Continues Integration (CI)

Machine Learning Operations
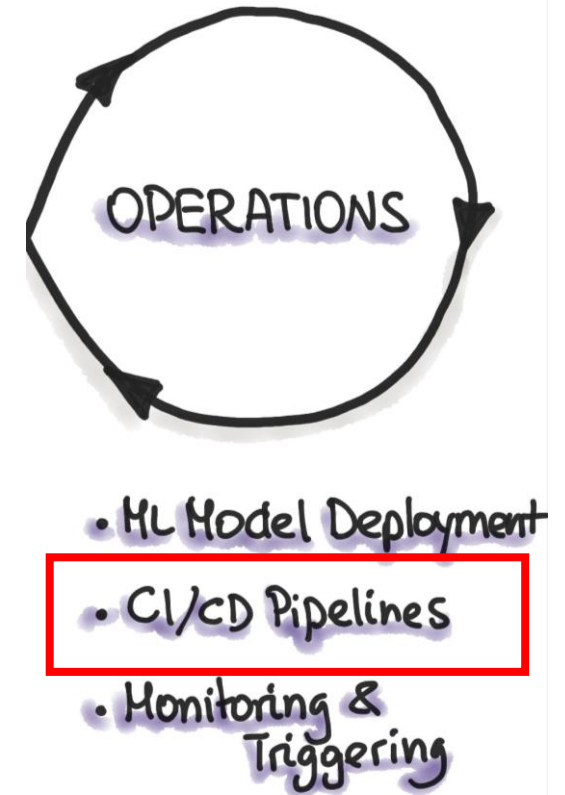
Nicki Skafte Detlefsen,

Postdoc

DTU Compute

What is continues integration?

- Software practise

  - Frequitly commit code to shared reposatory

  - By commit sooner than later, errors are captured early

  - Make merging easier

  - Automate build + test

- App independent

OPERATIONS

- ML Model Deployment
- CI/CD Pipelines
- Monitoring & Triggering

What is continues deployment?

- How to get your code to the user

  - 3 phases: Testing, staging and deploying

- App dependent

Note: Not covered in this course

Nicki Skafte Detlefsen
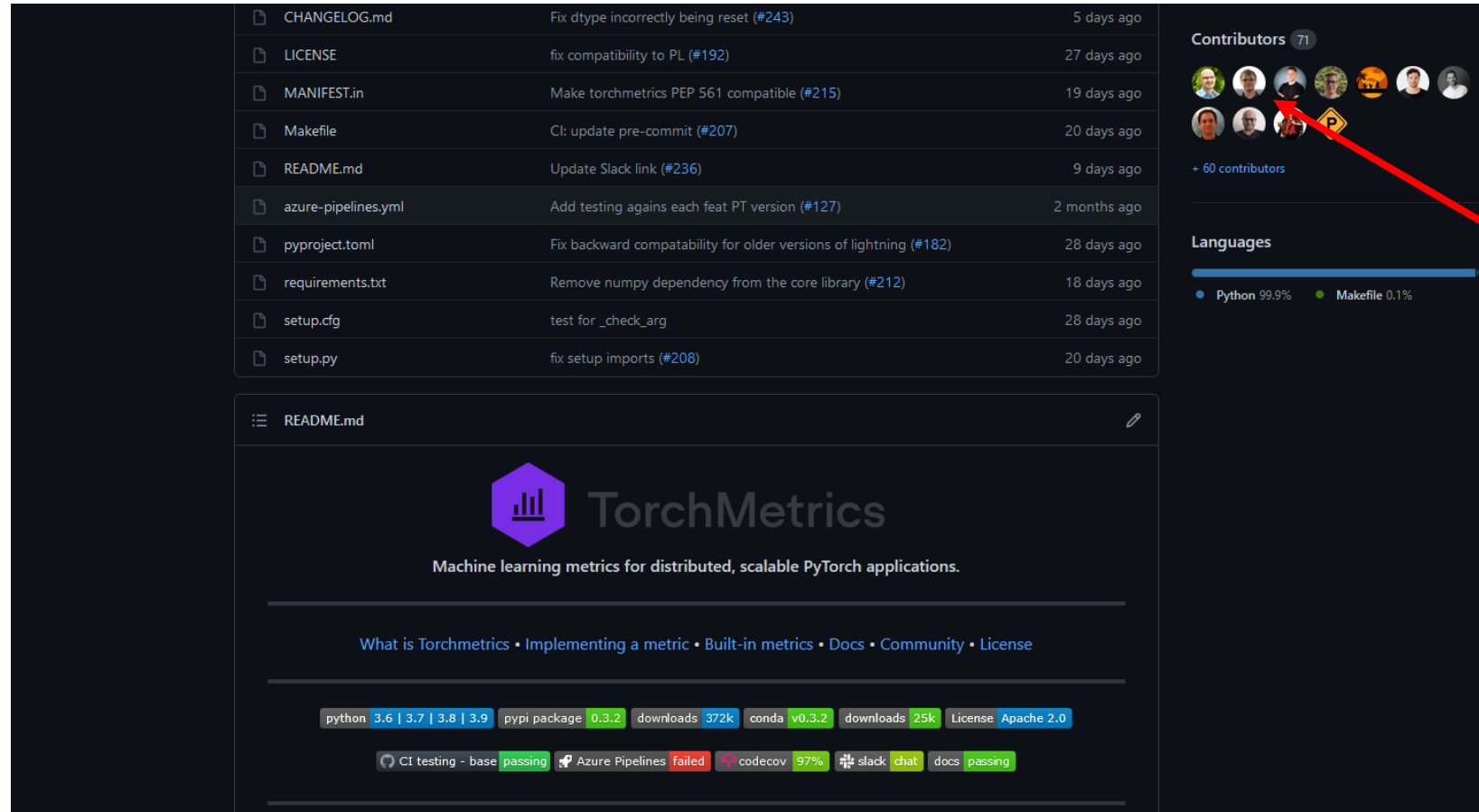
What should you know about CI?

- CI is one of those topics that are best though as learning-by-doing

- If you understand how to use git, the rest is basically googling stuff

Let's look at a practical example

A small case study

All the metric in the world – now in pytorch
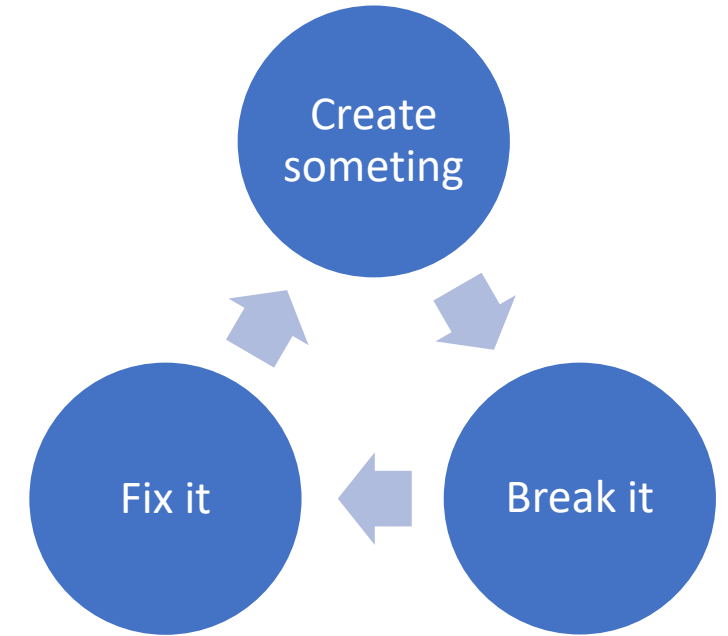

Have I seen that guy before?

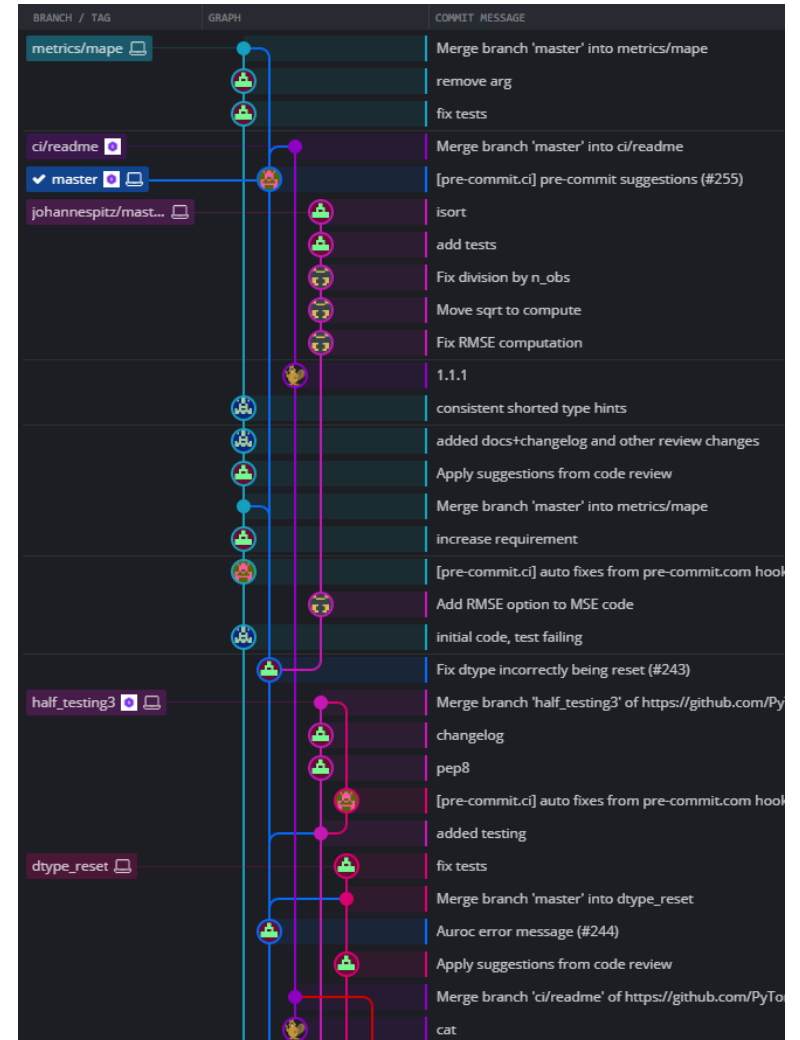# CI step 1: Commiting code

- Commit frequently

  - Catch errors sooner than later

  - Merging can be done automatically

CI step 1: Commiting code

- Use branches

    - Enables parallel workflow

    - Experimental features/changes are keept away from "stable" master

CI step 1: Commiting code

- Use PRs, other can review your code

CI step 2: Automating stuff

- What can be automated: EVERYTHING
  - Functional tests
  - Documentation creation
  - Linters (which check style formatting)
  - Security checks
  - Code coverage
  - Custom checks…

A small casestudy

Ranked by importance (biased)
1. Source code
2. Tests
3. Setup
4. CI workflows
5. Documentation
6. ★

Source code

Test code

In total we have 6365 tests

Test example 1

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')
```

Test example 2

Can be very complex

```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
            }
        )
```

Setup files

- Contains all information regarding the project

- Allow people to do:

  python setup.py install

  or if uploaded to pip

  pip install my_package

```python
setup(
    name='torchmetrics',
    version=about.__version__,
    description=about.__docs__,
    author=about.__author__,
    author_email=about.__author_email__,
    url=about.__homepage__,
    download_url=os.path.join(about.__homepage__, 'archive', 'master.zip'),
    license=about.__license__,
    packages=find_packages(exclude=['tests', 'docs']),
    long_description=long_description,
    long_description_content_type='text/markdown',
    include_package_data=True,
    zip_safe=False,
    keywords=['deep learning', 'machine learning', 'pytorch', 'metrics', 'AI'],
    python_requires='>=3.6',
    setup_requires=[],
    install_requires=setup_tools._load_requirements(_PATH_ROOT),
    project_urls={
        "Bug Tracker": os.path.join(about.__homepage__, 'issues'),
        "Documentation": "https://torchmetrics.rtfd.io/en/latest/",
        "Source Code": about.__homepage__,
    },
    classifiers=[
        'Environment :: Console',
        'Natural Language :: English',
        # How mature is this project? Common values are
        #   3 - Alpha, 4 - Beta, 5 - Production/Stable
        'Development Status :: 3 - Alpha',
        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Scientific/Engineering :: Artificial Intelligence',
        'Topic :: Scientific/Engineering :: Image Recognition',
        'Topic :: Scientific/Engineering :: Information Analysis',
        # Pick your license as you wish
        # 'License :: OSI Approved :: BSD License',
        'Operating System :: OS Independent',
        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.6',
        'Programming Language :: Python :: 3.7',
        'Programming Language :: Python :: 3.8',
        'Programming Language :: Python :: 3.9',
    ],
)
```
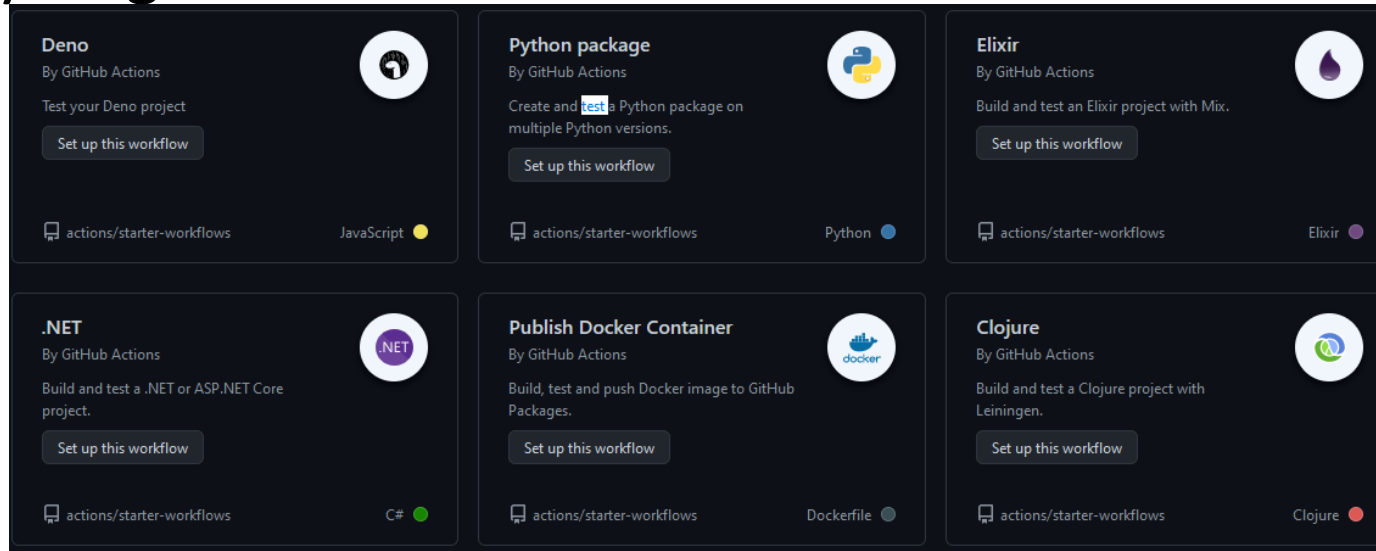
Github actions

- Build-in CI for github

- Free 2,000 automation minutes/month (public repository)
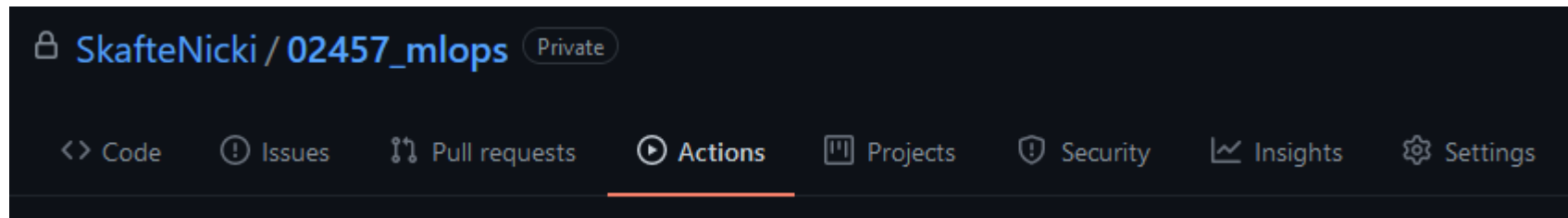
- Many ready to go workflows

The anatomy of a workflow file

MLOps

02457_mlops / .github / workflows / python-package.yml in main

<> Edit new file   👁 Preview

```
1    name: Python package
2    on:
3      push:
4        branches: [ main ]
5      pull_request:
6        branches: [ main ]
7    jobs:
8      build:
9        runs-on: ubuntu-latest
10       strategy:
11         fail-fast: false
12         matrix:
13           python-version: [3.7, 3.8, 3.9]
14       steps:
15       - uses: actions/checkout@v2
16       - name: Set up Python ${{ matrix.python-version }}
17         uses: actions/setup-python@v2
18         with:
19           python-version: ${{ matrix.python-version }}
20       - name: Install dependencies
21         run: |
22           python -m pip install --upgrade pip
23           python -m pip install flake8 pytest
24           if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
25       - name: Lint with flake8
26         run: |
27           # stop the build if there are Python syntax errors or undefined names
28           flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
29           # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
30           flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics
31       - name: Test with pytest
32         run: |
33           pytest
```

When tests should be triggered

Define Operating system + python version

Setup python

Install dependencies

Check linting (stop if errors)

Run tests

A small case study

All of our workflows – 36 in total

CI testing - complete
on: pull_request

- ✓ pytest (ubuntu-20.04, 3.6, minimal)
- ✓ pytest (ubuntu-20.04, 3.8, minimal)
- ✓ pytest (ubuntu-20.04, 3.8, latest)
- ✓ pytest (ubuntu-20.04, 3.9, latest)
- ✓ pytest (macOS-10.15, 3.6, minimal)
- ✓ pytest (macOS-10.15, 3.8, minimal)
- ✓ pytest (macOS-10.15, 3.8, latest)
- ✓ pytest (macOS-10.15, 3.9, latest)
- ✓ pytest (windows-2019, 3.6, mini...
- ✓ pytest (windows-2019, 3.8, mini...
- ✓ pytest (windows-2019, 3.8, latest)
- ✓ pytest (windows-2019, 3.9, latest)

WIP

- ✓ WIP

Check Code formatting
on: pull_request

- ✓ flake8
- ✓ imports-check-isort
- ✓ typing-check-mypy

Docs check
on: pull_request

- ✓ test-docs
- ✓ make-docs

CI testing - base
on: pull_request

- ✓ doctest (ubuntu-20.04, 3.7)
- ✓ doctest (windows-2019, 3.7)
- ✓ doctest (macOS-10.15, 3.7)

Azure Pipelines

- ✗ PyTorchLightning.metrics   Re-run

Mergify

- ✓ Summary

Codecov

- ✓ codecov/patch
- ✓ codecov/project

PyTorch & Conda   💬 2
on: pull_request

- ✓ conda (3.7, 1.3)
- ✓ conda (3.7, 1.4)
- ✓ conda (3.7, 1.5)
- ✓ conda (3.7, 1.6)
- ✓ conda (3.7, 1.7)
- ⚠ conda (3.7, 1.8)
- ✓ conda (3.7, 1.9)

Not github actions

Install pkg
on: pull_request

- ✓ pkg-check
- ✓ pkg-install (ubuntu-20.04, 3.7)
- ✓ pkg-install (macOS-10.15, 3.7)
- ✓ pkg-install (windows-2019, 3.7)
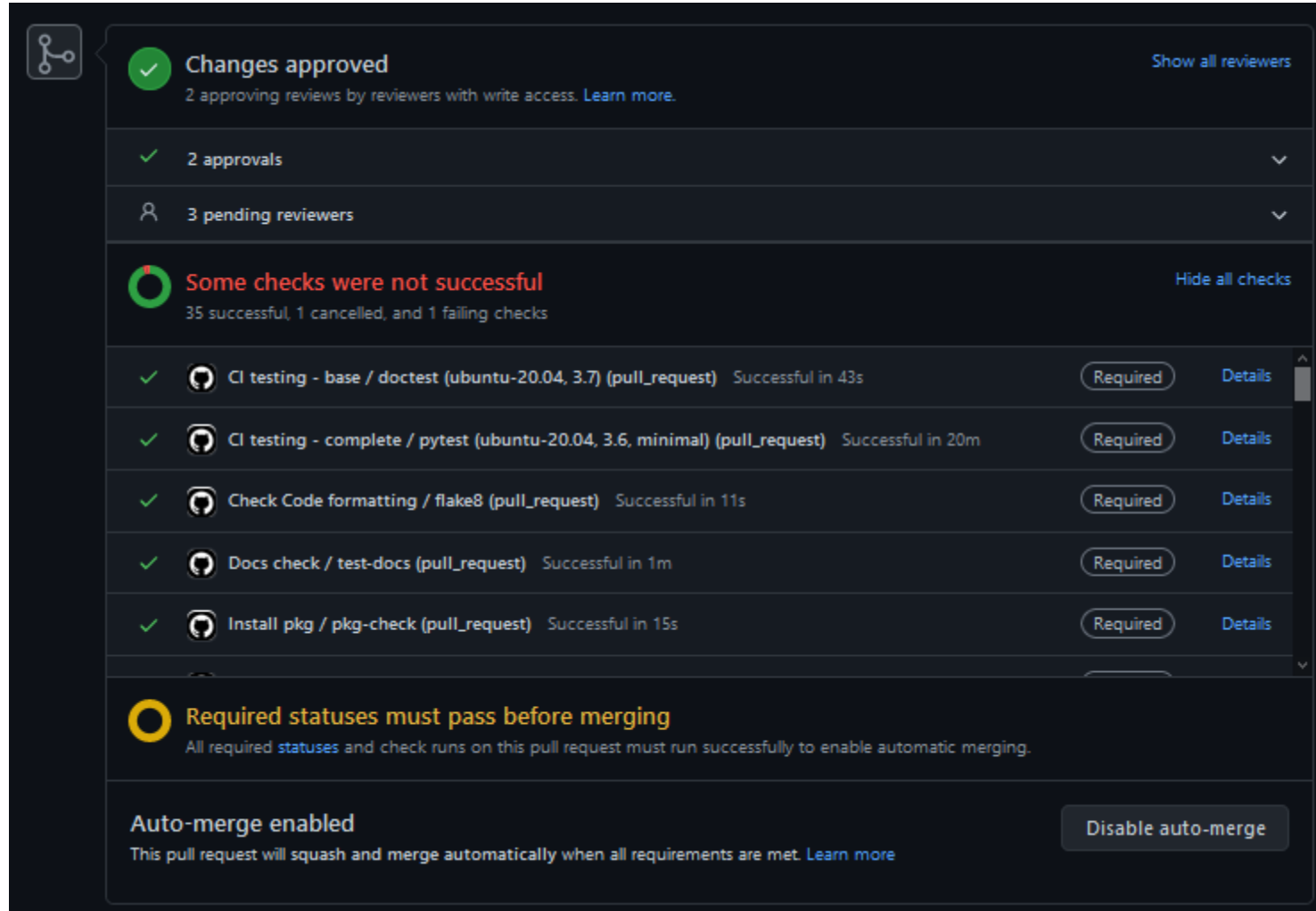
Test combination of
- Hardware setup
- Operating system
- Python version
- Dependencies

Runs tests, docs, coverage, lintint, package install ect

Each PR triggers all tests

Nicki Skafte Detlefsen
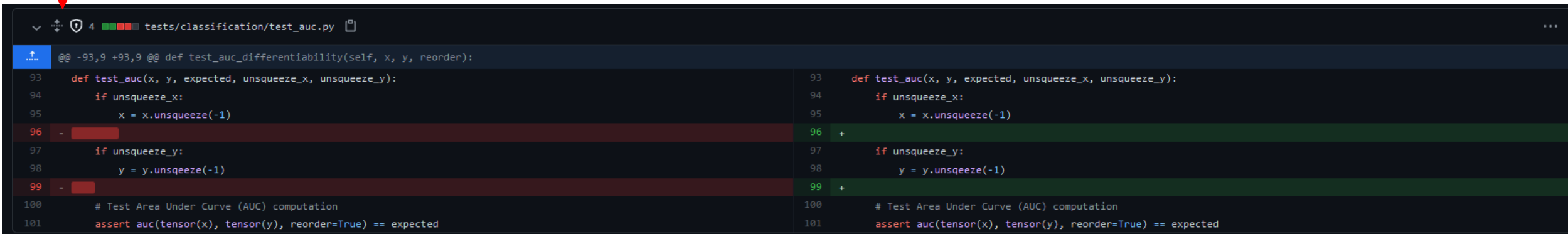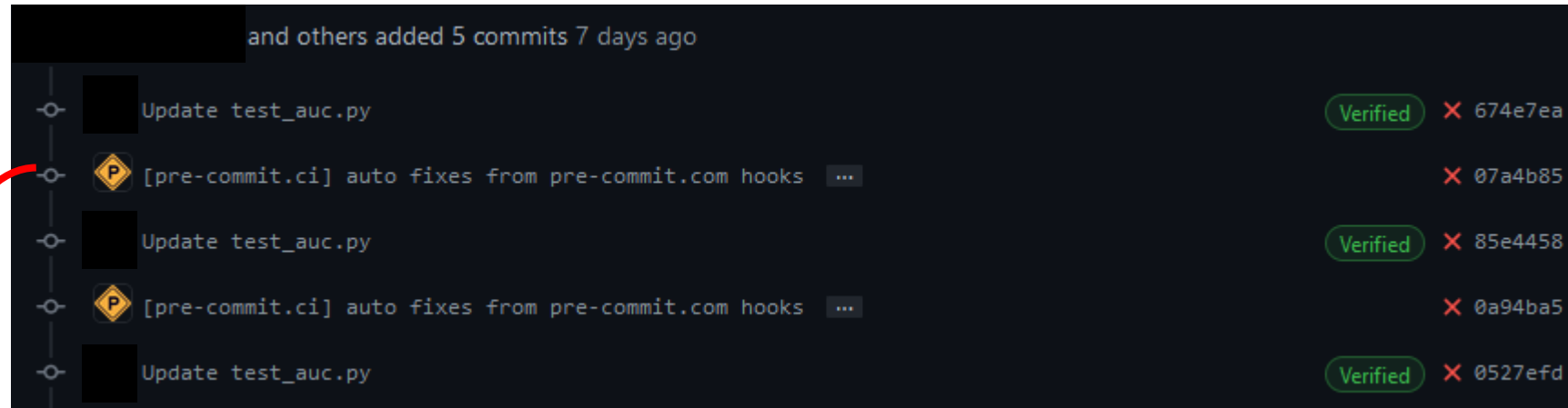
Other files to consider

- requirements.txt
- LICENSE
- README.md
- .gitignore
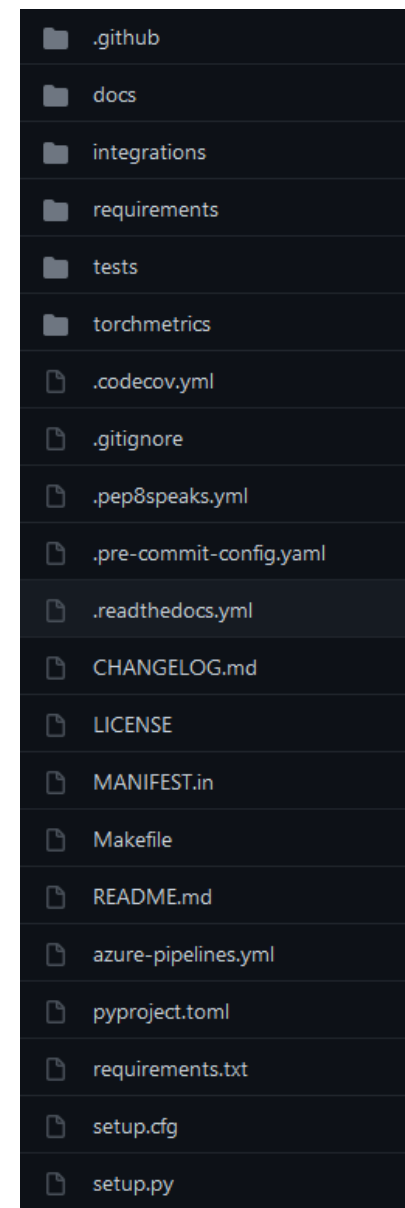
Advanced: Use bots

Bots can take care of tidious task for you (like linting)

The future is here

Nicki Skafte Detlefsen