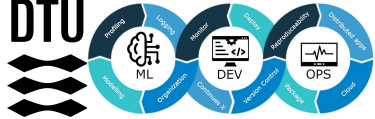


Continues Integration (CI)

02476 Machine Learning Operations

Nicki Skafte Detlefsen

Why you should care about today

Two years ago the day before this lecture, the internet went down for a couple of hours because someone f..ked up their CI.

Continues X

Term refers to a set of **software practises** for **automating** tedious tasks and make sure changes in a pipeline are **continuously propagated** through the pipeline.

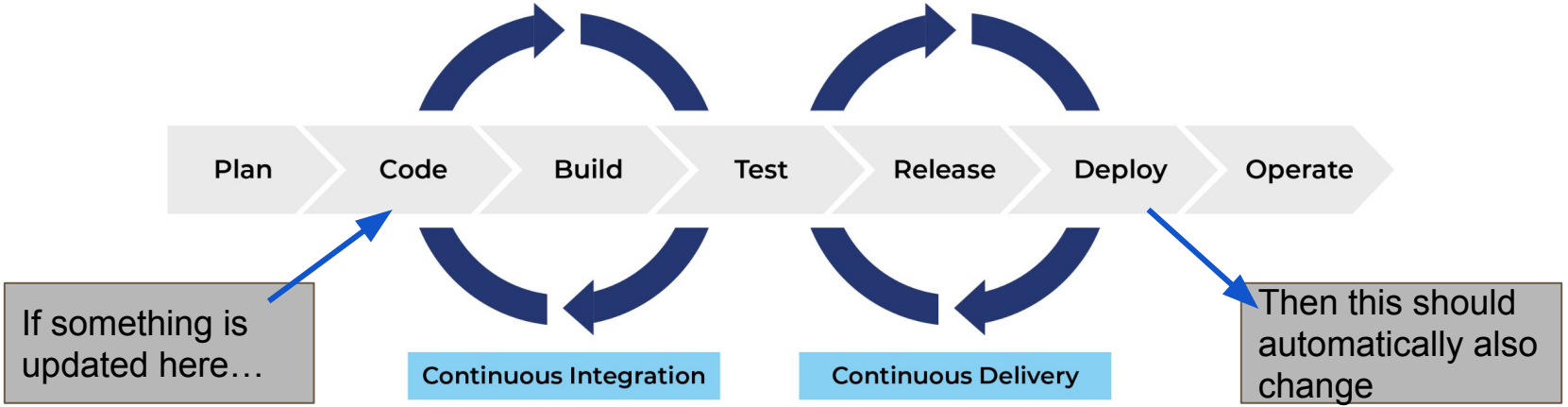
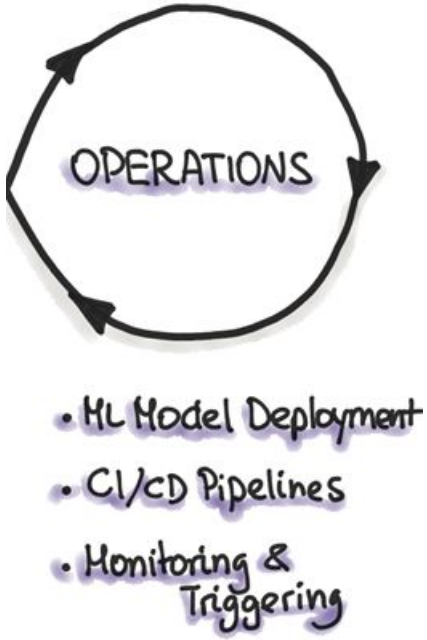
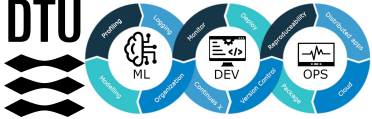


Figure credit: https://faun.pub/most-popular-ci-cd-pipelines-and-tools-ccfdce429867

CI vs CD vs CML

Continuous Integration	*Continuous Deployment*	*Continuous ML*
Core task:	Core task:	Core task:
How to automatically secure that code does not break during development?	How to get your code/ application to the end user automatically? + monitor life cycle	How to automatic retrain machine learning models when data changes?
App independent concept	App dependent concept	Specific to ML applications





This lecture: CI

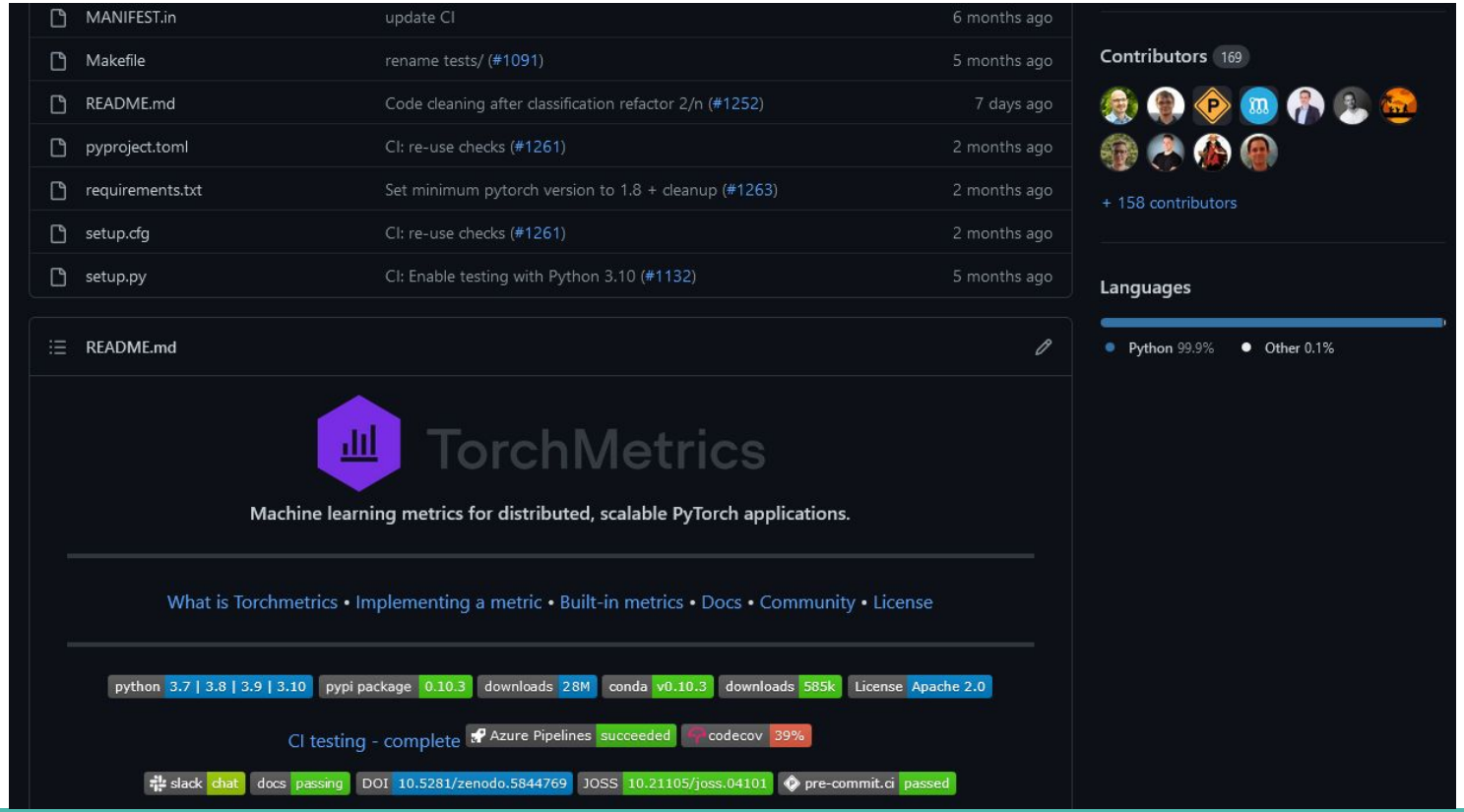
Core task:

How to automatically secure that code does not break during development?

3 steps to do this:

1. Use version control:
   - Frequently committing code to shared repository
2. Write (unit-)test for your code
   - Should capture unwanted bugs in your code
3. Automate build + test
   - Automatic run test so code cannot be merged without working

A small case study for CI



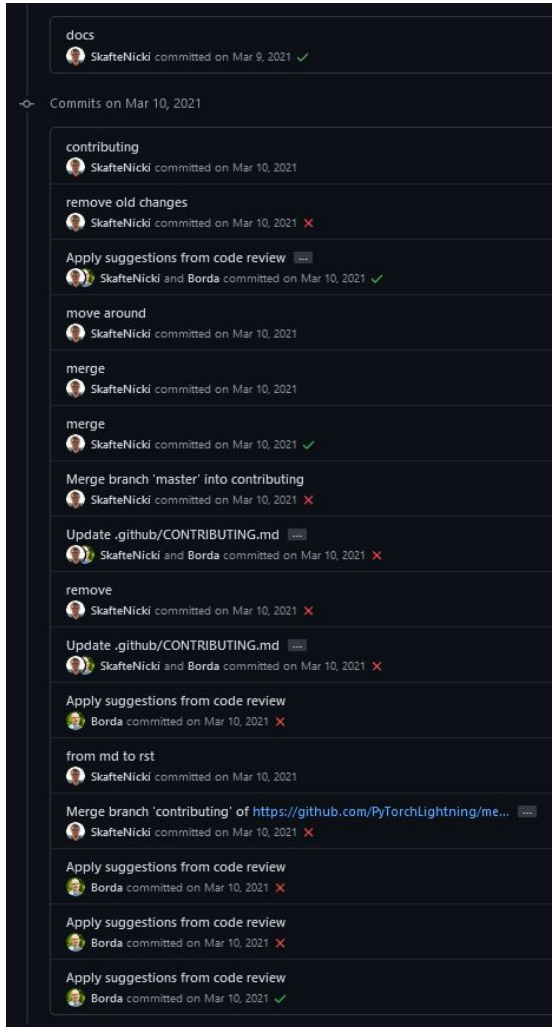
CI step 1: version control

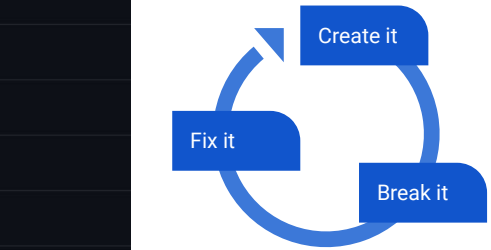
Use version control

- Code changes are tracked
- Branches for parallel work

Commit frequently

- Catch errors sooner than later
- Merging can be done automatically





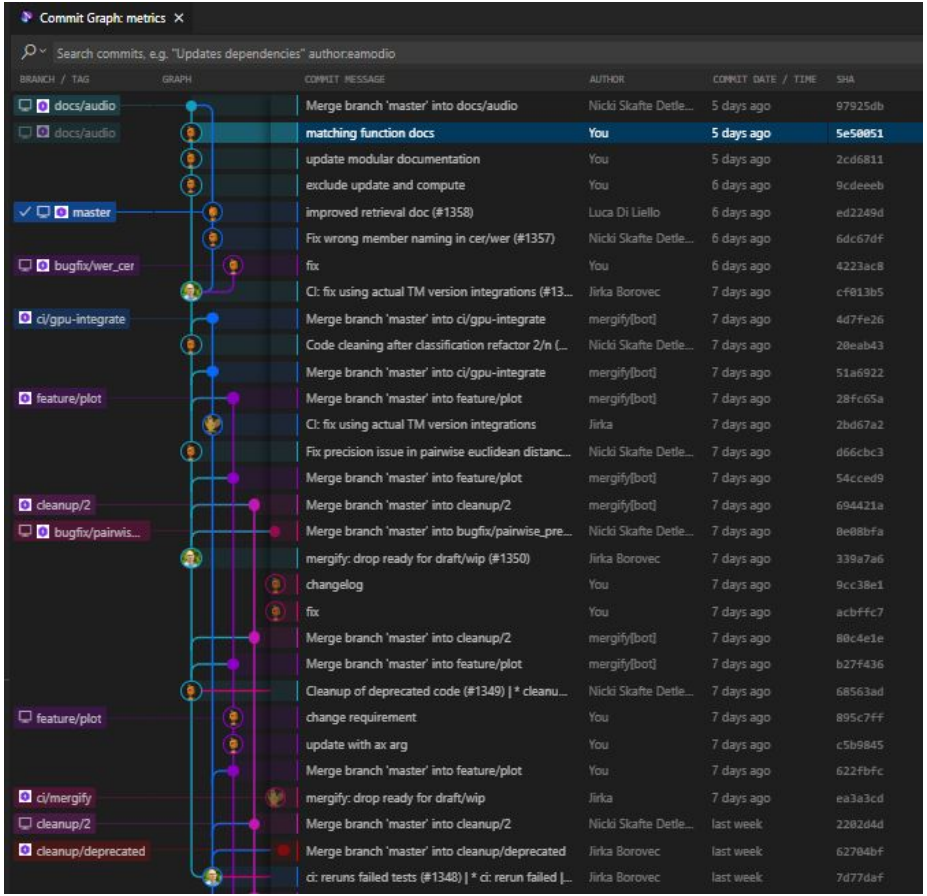
Ci step 1: Use branches

Parallel workflow

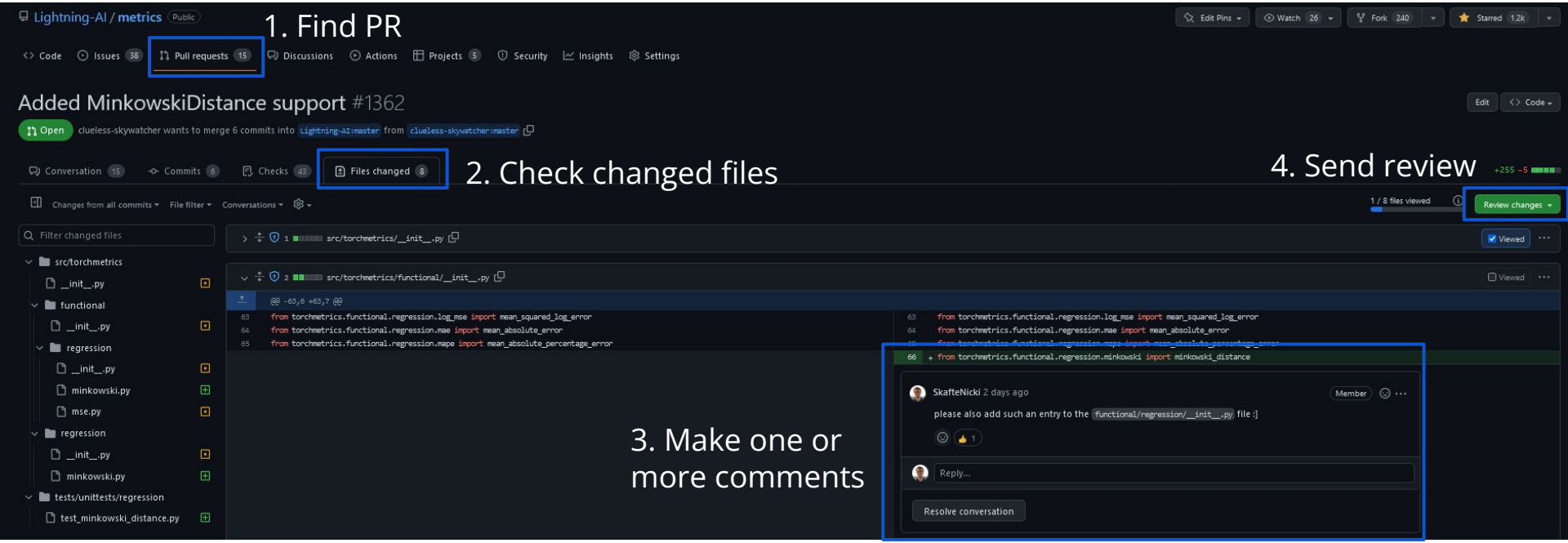
Experimental features/changes are kept away from master.

Recommend extensions for VScode:

- Gitlens or GitGraph
- Github PR and issues



CI step 1: version control



CI step 2: write tests

Test are the cornerstones of CI.

- In particular, *unit tests* are important.
- A single unittest, tests a small part of your code
- By testing code in small pieces, bugs are easier to find

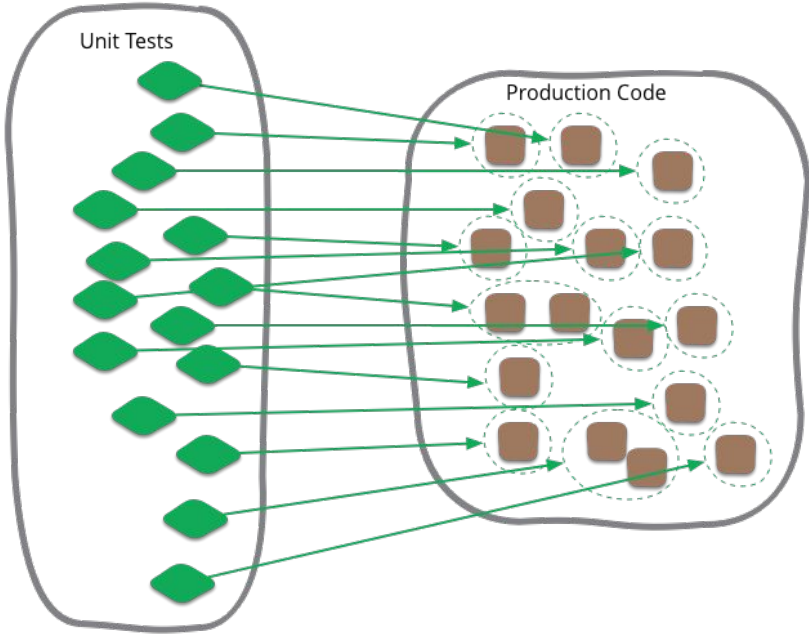
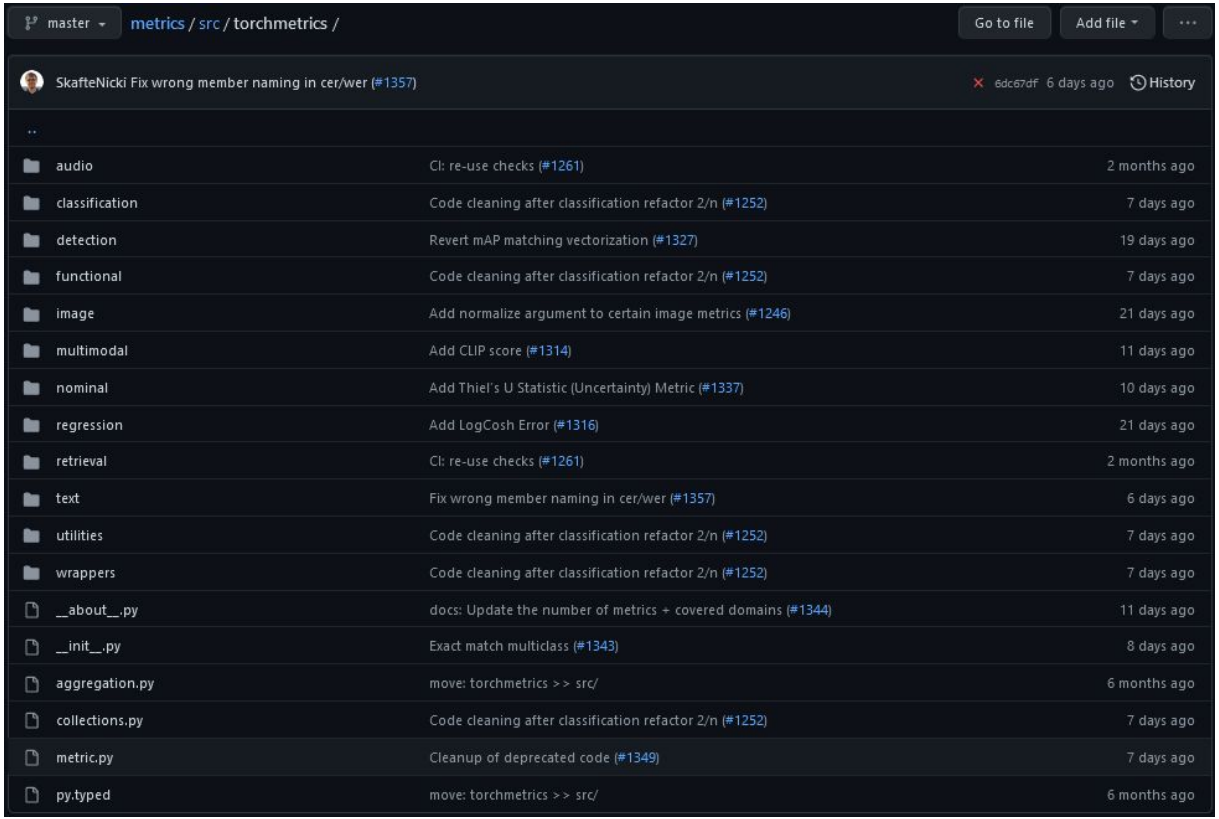


Figure credit: https://martinfowler.com/bliki/UnitTest.html

CI step 2: write tests

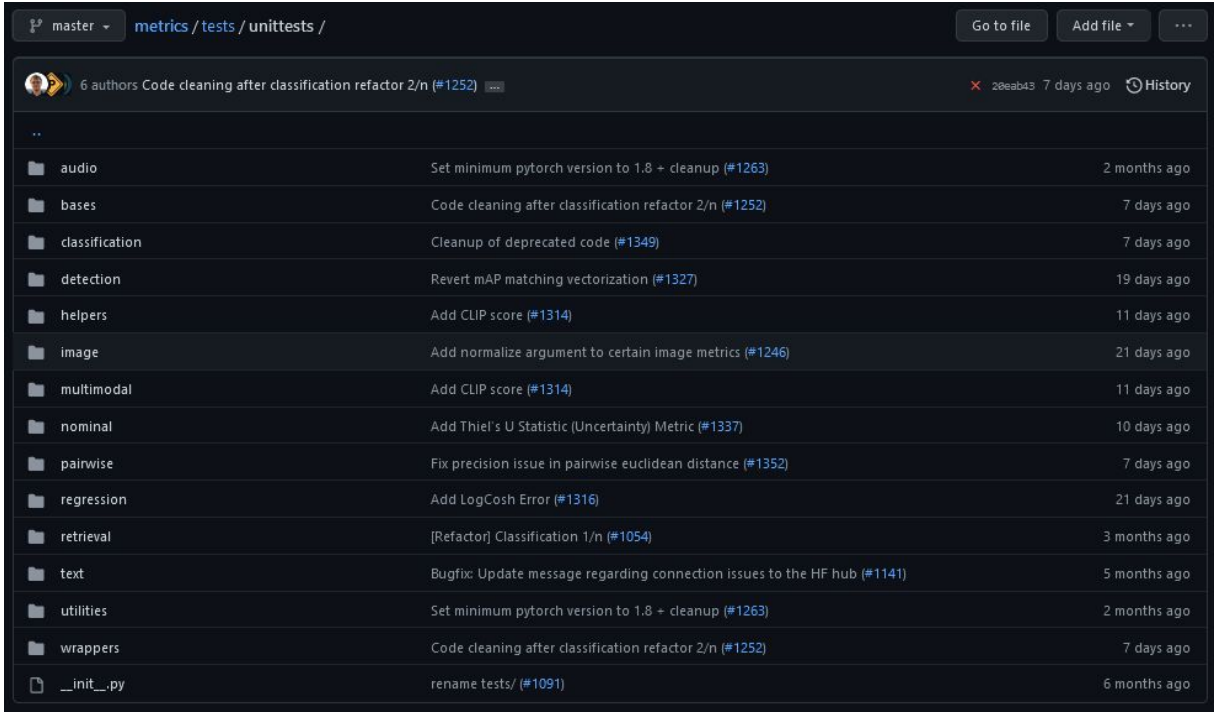
By python convention source code should be in a **src** folder



CI step 2: write tests

(Near) One to one match src folder structure

In total 17727 tests



CI step 2: write tests

In python, we recommend using the **pytest** framework.

Test are simple functions that start with *test_* and uses *assert*

```python
import torch
from torch.nn.functional import mse_loss

def test_mse_loss_zeros():
    # (0 - 0)**2 = 0
    assert mse_loss(torch.zeros(1,), torch.zeros(1,)) == 0

def test_mse_loss_ones():
    # (1 - 0)**2 = 1
    assert mse_loss(torch.ones(1,), torch.zeros(1,)) == 0
```

CI step 2: write tests

Test can be simple...

```python
def test_warning_on_nan(tmpdir):
    preds = torch.randint(3, size=(20, ))
    target = torch.randint(3, size=(20, ))

    with pytest.warns(
        UserWarning,
        match='.* nan values found in confusion matrix have been replaced with zeros.',
    ):
        confusion_matrix(preds, target, num_classes=5, normalize='true')
```

CI step 2: write tests

Test can be simple...

```python
def test_warning_on_nan(tmpdir):
    preds = torch.randint(3, size=(20, ))
    target = torch.randint(3, size=(20, ))

    with pytest.warns(
        UserWarning,
        match='.* nan values found in confusion matrix have been replaced with zeros.',
    ):
        confusion_matrix(preds, target, num_classes=5, normalize='true')
```

Or complicated

```python
@pytest.mark.parametrize("normalize", ['true', 'pred', 'all', None])
@pytest.mark.parametrize(
    "preds, target, sk_metric, num_classes, multilabel",
    [(_input_binary_prob.preds, _input_binary_prob.target, _sk_cm_binary_prob, 2, False),
     (_input_binary_logits.preds, _input_binary_logits.target, _sk_cm_binary_prob, 2, False),
     (_input_binary.preds, _input_binary.target, _sk_cm_binary, 2, False),
     (_input_mlb_prob.preds, _input_mlb_prob.target, _sk_cm_multilabel_prob, NUM_CLASSES, True),
     (_input_mlb_logits.preds, _input_mlb_logits.target, _sk_cm_multilabel_prob, NUM_CLASSES, True),
     (_input_mlb.preds, _input_mlb.target, _sk_cm_multilabel, NUM_CLASSES, True),
     (_input_mcls_prob.preds, _input_mcls_prob.target, _sk_cm_multiclass_prob, NUM_CLASSES, False),
     (_input_mcls_logits.preds, _input_mcls_logits.target, _sk_cm_multiclass_prob, NUM_CLASSES, False),
     (_input_mcls.preds, _input_mcls.target, _sk_cm_multiclass, NUM_CLASSES, False),
     (_input_mdmc_prob.preds, _input_mdmc_prob.target, _sk_cm_multidim_multiclass_prob, NUM_CLASSES, False),
     (_input_mdmc.preds, _input_mdmc.target, _sk_cm_multidim_multiclass, NUM_CLASSES, False)]
)
class TestConfusionMatrix(MetricTester):

    @pytest.mark.parametrize("ddp", [True, False])
    @pytest.mark.parametrize("dist_sync_on_step", [True, False])
    def test_confusion_matrix(
        self, normalize, preds, target, sk_metric, num_classes, multilabel, ddp, dist_sync_on_step
    ):
        self.run_class_metric_test(
            ddp=ddp,
            preds=preds,
            target=target,
            metric_class=ConfusionMatrix,
            sk_metric=partial(sk_metric, normalize=normalize),
            dist_sync_on_step=dist_sync_on_step,
            metric_args={
                "num_classes": num_classes,
                "threshold": THRESHOLD,
                "normalize": normalize,
                "multilabel": multilabel
            }
        )
```

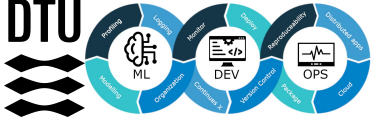
Parametrize is powerful:
4 x 11 x 2 x 2 = 176 tests!

CI step 2: execute locally

```
(lightning) C:\Users\nsde\Documents\metrics>pytest tests\unittests\regression\test_mean_error.py
============================================================================================================================================ test session starts ============================================================================================================================================
platform win32 -- Python 3.8.13, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: C:\Users\nsde\Documents\metrics, configfile: setup.cfg
plugins: cov-4.0.0, doctestplus-0.12.1, timeout-2.1.0
collected 116 items

tests\unittests\regression\test_mean_error.py sssssssssssss.............sssssssssssss.....................................................xxxxxxx.................

============================================================================================================================================ warnings summary ============================================================================================================================================
..\..\Anaconda3\envs\lightning\lib\site-packages\_pytest\config\__init__.py:1183
  C:\Users\nsde\Anaconda3\envs\lightning\lib\site-packages\_pytest\config\__init__.py:1183: PytestDeprecationWarning: The --strict option is deprecated, use --strict-markers instead.
    self.issue_config_time_warning(

-- Docs: https://docs.pytest.org/en/stable/warnings.html
============================================================================================================================ 80 passed, 28 skipped, 8 xfailed, 1 warning in 16.44s =========
```

- `.` Test passed
- `F` Test failed
- `s` Test skipped (`pytest.skipif`, `pytest.skip`)
- `x` Test was expected to fail (`pytest.xfail`)

Do you remember to do this after each commit?
Lets automate doing it instead



CI step 3: Automating stuff

What can be automated: EVERYTHING

- Unit testing
- Integration testing
- Documentation creation
- Linters (style formatting)
- Security checks
- Code coverage
- Custom checks...

Only your imagination is the limit

CI step 3: Setup on your side

To automate testing we need to wrap into a package: create `setup.py`

Allows easy install

`python setup.py install` or `python setup.py devel`

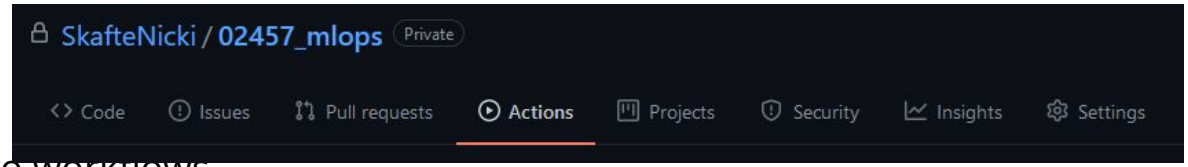
or if uploaded to pip: `pip install my_package`

10 lines (9 sloc) | 362 Bytes — Raw | Blame

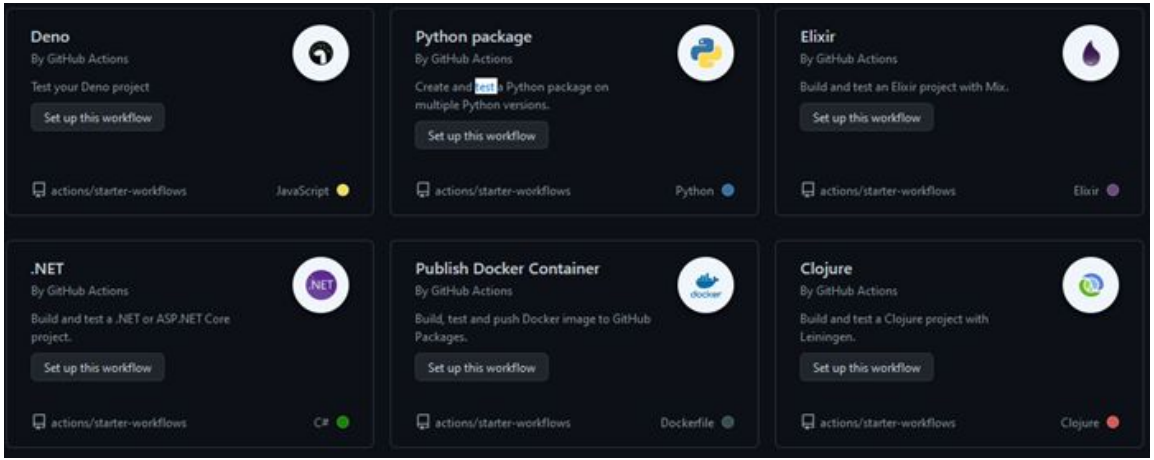
```
from setuptools import find_packages, setup

setup(
    name='src',
    packages=find_packages(),
    version='0.1.0',
    description='{{ cookiecutter.description }}',
    author='{{ cookiecutter.author_name }}',
    license='{% if cookiecutter.open_source_license == 'MIT' %}MIT{% elif cookiecutter.open_source_license == 'BSD-3-Clause' %}BSD-3{% endif %}',
)
```

CI step 3: Github actions

Built-in CI for github. Free 2,000 automation minutes/month (public repository)



Many ready to go workflows



CI step 3: Workflow files

Workflow files are a set of instructions that should be executed on a virtual machine

You can have many workflow files

When tests should be triggered

Define OS + python

Setup python

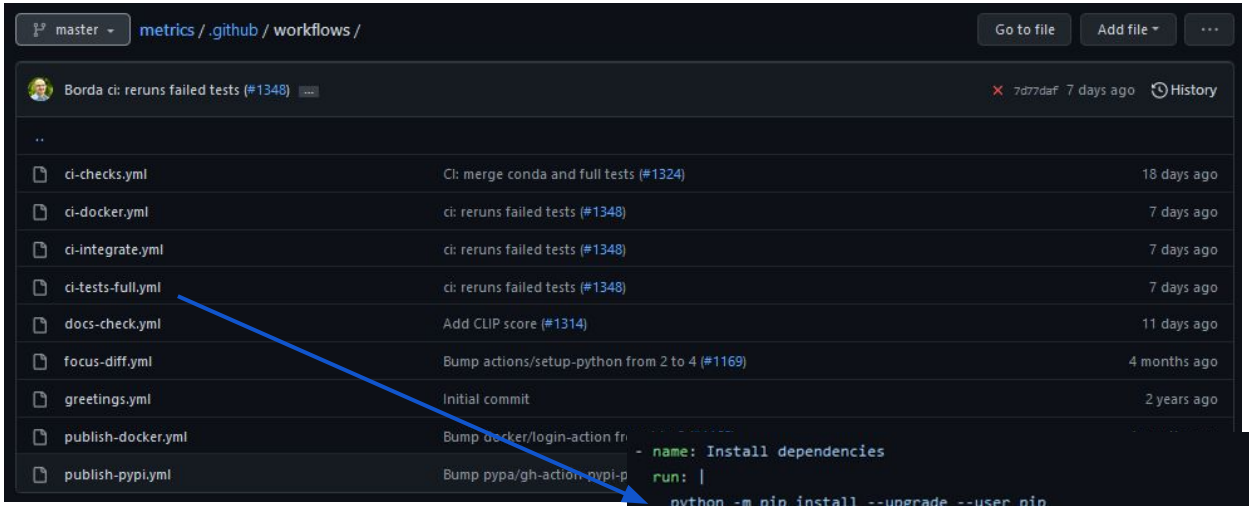
Install dependencies and package

Check formatting

Run tests

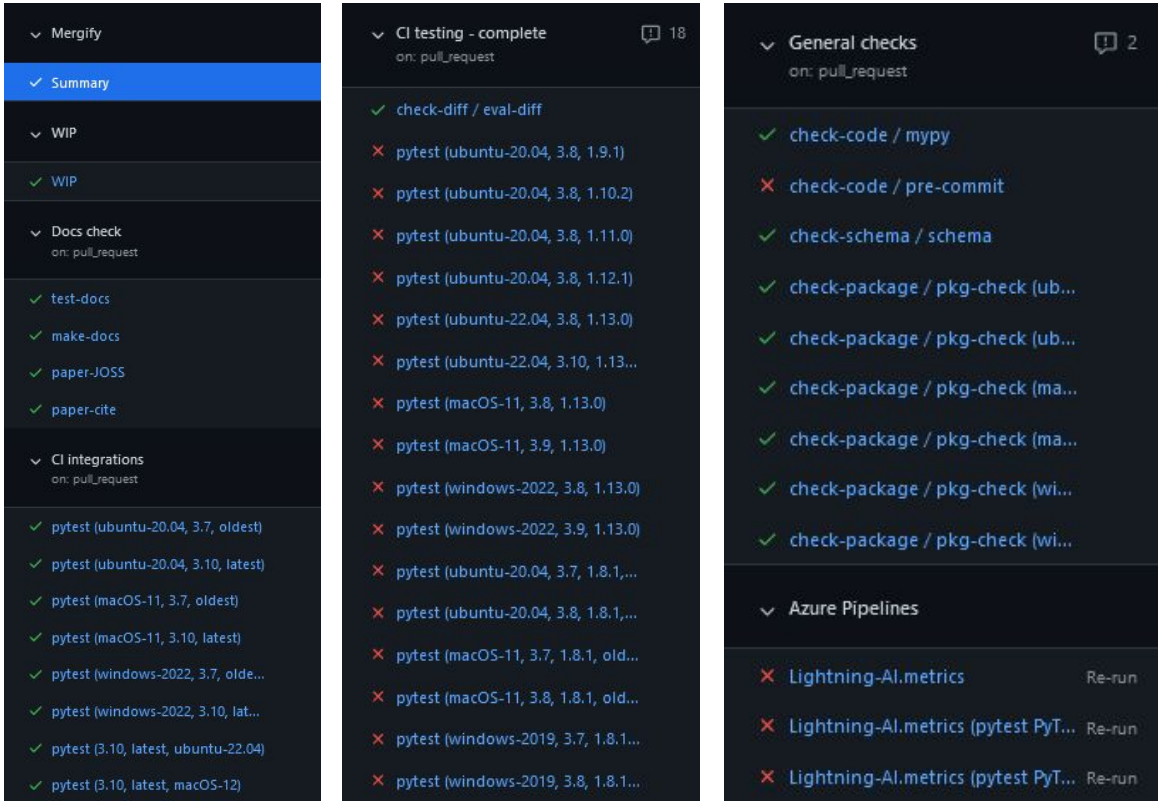
```yaml
name: Python package

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.7", "3.8", "3.9", "3.10"]

    steps:
      - uses: actions/checkout@v3
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install flake8 pytest
          pip install -r requirements.txt
          python setup.py install
      - name: Lint with flake8
        run: |
          flake8 src/
      - name: Test with pytest
        run: |
          pytest tests/
```

CI step 3: Workflow files




```yaml
- name: Install dependencies
  run: |
    python -m pip install --upgrade --user pip
    pip install --requirement ./requirements.txt --find-links https://download.pytorch.org/whl/cpu/torch_stable.html
    pip install "pytest>6.0" "pytest-cov>2.10" --upgrade-strategy only-if-needed
    python --version
    pip --version
    pip list
  shell: bash

- name: Test Package [only]
  run: |
    # NOTE: run coverage on tests does not propagare faler status for Win, https://github.com/nedbat/coveragepy/issues/1003
    python -m pytest torchmetrics -v --cov=torchmetrics --junitxml=junit/test-results-${{ runner.os }}-${{ matrix.python-version }}.xml
```

CI step 3: Workflow files



43 checks in total

Test combination of

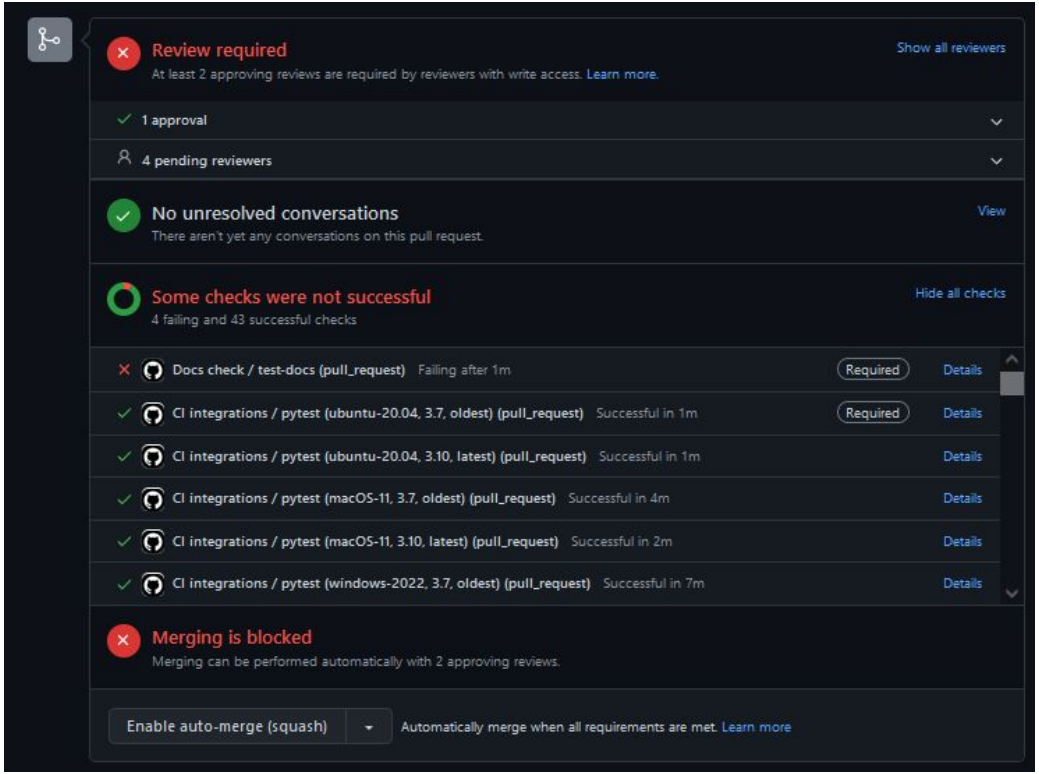
- Hardware setup
- Operating system
- Python version
- Dependencies

Runs unit tests, docs, coverage, linting, package installer etc.

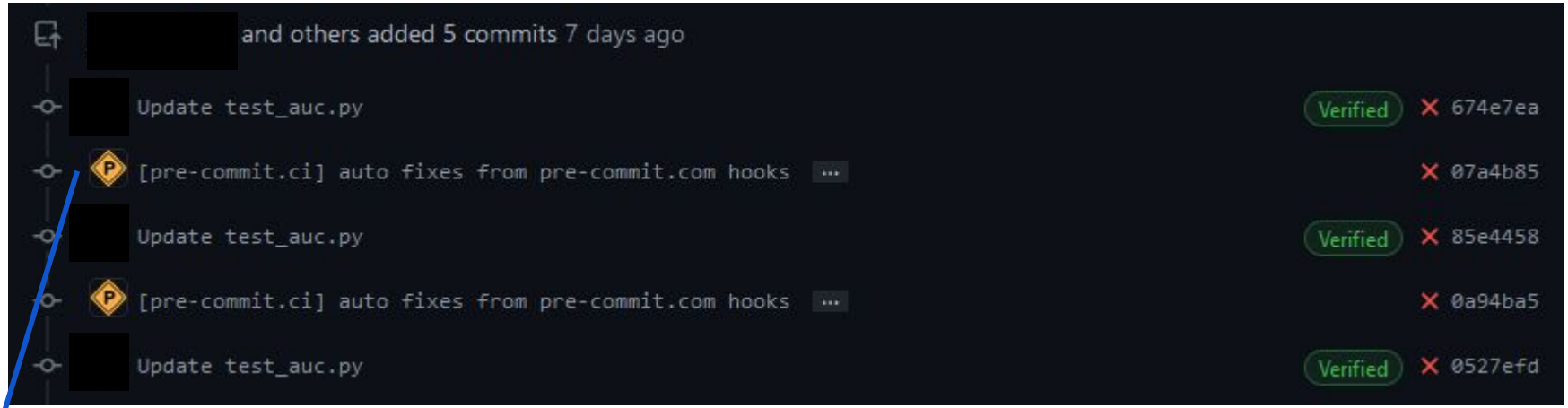
CI step 3: Code is checked before merge

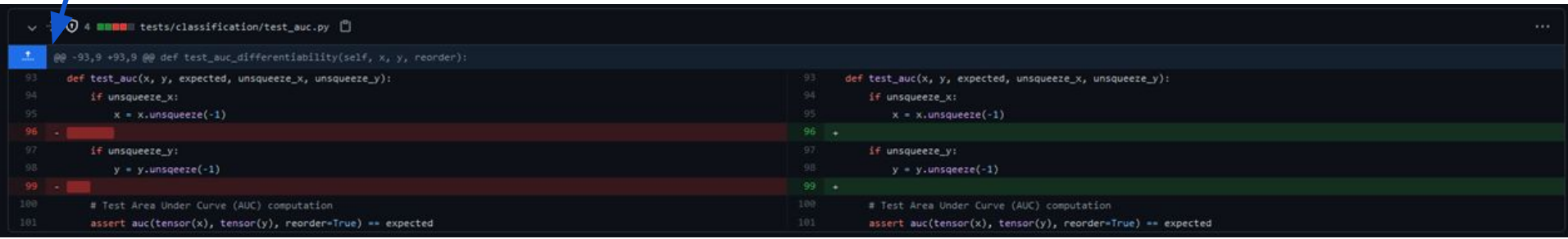
Protection rules:

- All/some tests should pass
- x developers should approve
- Comments should be taken care of



CI step 3: Automate tedious tasks with bots





CI summary

1. Use version control

2. Write (unit-)test for your code

3. Automate build + test

Meme of the day