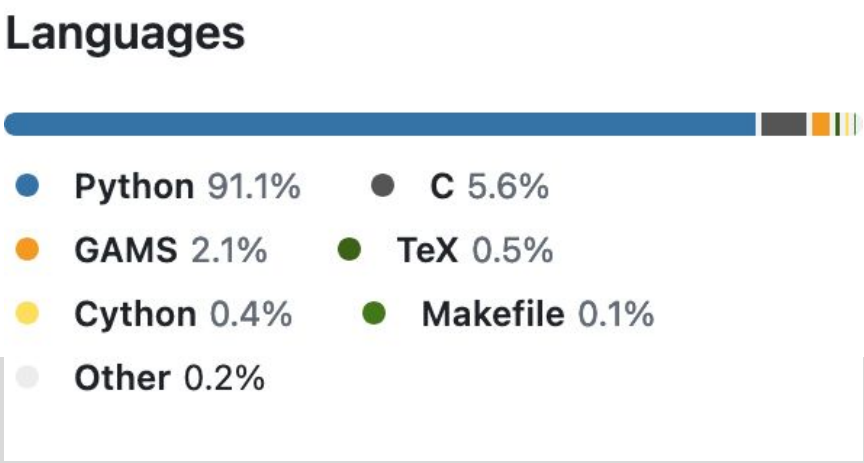
https://mdakits.mdanalysis.org/

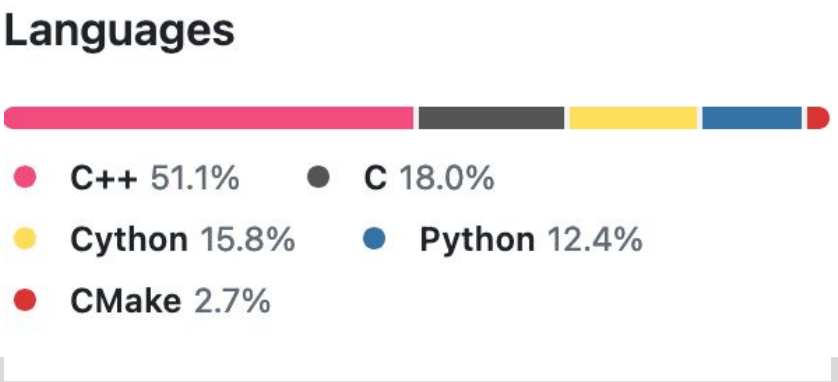
Coding in a relevant language

Know your language

Python / cython / C / C++







Using Virtual (Python) Environments

What is it?

- An *environment* is a collection of installed software/packages
- Using *Virtual Environments* allows you to have and switch between multiple, independent environments, with differents sets of packages (or different versions of the same packages) installed in each

Why is it important?

- Keeps the environment ‘clean’ (only the required stuff)
- Allows you to e.g. work on and test a development version of MDAnalysis, without it interfering with the stable version

How is it done?

- E.g. using conda/mamba or virtualenv

Follow a Style Guide

What is it?

- A style guide (such as the PEP 8 Guide for Python) provides a set of standard conventions for how code should be written/laid out
    - Maximum line lengths, where to put whitespace, etc

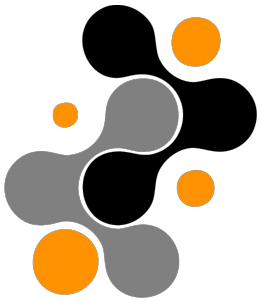
Why is it important?

- Keeps the code readable and easy to follow for both your future self and others

How is it done?

- Tools such as flake8 (for PEP 8) can scan your code and check for compliance

Distributed version control



Distributed version control

What is it?

- Software to keep track of changes in files
- Management of merging code from different sources

Why is it important?

- Enables multiple developers to work together on a common code base.
- Enables tracking of changes (e.g. to pinpoint fixes or introductions of bugs)

How is it done?

- SCM (source code management) tools like **git** (or mercurial)
- Store a database of changes in a *repository*
- Cloud repository providers (GitHub, GitLab, BitBucket, …)

Distributed version control

git — source code management & version control system

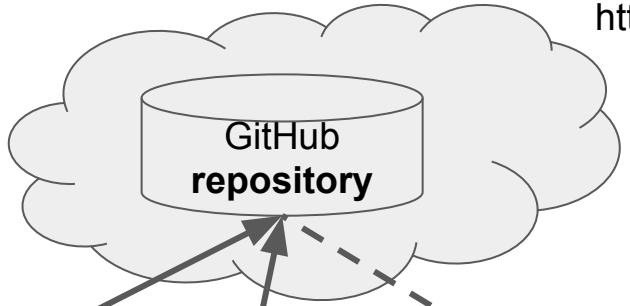

```
git add FILE_1 FILE_2 …

git commit -m "commit message"
```

```
git branch shiny

git switch shiny

…

git switch main

git merge shiny
```

Distributed version control

GitHub — cloud repository provider

https://github.com/



`git pull`

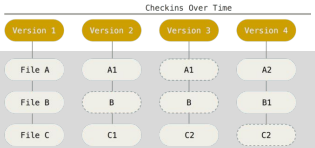
`git push`

```
git clone
https://github.com/MDA
nalysis/mdanalysis.git

git pull
```

Fiona's laptop
repository

Oliver's
workstation
repository

Neo's
repository

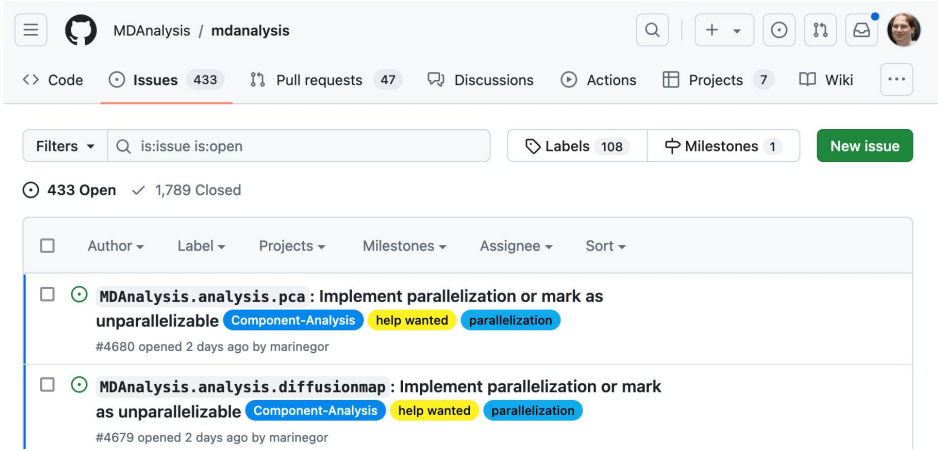


Distributed version control

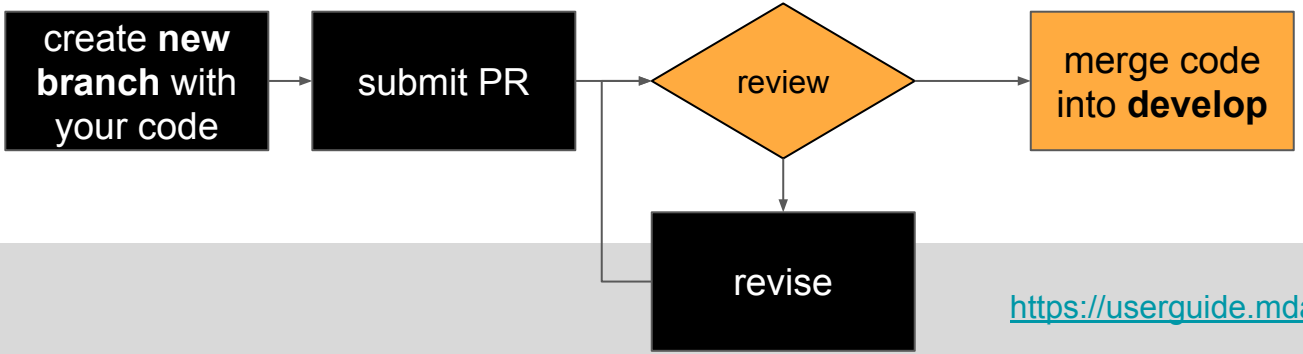
GitHub — code and project management

- **issues**
- **pull requests (PR)**
- discussion forum
- browsable history
- documentation hosting (gh pages)
- …

https://github.com/MDAnalysis/mdanalysis/issues

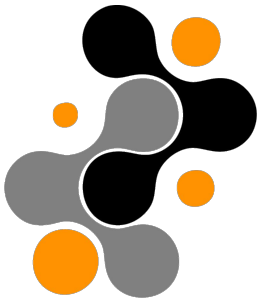


PR – review – merge development model



https://userguide.mdanalysis.org/stable/contributing.html

Documentation



Documentation

What is it? (levels of documentation)

- **README**: a file containing a basic overview describing the package and how it is used
- **Docstrings**: Text written in the code that describes what each piece of code does
  - E.g. docstrings for functions/methods describe what the function does, what the inputs are (and their data types), and what outputs are returned (and Errors that may be raised)
- **API documentation** provides a more technical description of the code - e.g. the docstrings
- **User Guide**: A more detailed and accessible explanation of the software package and how it can be used
- Detailed tutorials and examples can also be provided separately

Documentation

Why is it important?

- Without docstrings, even you will forget what your code is doing
- API docs help developers and contributors know what they can do; User Guides are more friendly to newcomers

Documentation

How is it done?

- READMEs: often written in markdown language, and stored in the base directory.
  - GitHub formats markdown files nicely for online display!
- Docstrings - write as you go!
  - MDAnalysis uses NumPy-style docstrings. These are written in ReStructuredText
  - Sphinx
  - Readthedocs
- User Guide: the MDAnalysis UserGuide is also a repository (separate from the code) so that changes in the Guide can also be tracked

Documentation

Restructured text (reST) with sphinx

```
class Universe(object):
    """The MDAnalysis Universe contains all the information describing the system.
    ...
    Parameters
    ----------
    topology: str, stream, Topology, numpy.ndarray, None
        A CHARMM/XPLOR PSF topology file, PDB file or Gromacs GRO file; used to
        ...
    coordinates: str, stream, list of str, list of stream (optional)
        Coordinates can be provided as files of a single frame (eg a PDB,
        ...

    .. versionchanged:: 2.5.0
        Added fudge_factor and lower_bound parameters for use with
        *guess_bonds*.
    """
    def __init__(self, topology=None, *coordinates, all_coordinates=False,
                 format=None, topology_format=None, transformations=None,
                 guess_bonds=False, vdwradii=None, fudge_factor=0.55,
                 lower_bound=0.1, in_memory=False,
                 in_memory_step=1, **kwargs):
        self._trajectory = None  # managed attribute holding Reader
        self._cache = {'_valid': {}}
        ...
```

Restructured text primer
https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

sphinx docs (read their User Guide)
https://www.sphinx-doc.org

MDA User Guide: Working with the docs
https://userguide.mdanalysis.org/stable/contributing_code.html#working-with-the-code-documentation

What do ***you*** consider most important for **scientific code**?

1. **Performance / speed**
2. **Userfriendliness**
3. **Correctness**
4. **Free to use**
5. **Open source**

Testing (and Continuous Integration)

Testing and Continuous Integration

What is it?

- **Unit Tests**: tests that check a small unit of code (e.g. a single function) behave as expected
- **Continuous Integration (or CI)**: continual checking that the software still works (e.g. can be successfully installed, tests pass) as both the software code itself is changed and upstream dependencies (e.g. Python, NumPy, …) are changed

Testing and Continuous Integration

How is it done?

- Unit Tests (e.g. using pytest)
  - A lot of code means a lot of tests! The MDAnalysis testssuite is a whole package of its own, following the same structure as the code
  - A common test is to feed a set of dummy inputs to a function/method of the package and check (or ‘assert’) that the output matches the provided expected values
  - Tests should also cover cases of ‘bad’ input - ie. that the expected Errors are raised
- Code coverage is a measure of how much “code” is covered by tests
  - Automated - codecov
- Continuous Integration
  - GitHub actions can be used to trigger automatic running of tests e.g. on each push, or at regular intervals

Testing code

pytest

- Labor intensive!
- Absolutely necessary
- MDAnalysis merges code *only if tests are included*

```python
# content of test_sample.py
def inc_by_2(x):
    """Increment x by 2"""
    return x + 1


def test_answer():
    assert inc_by_2(3) == 5
```

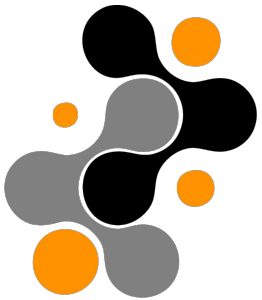
https://docs.pytest.org/

pytest

```
-bash                                                    ⌥⌘2
(base) deathstar:foo oliver$ pytest
============================= test session starts ==============================
platform darwin -- Python 3.11.7, pytest-7.3.1, pluggy-1.0.0
rootdir: /Users/oliver/tmp/foo
plugins: timeout-2.1.0, anyio-3.6.2
collected 1 item

test_sample.py F                                                         [100%]

================================= FAILURES =================================
_____________________________ test_answer ______________________________

    def test_answer():
>       assert inc_by_2(3) == 5
E       assert 4 == 5
E        +  where 4 = inc_by_2(3)

test_sample.py:8: AssertionError
=========================== short test summary info ============================
FAILED test_sample.py::test_answer - assert 4 == 5
============================== 1 failed in 0.11s ===============================
(base) deathstar:foo oliver$
```

https://userguide.mdanalysis.org/stable/contributing_code.html#testing-your-code

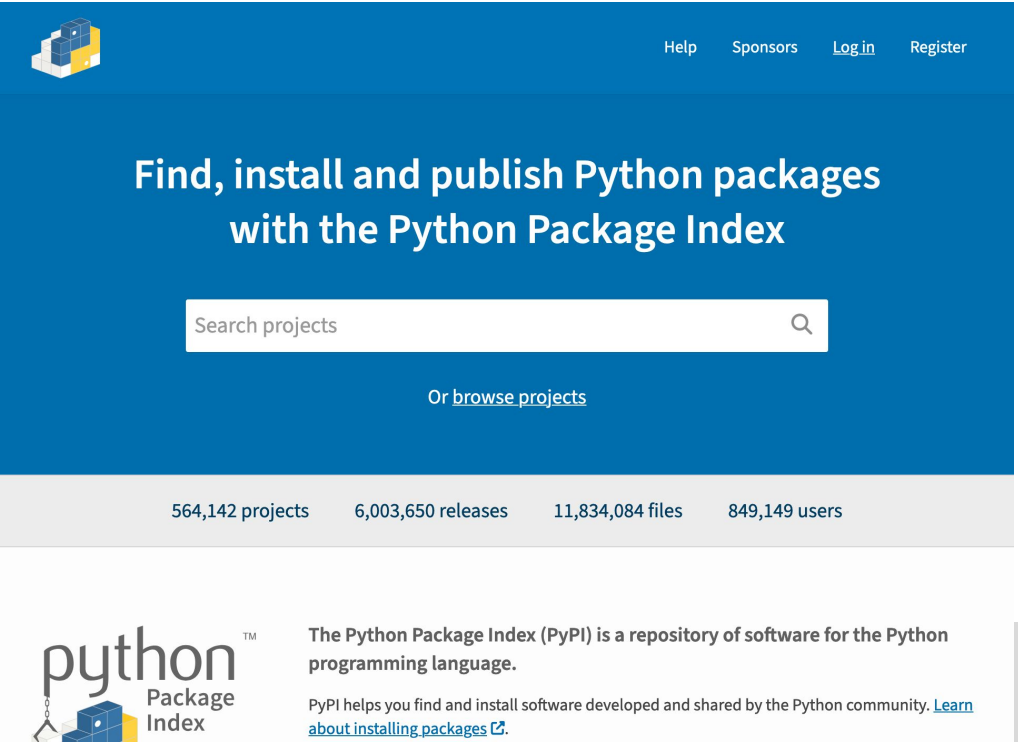
Packaging



Where users see “packages”: distribution platforms

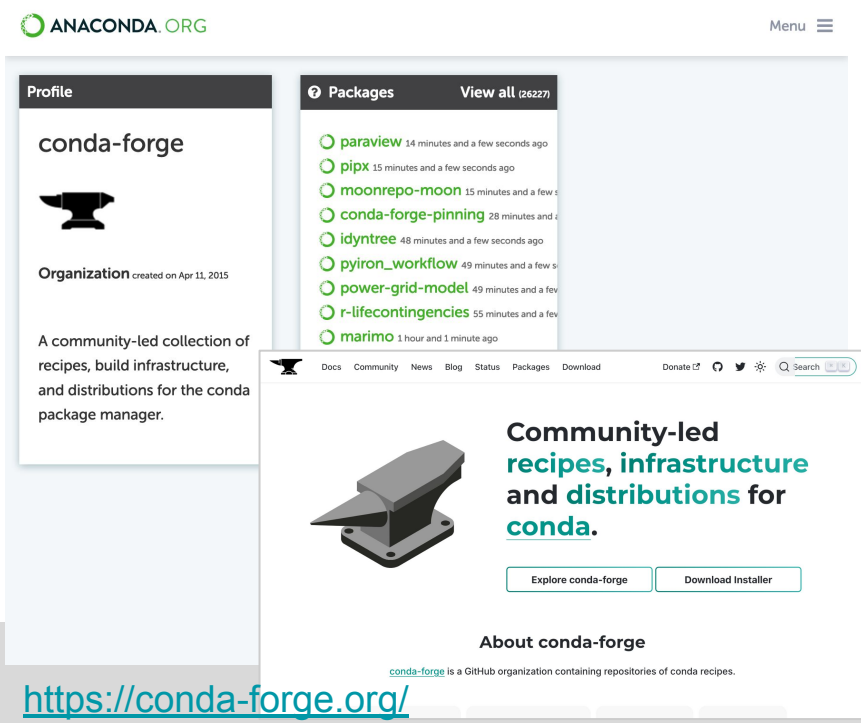
PyPi

https://pypi.org/



conda/conda-forge

https://anaconda.org/conda-forge/



https://conda-forge.org/

Package

What is it?

- Installable and distributable unit of code
  - Versioned (release number)
  - Includes executables and libraries
- Dependency declaration

```
pip install mdanalysis
```

Why is it important?

- Provides everything that the user needs to make correct use of the code
- Links code into the wider ecosystem (*dependencies*)

How is it done?

- Python package: `__init__.py`
- Installable package: `build` (pyproject.toml) or conda/conda-forge

https://docs.python.org/3/tutorial/modules.html#packages

https://packaging.python.org/

https://conda-forge.org/docs/

License

What is it?

- Legal document that states how others may use your code.
- Explicit statement of copyright ownership

Why is it important?

- Without a license, **nobody may use your code**
- “Open Source” requires an open-source license
- Enables others to build on your work but you set the terms

How is it done?

- Include LICENSE file in your code repository
- Add copyright and license information to source code

Pick an existing license, don’t invent your own**!!!!!**

https://choosealicense.com/

“An open source license protects contributors and users. Businesses and savvy developers won’t touch a project without this protection.”

Dependencies

What is it?

- List of all *other non-standard packages* that your code is using
- Include minimally required version numbers or forbidden versions

Why is it important?

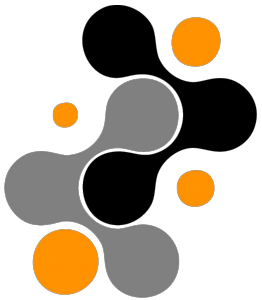
- package manager can *resolve dependencies* to ensure that
  - Dependencies are available (download & install automatically)
  - Bad dependencies do not crash your code or lead to subtle errors
- Users want the convenience of simple package manager installation

How is it done?

- pypi: pyproject.toml
- conda: meta.yml

```
requires-python = ">=3.10"
dependencies = [
    'numpy>=1.23.2',
    'GridDataFormats>=0.4.0',
    'mmtf-python>=1.0.0',
    'joblib>=0.12',
    'scipy>=1.5.0',
    'matplotlib>=1.5.1',
    'tqdm>=4.43.0',
    'threadpoolctl',
    'packaging',
    'fasteners',
    'mda-xdrlib',
    'waterdynamics',
    'pathsimanalysis',
    'mdahole2',
]
```

Deployment and releases



Making a release

What is it?

- A release is state of the software package that users can be directed to for installation
  - A **stable release** (should) be free of major bugs
  - **Nightly builds** can be run automatically + use the latest code, but might be buggy

- **Semantic versioning** is a way of assigning numbers to releases to indicate the changes introduced since the previous version; MAJOR.MINOR.PATCH where:
  - The MAJOR number increments when there is an incompatible API change (ie. the way the code is used changes)
  - The MINOR number increases when functionality is added but backwards-compatibility is maintained
  - The PATCH number increases when bug fixes are made
  - Until release, the current in-development version has ‘-dev0’ appended

Making a release

How is it done? (e.g. how does MDAnalysis do it)

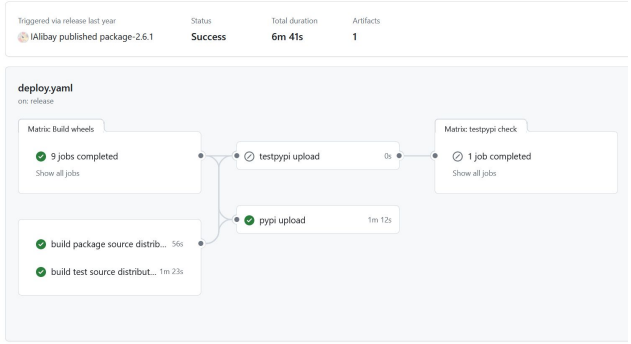
- https://userguide.mdanalysis.org/stable/preparing_releases_and_hotfixes.html

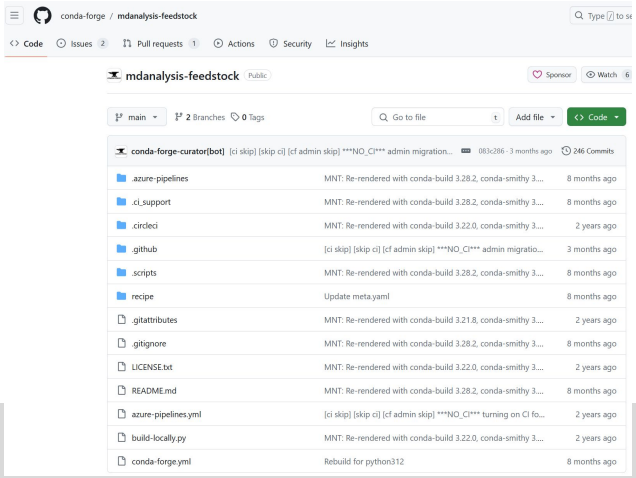
- When code is ready for a release:
  - Make a *git tag* (a ‘snapshot’ of a branch)
  - Intermediate step: a test pypi upload is triggered. Wait and hope there’s no errors. Depends on phase of moon.
  - (skipping some extra steps required due to our Cython files)
  - Make a *release* from the tag using GitHub
    - GitHub can auto-generate a release message based on the changes since the last release; this can be edited

Making a release

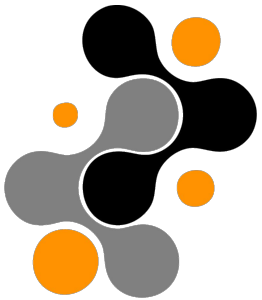
How is it done? (e.g. how does MDAnalysis do it)

- Making a release available
  - Thanks to GitHub Actions, the new version is automagically uploaded to PyPi
  - This is then detected by conda-forge, and a PR is made in the conda-forge/mdanalysis-feedstock repo
    - Check, update minimum versions etc as necessary, and more hoping for the right phase of the moon - merge when all good





… so what was it I need to know again?



Contributing to MDAnalysis

E.g. fix an issue, add feature, add to the User Guide

- Git and GitHub:
  - Fork and Clone a repo
  - Create a new branch and commit changes
  - Open a pull request
- Write your contribution!
  - Likely in Python (PEP 8)
  - Docstrings as appropriate (ReST) + check docs build
  - Tests as appropriate + check tests run
- The framework for building docs/running tests is there - you just need to make sure it still runs!

1. Fork the MDAnalysis repository from the mdanalysis account into your own account
2. Set up an isolated virtual environment for code development
3. Build development versions of MDAnalysis and MDAnalysisTests on your computer into the virtual environment
4. Create a new branch off the develop branch
5. Add your new feature or bug fix or add your new documentation
6. Add and run tests (if adding to the code)
7. Build and view the documentation (if adding to the docs)
8. Ensure PEP8 compliance (mandatory) and format your code with Darker (optional)
9. Commit and push your changes, and open a pull request.

https://userguide.mdanalysis.org/stable/contributing_code.html

Making/Registering an MDAKit

Requirements

- Uses MDA
- OSI-approved license
- Versioned and in a version-controlled repository
- List of Authors/maintainers
- Some form of minimal documentation (e.g. README)
- Minimal tests

Ideal, but not required

- More detailed documentation
- Package on distribution platform (PyPi, conda-forge)
- An issue tracker/place for community discussion

https://mdakits.mdanalysis.org/

To **register a Kit**, create a Pull Request in the MDAKits repository with a metadata.yaml file containing basic information like the name and description of the Kit, where the code is located, authors, steps for installation, steps for running tests, link to documentation, …

Making/Registering an MDAKit

The CookieCutter can set up the framework

https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?

```
example-repository
├── .codecov.yml
├── .gitattributes
├── .github
│   ├── ISSUE_TEMPLATE
│   │   ├── bug_report.md
│   │   └── feature_request.md
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows
│       └── gh-ci.yaml
├── .gitignore
├── .pre-commit-config.yaml
├── .pylintrc
├── AUTHORS.md
├── CHANGELOG.md
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── LICENSE
├── MANIFEST.in
├── README.md
├── devtools
│   └── conda-envs
│       └── test_env.yaml
```

```
├── docs
│   ├── Makefile
│   ├── README.md
│   ├── make.bat
│   ├── requirements.yaml
│   └── source
│       ├── _static
│       │   ├── README.md
│       │   └── logo
│       │       ├── mdakits-empty-favicon-template.svg
│       │       ├── mdakits-empty-logo-template.svg
│       │       └── mdakits-placeholder-logo.png
│       ├── _templates
│       │   └── README.md
│       ├── api.rst
│       ├── conf.py
│       ├── getting_started.rst
│       └── index.rst
├── package_name
│   ├── __init__.py
│   ├── analysis
│   │   ├── __init__.py
│   │   └── myanalysisclass.py
│   ├── data
│   │   ├── README.md
│   │   ├── __init__.py
│   │   ├── files.py
│   │   └── mda.txt
│   └── tests
│       ├── __init__.py
│       ├── analysis
│       │   ├── __init__.py
│       │   └── test_myanalysisclass.py
│       ├── conftest.py
│       ├── test_package_name.py
│       └── utils.py
├── pyproject.toml
└── readthedocs.yaml
```

Making/Registering an MDAKit

The CookieCutter can set up the framework

https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?
  - Style formatting

```
example-repository
├── .codecov.yml
├── .gitattributes
├── .github
│   ├── ISSUE_TEMPLATE
│   │   ├── bug_report.md
│   │   └── feature_request.md
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows
│       └── gh-ci.yaml
├── .gitignore
├── .pre-commit-config.yaml
├── .pylintrc
├── AUTHORS.md
├── CHANGELOG.md
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── LICENSE
├── MANIFEST.in
├── README.md
├── devtools
│   └── conda-envs
│       └── test_env.yaml
```

```
├── docs
│   ├── Makefile
│   ├── README.md
│   ├── make.bat
│   ├── requirements.yaml
│   └── source
│       ├── _static
│       │   ├── README.md
│       │   └── logo
│       │       ├── mdakits-empty-favicon-template.svg
│       │       ├── mdakits-empty-logo-template.svg
│       │       └── mdakits-placeholder-logo.png
│       ├── _templates
│       │   └── README.md
│       ├── api.rst
│       ├── conf.py
│       ├── getting_started.rst
│       └── index.rst
├── package_name
│   ├── __init__.py
│   ├── analysis
│   │   ├── __init__.py
│   │   └── myanalysisclass.py
│   ├── data
│   │   ├── README.md
│   │   ├── __init__.py
│   │   ├── files.py
│   │   └── mda.txt
│   └── tests
│       ├── __init__.py
│       ├── analysis
│       │   ├── __init__.py
│       │   └── test_myanalysisclass.py
│       ├── conftest.py
│       ├── test_package_name.py
│       └── utils.py
├── pyproject.toml
└── readthedocs.yaml
```

Making/Registering an MDAKit

The CookieCutter can set up the framework

https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?

```
example-repository
├── .codecov.yml
├── .gitattributes
├── .github
│   ├── ISSUE_TEMPLATE
│   │   ├── bug_report.md
│   │   └── feature_request.md
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows
│       └── gh-ci.yaml
├── .gitignore
├── .pre-commit-config.yaml
├── .pylintrc
├── AUTHORS.md
├── CHANGELOG.md
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── LICENSE
├── MANIFEST.in
├── README.md
├── devtools
│   └── conda-envs
│       └── test_env.yaml
```

- Style formatting
- Git things for Version Control
- GitHub configuration stuff

```
├── docs
│   ├── Makefile
│   ├── README.md
│   ├── make.bat
│   ├── requirements.yaml
│   └── source
│       ├── _static
│       │   ├── README.md
│       │   └── logo
│       │       ├── mdakits-empty-favicon-template.svg
│       │       ├── mdakits-empty-logo-template.svg
│       │       └── mdakits-placeholder-logo.png
│       ├── _templates
│       │   └── README.md
│       ├── api.rst
│       ├── conf.py
│       ├── getting_started.rst
│       └── index.rst
├── package_name
│   ├── __init__.py
│   ├── analysis
│   │   ├── __init__.py
│   │   └── myanalysisclass.py
│   ├── data
│   │   ├── README.md
│   │   ├── __init__.py
│   │   ├── files.py
│   │   └── mda.txt
│   └── tests
│       ├── __init__.py
│       ├── analysis
│       │   ├── __init__.py
│       │   └── test_myanalysisclass.py
│       ├── conftest.py
│       ├── test_package_name.py
│       └── utils.py
├── pyproject.toml
└── readthedocs.yaml
```

Making/Registering an MDAKit

The CookieCutter can set up the framework

https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?

```
example-repository
├── .codecov.yml
├── .gitattributes
├── .github
│   ├── ISSUE_TEMPLATE
│   │   ├── bug_report.md
│   │   └── feature_request.md
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows
│       └── gh-ci.yaml
├── .gitignore
├── .pre-commit-config.yaml
├── .pylintrc
├── AUTHORS.md
├── CHANGELOG.md
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── LICENSE
├── MANIFEST.in
├── README.md
├── devtools
│   └── conda-envs
│       └── test_env.yaml
```

- Style formatting
- Git things for Version Control
- GitHub configuration stuff
- Documentation: configuration things, doc contents,

```
├── docs
│   ├── Makefile
│   ├── README.md
│   ├── make.bat
│   ├── requirements.yaml
│   └── source
│       ├── _static
│       │   ├── README.md
│       │   └── logo
│       │       ├── mdakits-empty-favicon-template.svg
│       │       ├── mdakits-empty-logo-template.svg
│       │       └── mdakits-placeholder-logo.png
│       ├── _templates
│       │   └── README.md
│       ├── api.rst
│       ├── conf.py
│       ├── getting_started.rst
│       └── index.rst
├── package_name
│   ├── __init__.py
│   ├── analysis
│   │   ├── __init__.py
│   │   └── myanalysisclass.py
│   ├── data
│   │   ├── README.md
│   │   ├── __init__.py
│   │   ├── files.py
│   │   └── mda.txt
│   └── tests
│       ├── __init__.py
│       ├── analysis
│       │   ├── __init__.py
│       │   └── test_myanalysisclass.py
│       ├── conftest.py
│       ├── test_package_name.py
│       └── utils.py
├── pyproject.toml
└── readthedocs.yaml
```

Making/Registering an MDAKit

The CookieCutter can set up the framework

https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?

```
example-repository
├── .codecov.yml
├── .gitattributes
├── .github
│   ├── ISSUE_TEMPLATE
│   │   ├── bug_report.md
│   │   └── feature_request.md
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows
│       └── gh-ci.yaml
├── .gitignore
├── .pre-commit-config.yaml
├── .pylintrc
├── AUTHORS.md
├── CHANGELOG.md
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── LICENSE
├── MANIFEST.in
├── README.md
├── devtools
│   └── conda-envs
│       └── test_env.yaml
```

- Style formatting
- Git things for Version Control
- GitHub configuration stuff
- Documentation: configuration things, doc contents,
- Testing: configuration stuff, your actual tests, CI stuff

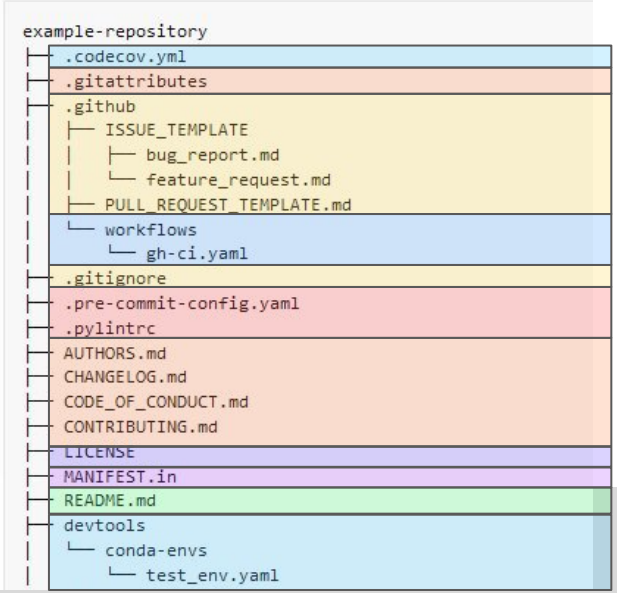
```
├── docs
│   ├── Makefile
│   ├── README.md
│   ├── make.bat
│   ├── requirements.yaml
│   └── source
│       ├── _static
│       │   ├── README.md
│       │   └── logo
│       │       ├── mdakits-empty-favicon-template.svg
│       │       ├── mdakits-empty-logo-template.svg
│       │       └── mdakits-placeholder-logo.png
│       ├── _templates
│       │   └── README.md
│       ├── api.rst
│       ├── conf.py
│       ├── getting_started.rst
│       └── index.rst
├── package_name
│   ├── __init__.py
│   ├── analysis
│   │   ├── __init__.py
│   │   └── myanalysisclass.py
│   ├── data
│   │   ├── README.md
│   │   ├── __init__.py
│   │   ├── files.py
│   │   └── mda.txt
│   └── tests
│       ├── __init__.py
│       ├── analysis
│       │   ├── __init__.py
│       │   └── test_myanalysisclass.py
│       ├── conftest.py
│       ├── test_package_name.py
│       └── utils.py
├── pyproject.toml
└── readthedocs.yaml
```

Making/Registering an MDAKit

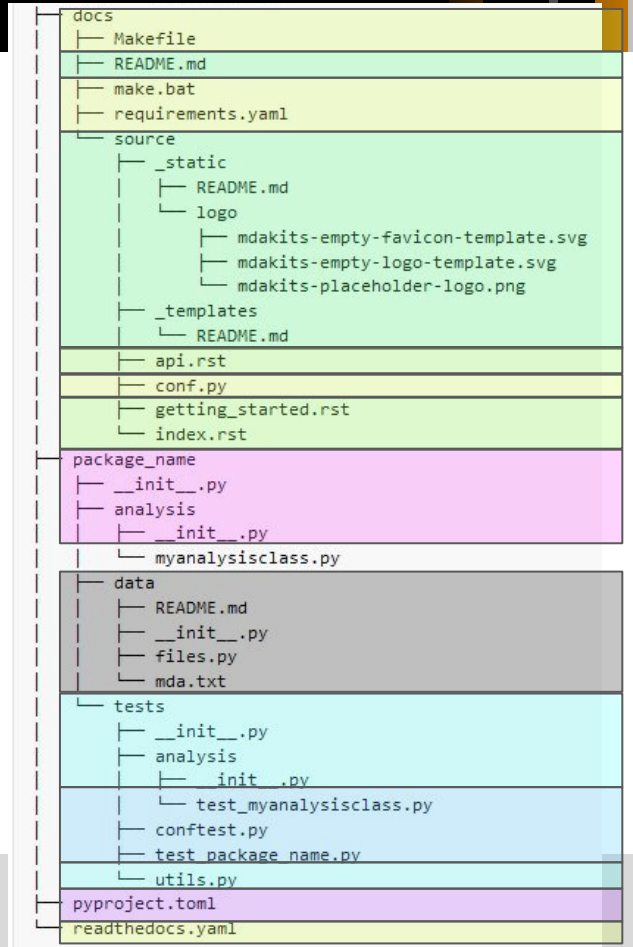
The CookieCutter can set up the framework

https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?



- Style formatting
- Git things for Version Control
- GitHub configuration stuff
- Documentation: configuration things, doc contents,
- Testing: configuration stuff, your actual tests, CI stuff
- Packaging etc



Making/Registering an MDAKit

The CookieCutter can set up the framework

https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?

```
example-repository
├── .codecov.yml
├── .gitattributes
├── .github
│   ├── ISSUE_TEMPLATE
│   │   ├── bug_report.md
│   │   └── feature_request.md
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows
│       └── gh-ci.yaml
├── .gitignore
├── .pre-commit-config.yaml
├── .pylintrc
├── AUTHORS.md
├── CHANGELOG.md
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── LICENSE
├── MANIFEST.in
├── README.md
├── devtools
│   └── conda-envs
│       └── test_env.yaml
```

- Style formatting
- Git things for Version Control
- GitHub configuration stuff
- Documentation: configuration things, doc contents,
- Testing: configuration stuff, your actual tests, CI stuff
- Packaging etc
- Your code!

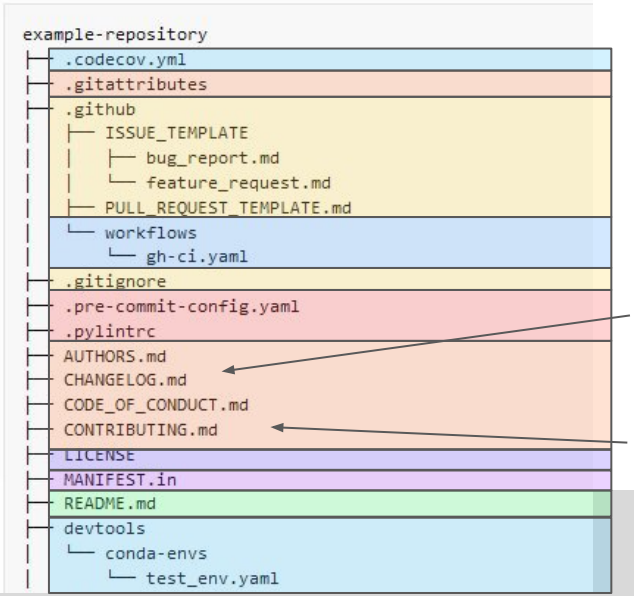
```
├── docs
│   ├── Makefile
│   ├── README.md
│   ├── make.bat
│   ├── requirements.yaml
│   └── source
│       ├── _static
│       │   ├── README.md
│       │   └── logo
│       │       ├── mdakits-empty-favicon-template.svg
│       │       ├── mdakits-empty-logo-template.svg
│       │       └── mdakits-placeholder-logo.png
│       ├── _templates
│       │   └── README.md
│       ├── api.rst
│       ├── conf.py
│       ├── getting_started.rst
│       └── index.rst
├── package_name
│   ├── __init__.py
│   ├── analysis
│   │   ├── __init__.py
│   │   └── myanalysisclass.py
│   ├── data
│   │   ├── README.md
│   │   ├── __init__.py
│   │   ├── files.py
│   │   └── mda.txt
│   └── tests
│       ├── __init__.py
│       ├── analysis
│       │   ├── __init__.py
│       │   └── test_myanalysisclass.py
│       ├── conftest.py
│       ├── test_package_name.py
│       └── utils.py
├── pyproject.toml
└── readthedocs.yaml
```

Making/Registering an MDAKit

The CookieCutter can set up the framework

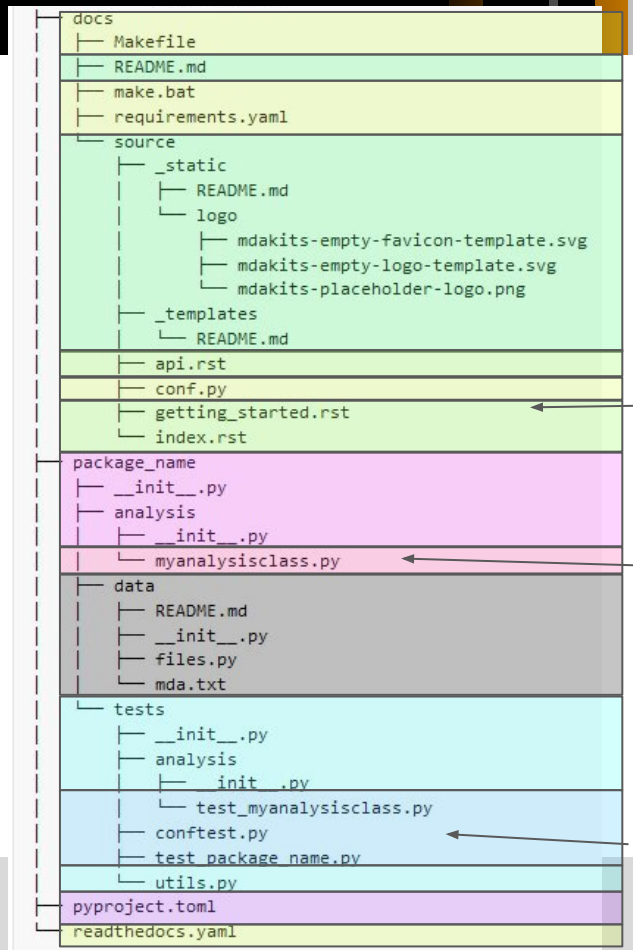
https://cookiecutter-mdakit.readthedocs.io/en/latest/usage.html

- Maybe some of this looks a little more familiar now?



Log changs here

Project info here



Docs here

Code goes here

Tests here

The ‘curriculum’ for FAIR code

How can I learn to do these things?

- It’s OK not to know everything, but good to know where/how to learn

MDAnalysis Contributing Guide

https://userguide.mdanalysis.org/stable/contributing.html

MDAnalysis/MolSSI Workshop

https://github.com/MDAnalysis/MDAnalysisMolSSIWorkshop-Intermediate2Day

Other Resources

MolSSI

software carpentry

The curriculum for contributing

How can I learn to do these things?

- It's OK not to know everything, but good to know w

Learn to work with people you don't know!

MDAnalysis Contributing Guide

https://userguide.mdanalysis.org/stable/contributing.html

MDAnalysis/MolSSI Works p

https://github.com/MDAnalysis/MDAnalysisMolSSIWorkshop-Intermediate2Day

Other Resources

MolSSI

software carpentry

Contributing can involve more than just code

participating/answering questions on forums (discord, GH discussions), X, bluesky, LinkedIn

documentation

MDAnalysis library (`MDAnalysis`)

GitHub.org/MDAnalysis/mdanalysis

open source

community

MD ANALYSIS

other software (data processing in molecular sciences)

GitHub.org/MDAnalysis

workshops (teaching and developing materials)

education & outreach

mentoring

Google Summer of Code

OUTREACHY

STATION

SOCIALLY-DIRECTED SCIENCE AND TECHNOLOGY

eco-system

Interoperability (RDKit, blender, …)

MDAKITS

MDAKits

mdakits.mdanalysis.org

Slide adapted from: Oliver Beckstein, https://github.com/MDAnalysis/MDAnalysisMolSSIWorkshop-Intermediate2Day/blob/jun24-ws/June%202024%20MDAnalysis-MolSSI%20Workshop%20Introduction%20Slides.pdf

Join an existing MDAnalysis team/initiative!

Learn more about what activities you can get involved with on the mdanalysis.org website

https://www.mdanalysis.org/pages/team/

MDAnalysis team

MDAnalysis is a community-driven project that is made possible through the efforts of many members who contribute in numerous and diverse ways, ranging from direct package development, maintenance, documentation, communication, and managerial responsibilities. On this page we list identified project roles and team members for each of those roles. We note that the listed roles on this page can differ significantly in scope and required effort.

We also invite community members to reach out to mdanalysis@numfocus.org or the current team members if they are interested in filling missing roles or joining an existing team!