02476 Machine Learning Operations

Nicki Skafte Detlefsen

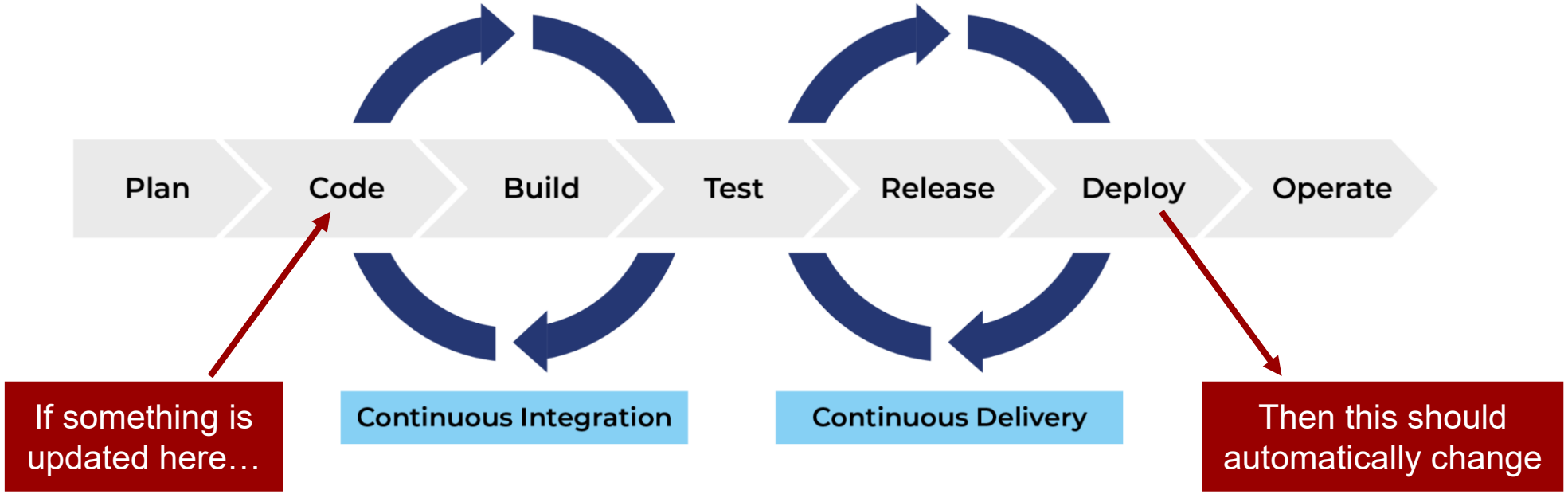
Continuous Integration

Why you should care about today

3 years ago, the day before this lecture, the internet went down for a couple of hours because someone f..ked up their continues integration at [Fastly]

Continues X

Term refers to a set of software practices for automating tedious tasks and make sure changed in a pipeline are continuously propagated through the pipeline



CI

Continues Integration

Core tasks:

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

💻App independent concept

CD

Continues Deployment

Core tasks:

💡 How to get your code/application to the user automatically? + monitor life cycle

💻App dependent concept

CML

Continues Machine Learning

Core tasks:

💡 How to automatically retrain machine learning models when data and code changes?

💻Specific to ML applications

MLOps levels

The **Maturity model** overall describes the DevOps practices to run a successful MLOps environment.

Intended to identify gaps in an existing organization's attempt to implement such an environment.

- Estimate the scope of the work for new engagements.
- Establish realistic success criteria.
- Identify deliverables you'll hand over at the conclusion of the engagement.

Level	Description	Highlights	Technology
0	No MLOps	• Difficult to manage full machine learning model lifecycle • The teams are disparate and releases are painful • Most systems exist as "black boxes," little feedback during/post deployment	• Manual builds and deployments • Manual testing of model and application • No centralized tracking of model performance • Training of model is manual
1	DevOps but no MLOps	• Releases are less painful than No MLOps, but rely on Data Team for every new model • Still limited feedback on how well a model performs in production • Difficult to trace/reproduce results	• Automated builds • Automated tests for application code
2	Automated Training	• Training environment is fully managed and traceable • Easy to reproduce model • Releases are manual, but low friction	• Automated model training • Centralized tracking of model training performance • Model management
3	Automated Model Deployment	• Releases are low friction and automatic • Full traceability from deployment back to original data • Entire environment managed: train > test > production	• Integrated A/B testing of model performance for deployment • Automated tests for all code • Centralized tracking of model training performance
4	Full MLOps Automated Operations	• Full system automated and easily monitored • Production systems are providing information on how to improve and, in some cases, automatically improve with new models • Approaching a zero-downtime system	• Automated model training and testing • Verbose, centralized metrics from deployed model

CI

CD

CML

This lecture: continues integration

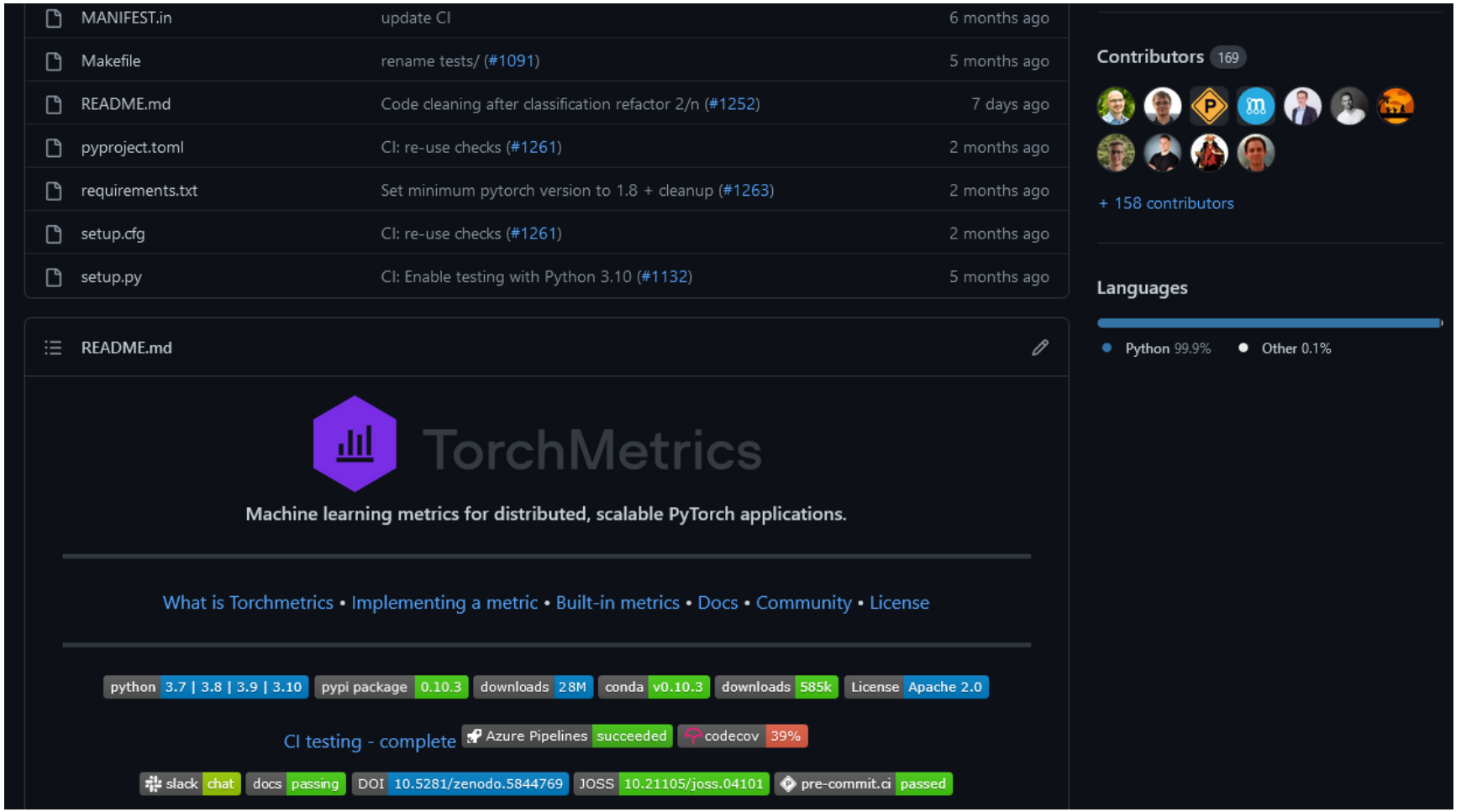
Core task:

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

3 steps to do this:

- Use version control
  - Frequently committing code to a shared repository
- Write (unit)test for your code
  - Should capture unwanted bugs in your code
- Automate build + testing
  - Automatically run test so code cannot be merged without working

A small case study for continuous integration



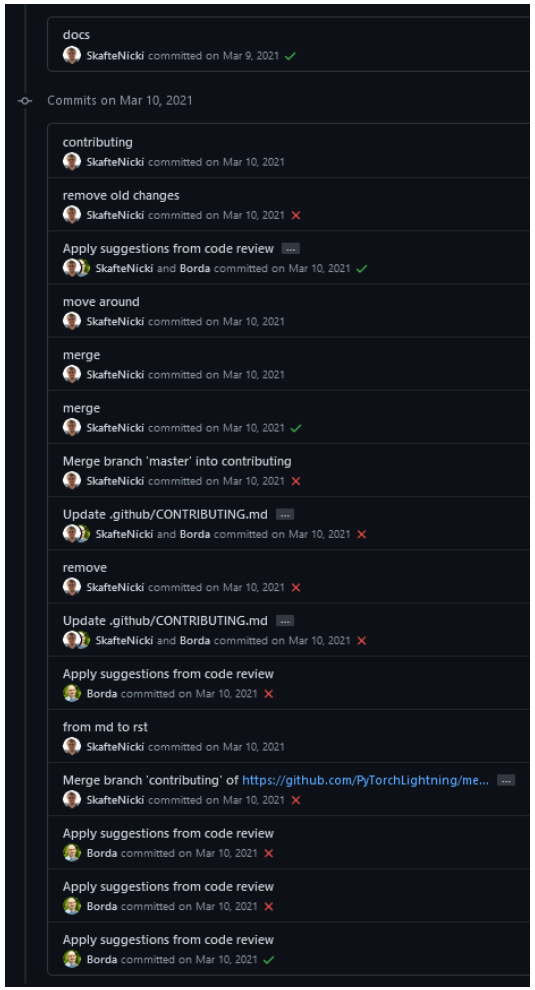
CI step 1: version control

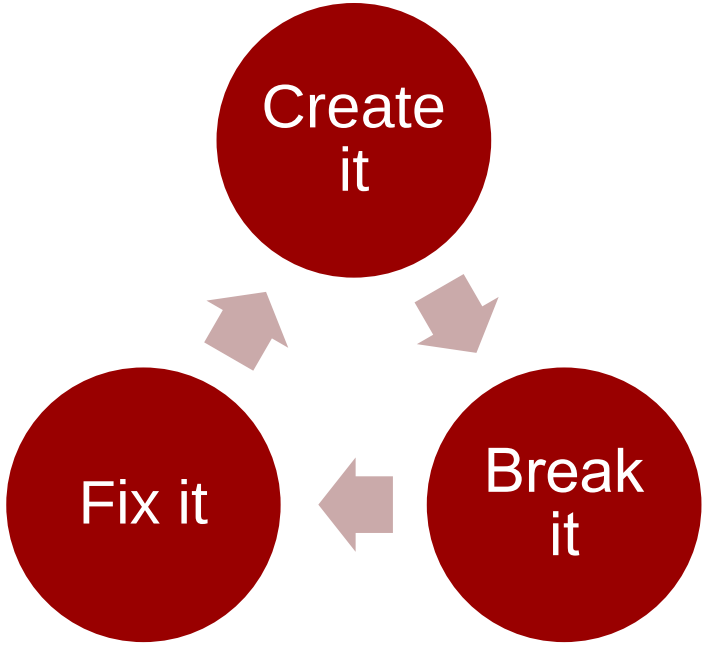
User version control:

- Code changes are tracked
- Branches for parallel work

Commit frequently:

- Catch errors sooner than later
- Revert back easily to when things were working
- Merging can be done automatically





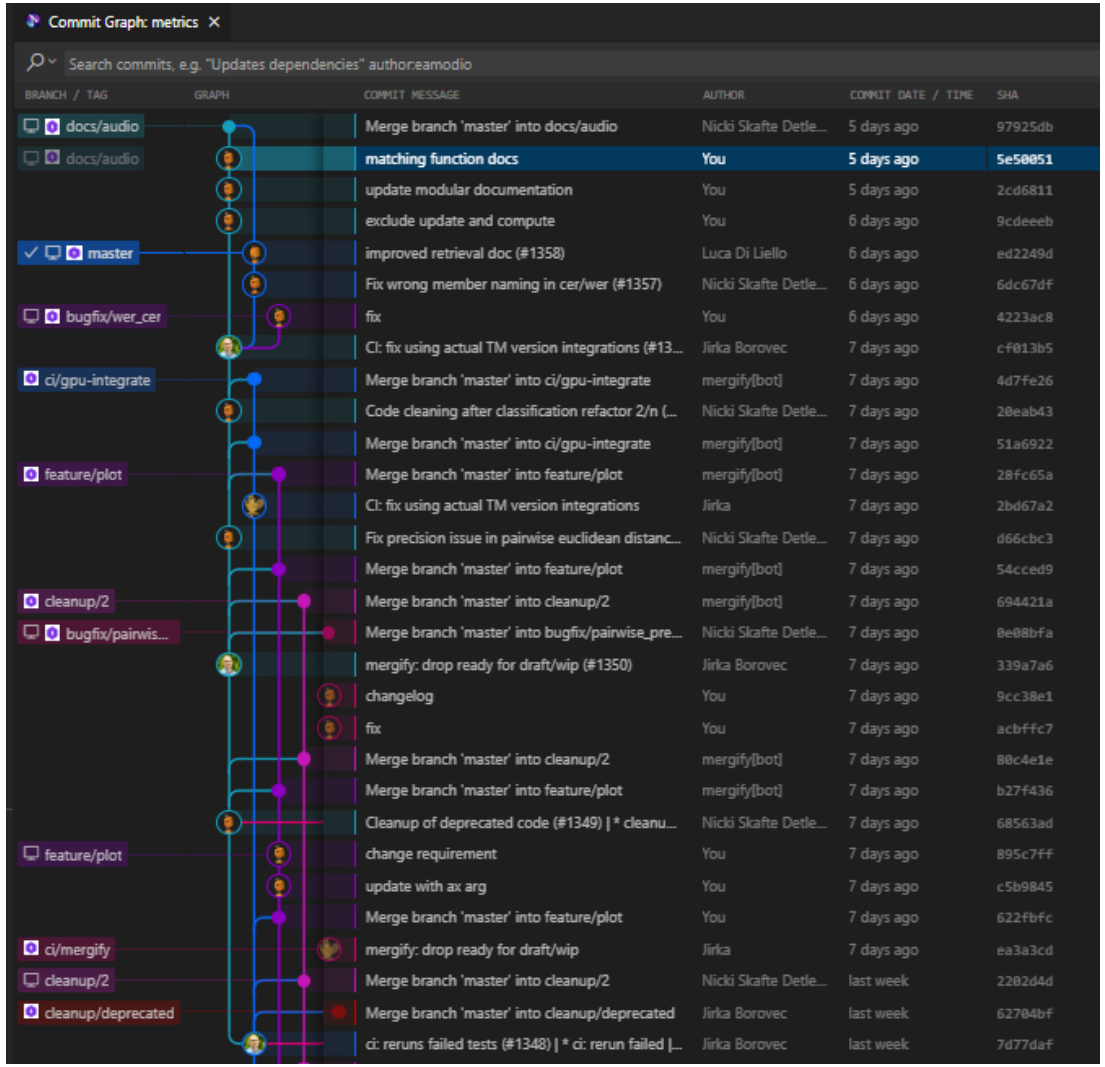
CI step 1: Use branches

Parallel workflow

Experimental features changes are kept away from master/main

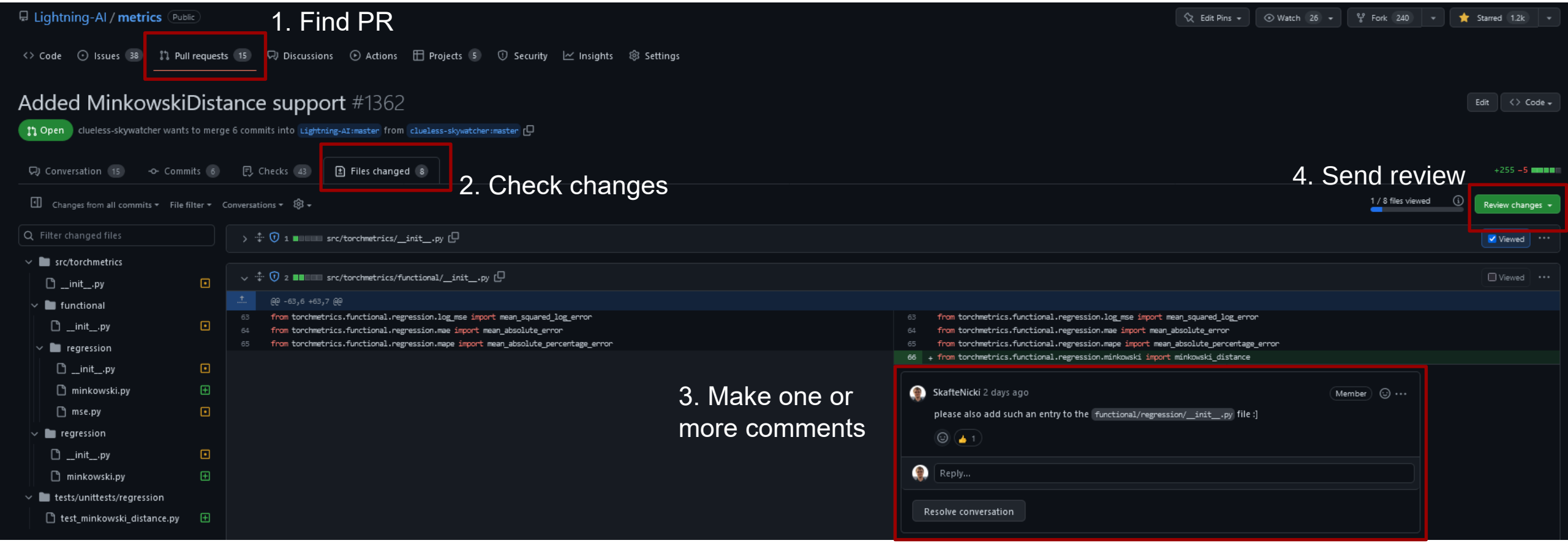
Recommend extensions for VS code:

- Gitlens or GitGraph
- Github PR and issues



CI step 1: Use pull requests

⚠ No commit can be pushed to master without being in a pull request



CI step 1: pre-commit

☑ Check that everything is up to standard before commits are created

.pre-commit-config.yaml

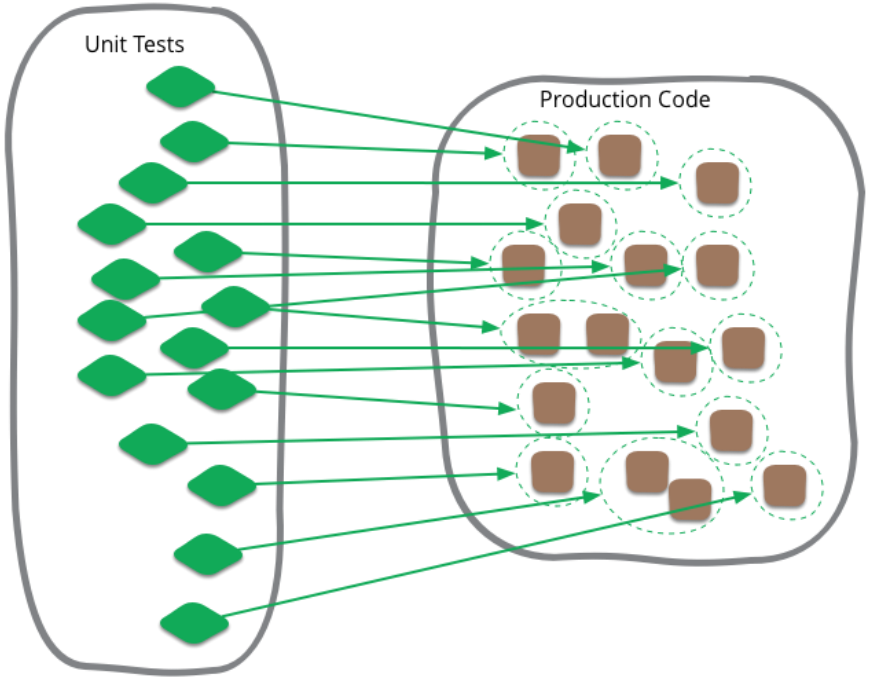
```yaml
default_language_version:
  python: python3

repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v4.4.0
    hooks:
      - id: end-of-file-fixer
      - id: trailing-whitespace
      # - id: check-json
      # - id: check-yaml
      - id: check-toml
      - id: check-docstring-first
      - id: check-executables-have-shebangs
      - id: check-case-conflict
      - id: detect-private-key

  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.1.3
    hooks:
      - id: ruff
        args: [--fix, --exit-non-zero-on-fix]

  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.1.3
    hooks:
      - id: ruff-format

  - repo: https://github.com/codespell-project/codespell
    rev: v2.2.5
    hooks:
      - id: codespell
        additional_dependencies: [tomli]
```

```
dtu_mlops on  main [!?] via  v3.11.5 (mlops)
❯ git commit -m "implementation of client"
fix end of files.....................................................Failed
- hook id: end-of-file-fixer
- exit code: 1
- files were modified by this hook

Fixing s8_monitoring/exercise_files/client.py

trim trailing whitespace.....................................................Passed
check toml.....................................(no files to check)Skipped
check docstring is first.....................................................Passed
check that executables have shebangs.....................................Passed
check for case conflicts.....................................................Passed
detect private key.....................................................Passed
ruff.....................................................Failed
- hook id: ruff
- exit code: 1
- files were modified by this hook

s8_monitoring\exercise_files\client.py:17:12: S113 Probable use of requests call without timeout
Found 2 errors (1 fixed, 1 remaining).

ruff-format.....................................................Passed
codespell.....................................................Passed
markdownlint-docker.....................................(no files to check)Skipped
```

CI step 2: write tests

Tests are the cornerstones of continuous integration

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



Other test types worth considering:

- Integration tests
- Regression tests
- Performance tests
- Security tests

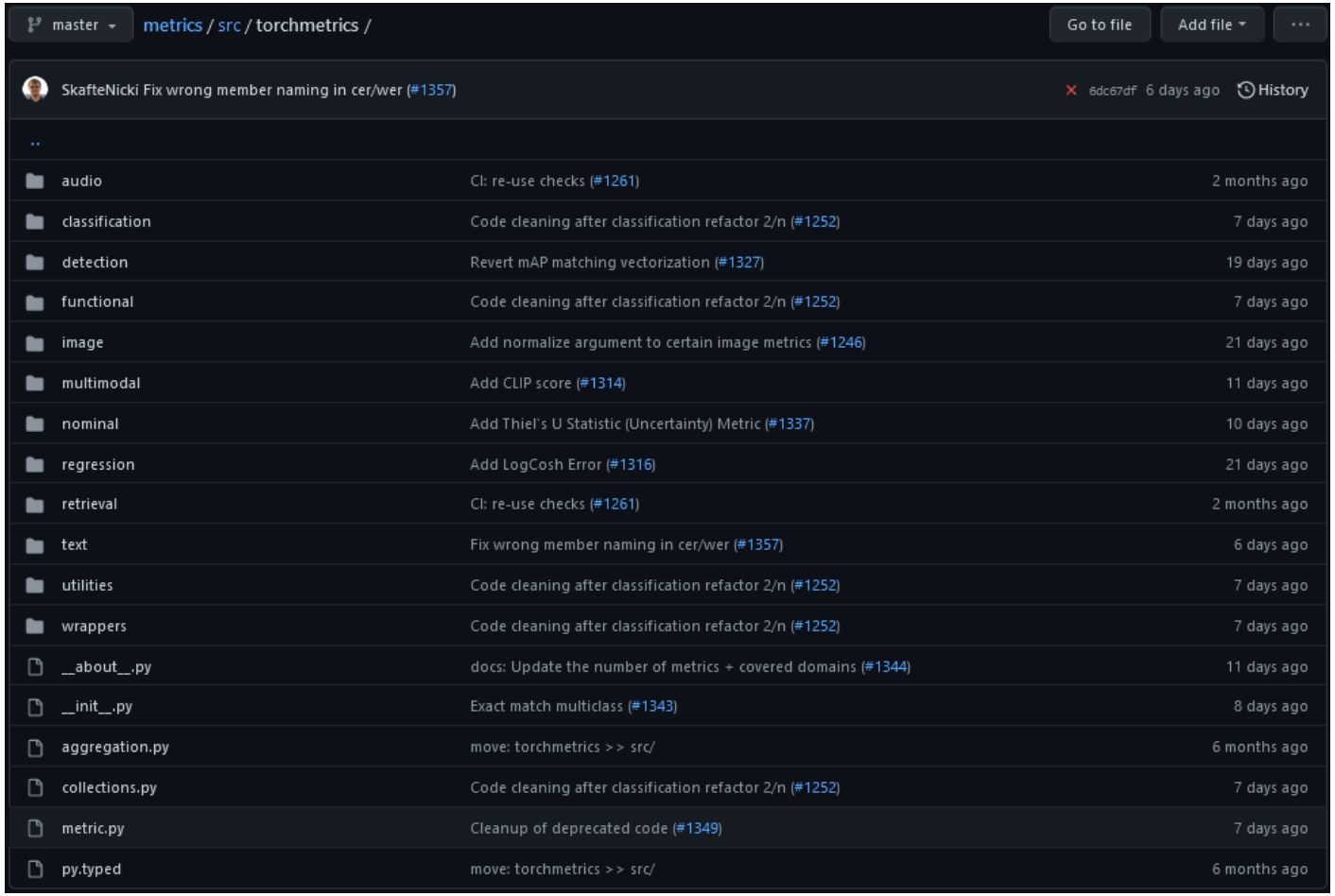
CI step 2: write tests

By Python convention your source code should either be

src/<project_name>
(src-layout)

or

<project_name>
(flat-layout)



CI step 2: write tests

For tests, the convention is to either

```
├── README.md
├── src/
│   ├── __init__.py
│   └── important_functions.py
├── tests/
│   │
│   ├── index.md
```

CI step 2: write tests

CI step 2: write tests

CI step 2: execute locally

CI step 3: Automating stuff

What can be automated: EVERYTHING 🤖

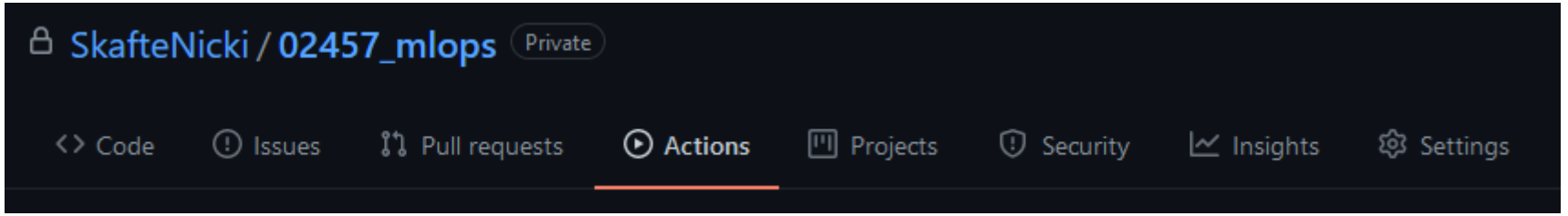
- 💡 Unit testing
- 💡 Integration testing
- 💡 Documentation creation
- 💡 Linters (style formatting)
- 💡 Security checks
- 💡 Code coverage
- 💡 Custom checks…

Only your imagination is the limit…

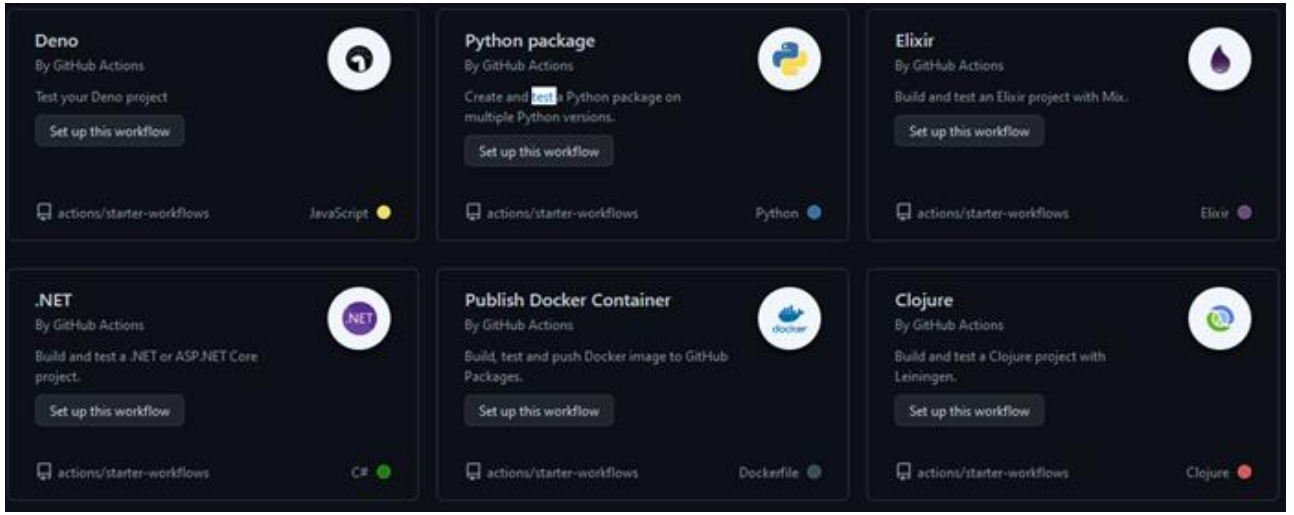
CI step 3: Github actions

Build-in continuous integration in Github.

Free 2000 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 hosted by Github

You can have one or many workflow files (runs in parallel)

When should workflow be triggered

Define OS + python

Clones code

Setup Python

Install dependencies

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
```

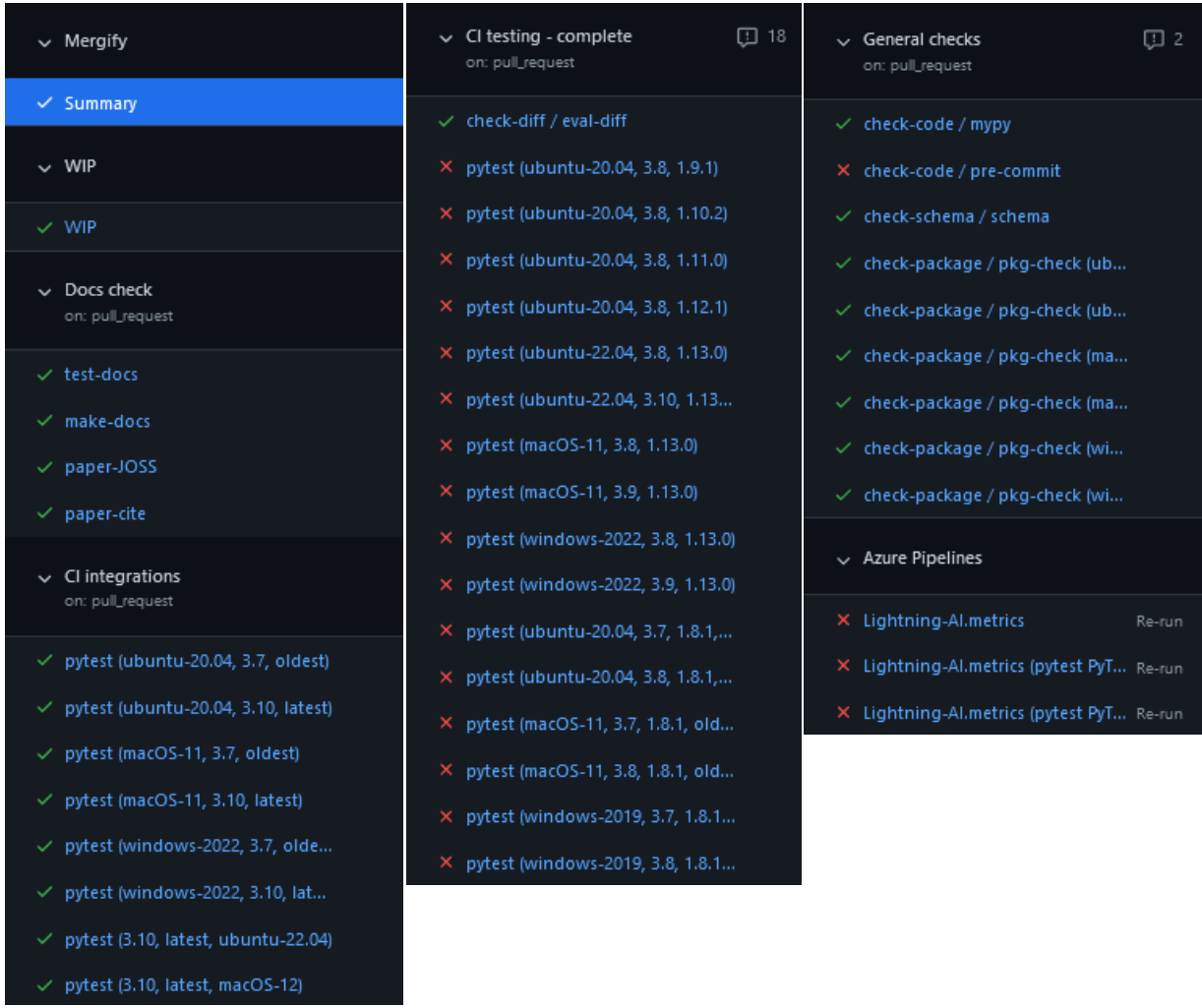
CI step 3: Workflow files

☑ 43 checks in total

Test a combination of

- Hardware setup
- Operating system
- Python version
- Core dependencies

Runs unit tests, build documentation, test coverage, linting of code, package installer etc.



CI step 3: Code is checked before merging

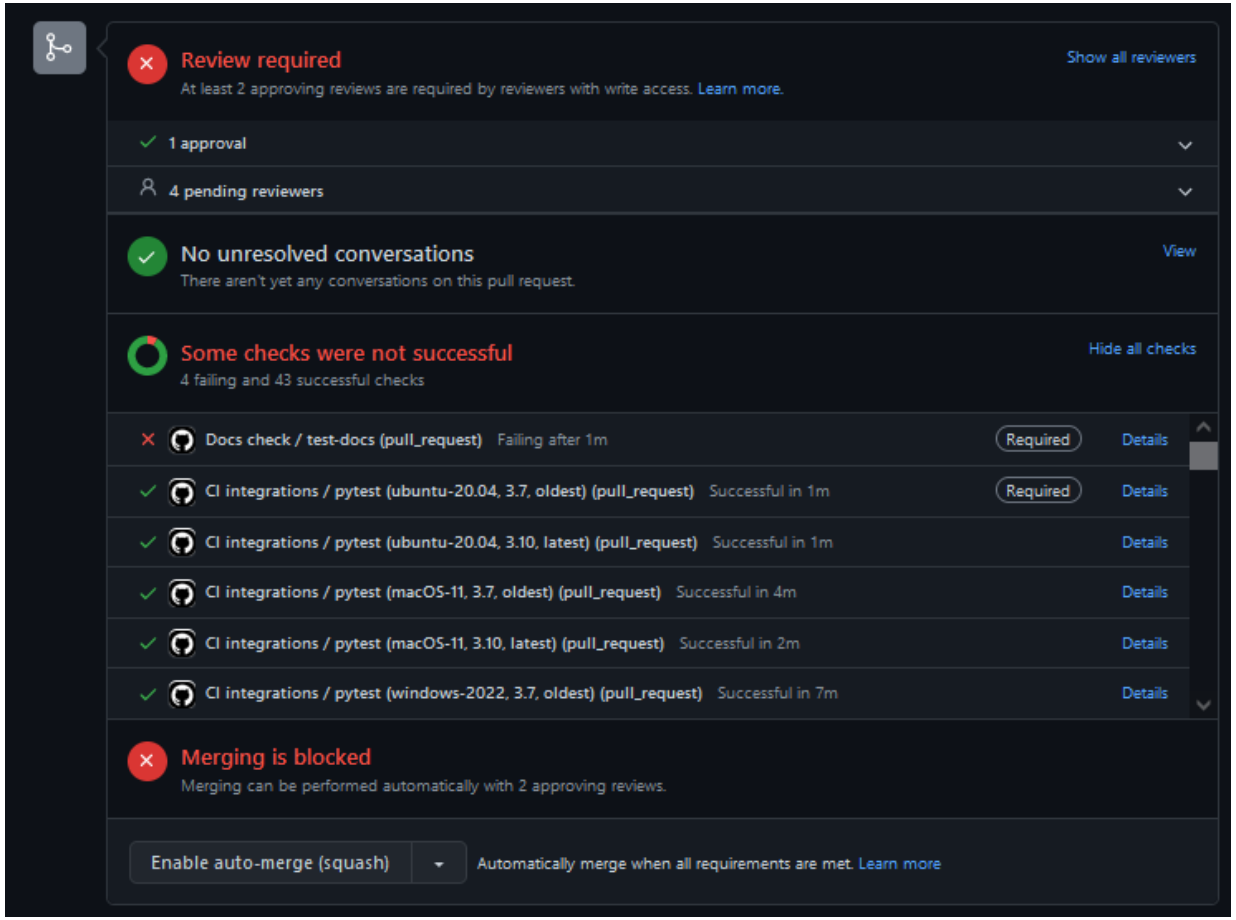
Branch protection rules:

⚠ All/some tests should pass

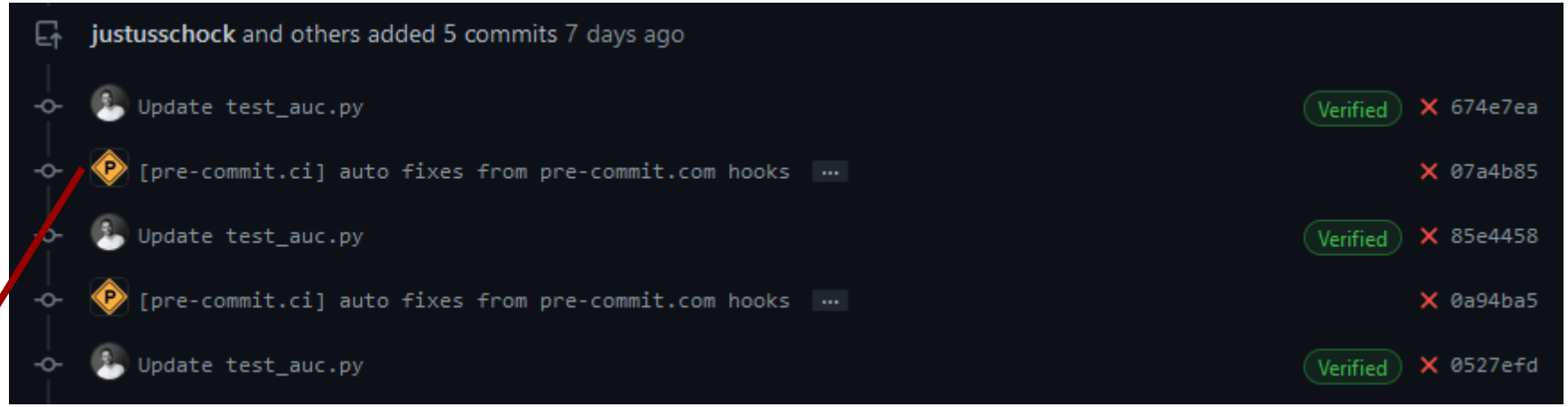
⚠ At least x core members need to approve

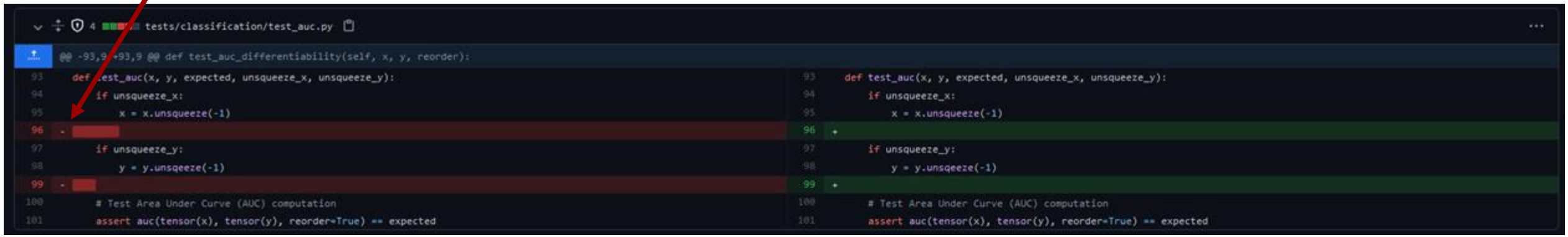
⚠ Comments should be taken care of

View more here



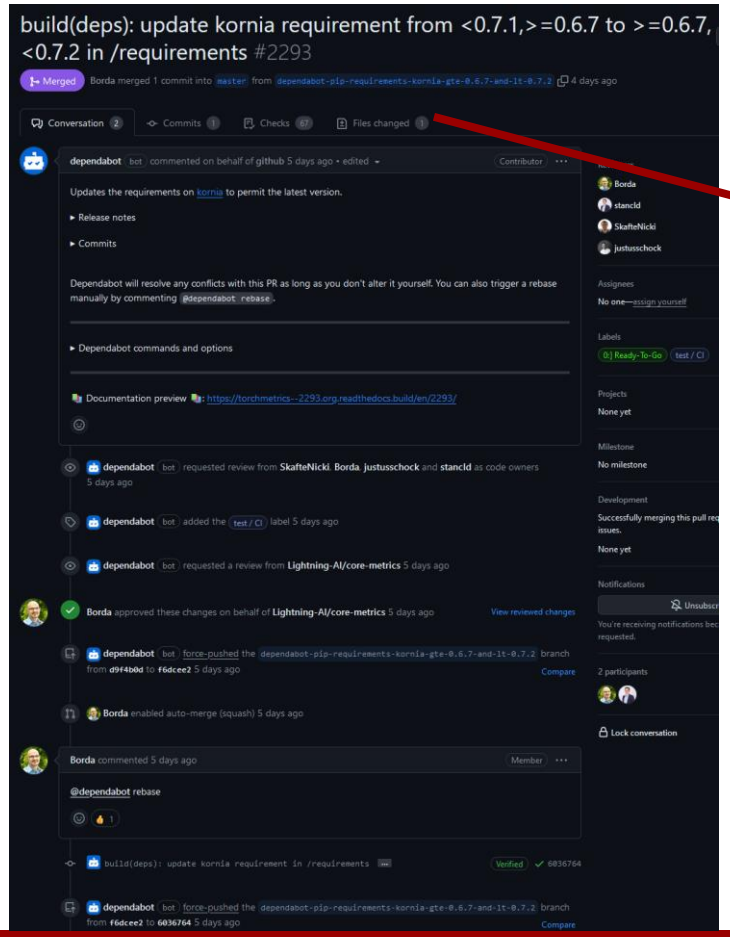
CI step 3: Automate tedious tasks with bots

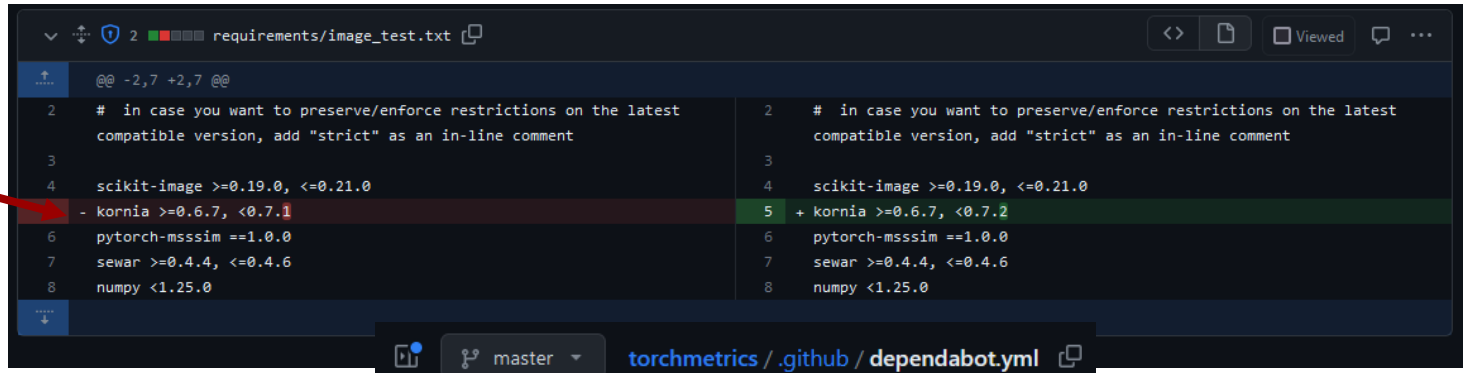


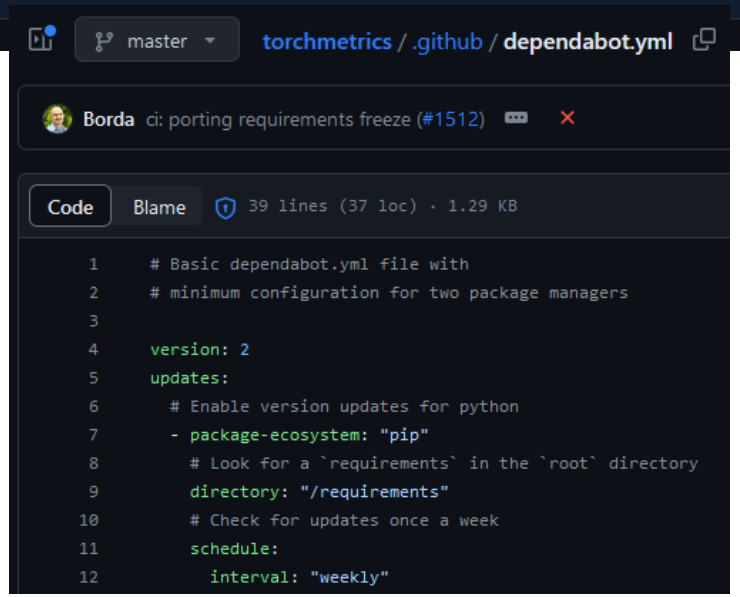


CI step 3: Automate tedious tasks with bots

Dependabot can help auto checking new releases of dependencies in your project







Summary of continues integration

1. Use version control

2. Write (unit-)test for your code

3. Automate build + test

Meme of the day