Table of contents (../toc.ipynb)

Continuous integration (CI)

- Continuous integration (CI) is a concept where changes from developers are integrated and tested into main version of software within short cycles.
- CI is often twinned with CD, which mean continuous delivery. In CD, the concept is to be able to create a deliverable software product for costumers in short cycles.
- CI/CD (we will just consider CI in the latter) fosters automation of manual tasks and provides foundation for distributed development on short cycles.
- Short cycles means daily and often many times a day.

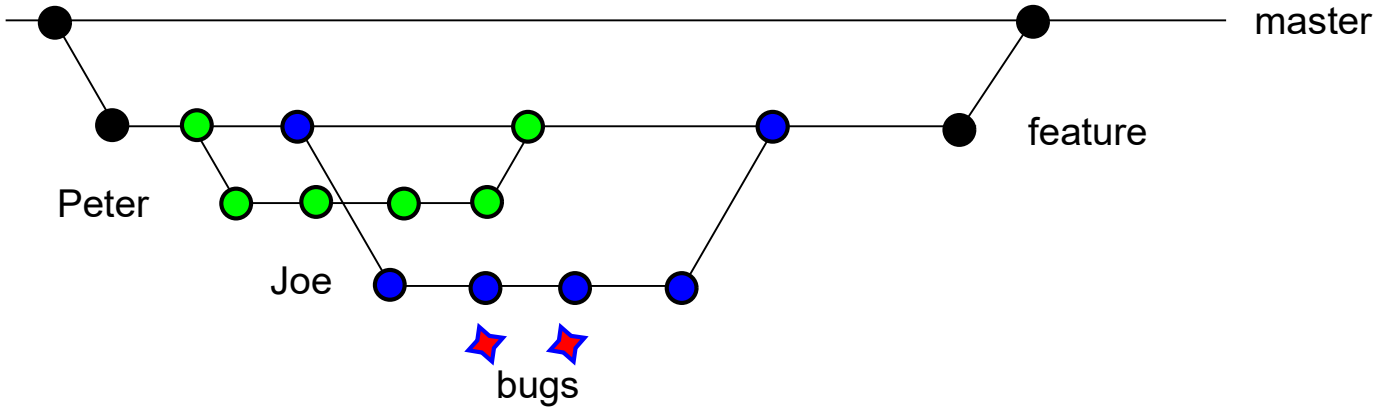
Why integrate in short cycles?

We have seen previously that **cost of change is roughly an exponential function of time** of development process. Hence, the earlier you find errors the cheaper and easier they can be resolved.

Add to this, **software development in teams requires to split larger chunks of work** (often called a feature) into smaller ones (often called story) so that developers can work simultaneously. **Without CI, tests can be just applied after integration of all stories**. If this fails, the entire feature is incomplete.

Take a look at this figure where a feature is developed by Peter and Joe. With CI, each commit (circle) of Peter and Joe triggers a test of the entire software.

Note that two commits of Joe contained a bug. However, due to direct feedback of CI, Joe noticed directly that something went wrong and he resolved the bug in the next commit. Therefore, Peters changes were not corrupted by the bugs and integration went well through.



Therefore, CI ensures that the software remains *soft*. Developers do not hesitate to change the software and small changes can be integrated very fast.

Requirements for CI

If you want to apply in your development - which is a must today I would say - your project needs to address these requirements:

- The software is under version control (git).
- Tests are either together with code or before coding written (test driven development).
- The test and build steps are automated and this automation is also under version control.
- A CI software triggers tests and builds on events like commits or pull requests and stores and returns results.
- Build slaves are available (hardware PCs or cloud services) to test and build the software.
- Tests and build are fast.
- You need a cultural environment that fosters failing fast and prioritizes bug fixes.

You see, there are many requirements. However, if you address them, the quality and maintainability of software will improve drastically.

Theory or practice?

We could of course proceed with theory about CI. However, I guess the mindset behind CI is more important.

- Automate the boring stuff! People believing that a checklist and manual quality tests are a way to ensure quality have not much practical experience.
    - No one wants and will go through a checklist in daily business and run manual tests.
- Test often, test fast!
- Use all weapons you have (static analysis, dynamic tests, integration tests,...).

CI techniques are developing very fast and that is why we explore some tools shortly before we get our hands dirty and set up a CI pipeline.

CI solutions

CI is state of the art in software development and there are many tools and solutions available. Here a short incomplete list of solutions.

- Jenkins (https://jenkins.io/) the probably most applied automation server. Very flexible and rich of features on the price of not so simple to learn and maintain from my experience. With Jenkins, you can do anything, but you need rather expert knowledge.
- Travis CI (https://travis-ci.org/) a more lightweight solution compared with Jenkins for CI. Easy to use and directly integrated as service in Github. I used Travis CI for instance to test all Jupyter notebooks of this course. Take a look here:

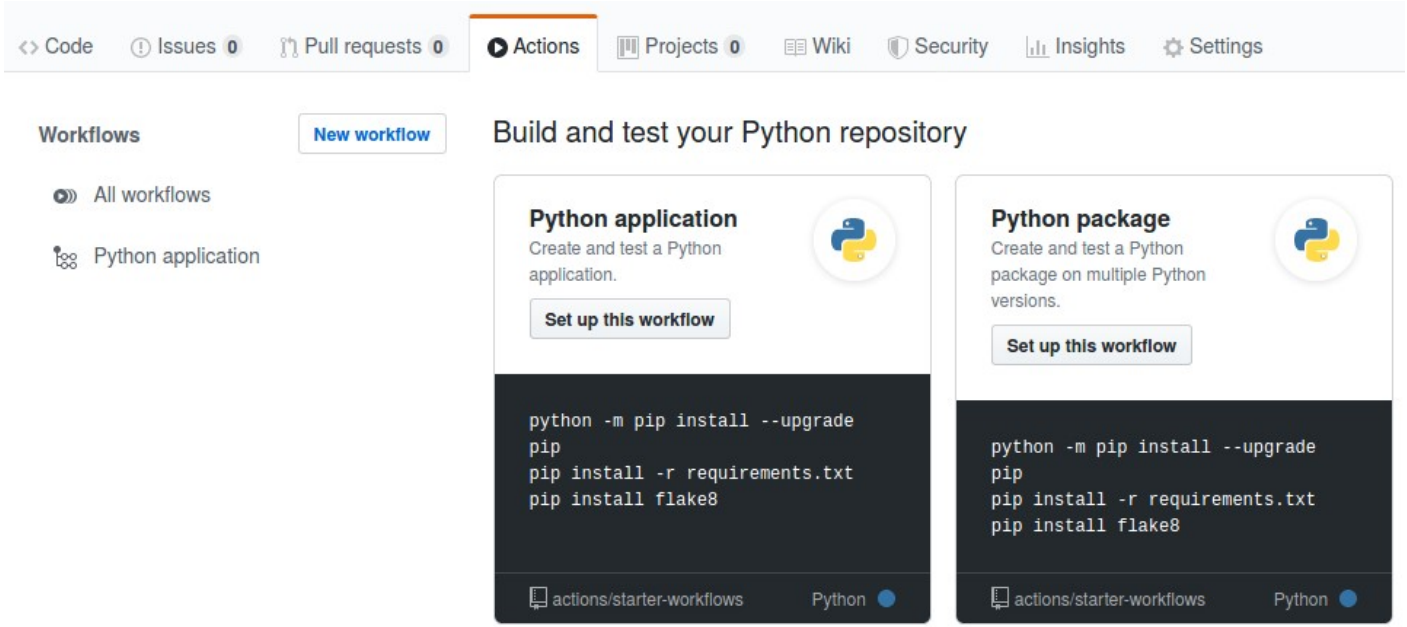
  

  (https://travis-ci.com/StephanRhode/py-algorithms-4-automotive-engineering)
- GitHub Actions (https://github.com/features/actions) a quite recent CI feature of GitHub which of course is very easy to use. Most lightweight solution for GitHub repos currently I would say.
- Tens of other solutions! Just search for CI and many tools will show up.

Exercise: Build a CI pipeline in GitHub-Actions (30 minutes)

Now we want to create our first small CI pipeline with the next steps. This will be a rather long exercise in several steps.

- Login into your GitHub account and create a repository with an open source license of your choice.
- Clone this repository to your computer.
- Create commit and push the following empty files to the repo: `my_source.py`, `test_my_source.py`, and `requirements.txt`.
- Go to GitHub, open this repository and click on actions and select `Python application`.



Check structure and `pythonapp.yml`

The repository should have now this structure:

```
|___ .github
    |___ workflows
        |___ pythonapp.yml
|___ README.md
|___ LICENSE
|___ my_source.py
|___ test_my_source.py
|___ requirements.txt
```

Open the `pythonapp.yml` file and try to understand what the code tells.

`requirements.txt`

Now copy and paste

```
pytest
pytest-cov
flake8
pylint
pydocstyle
```

into `requirements.txt`

my_source.py

Copy and paste

```
"""Contains code under test for solution_ci."""

def add_func(a, var_two):
    """
    Add two variables.

    Args
        a: first variable
        var_two: second variable

    """
    return a + var_two
```

into `my_source.py`.

test_my_source.py

Copy and paste

```
import pytest
from my_source import add_func


def test_add_one():
    assert add_func(2, 5) == 7


@pytest.mark.xfail
def test_add_fail():
    assert add_func(5, 5) == 9
```

into `test_my_source.py`.

Append `pythonapp.yml`

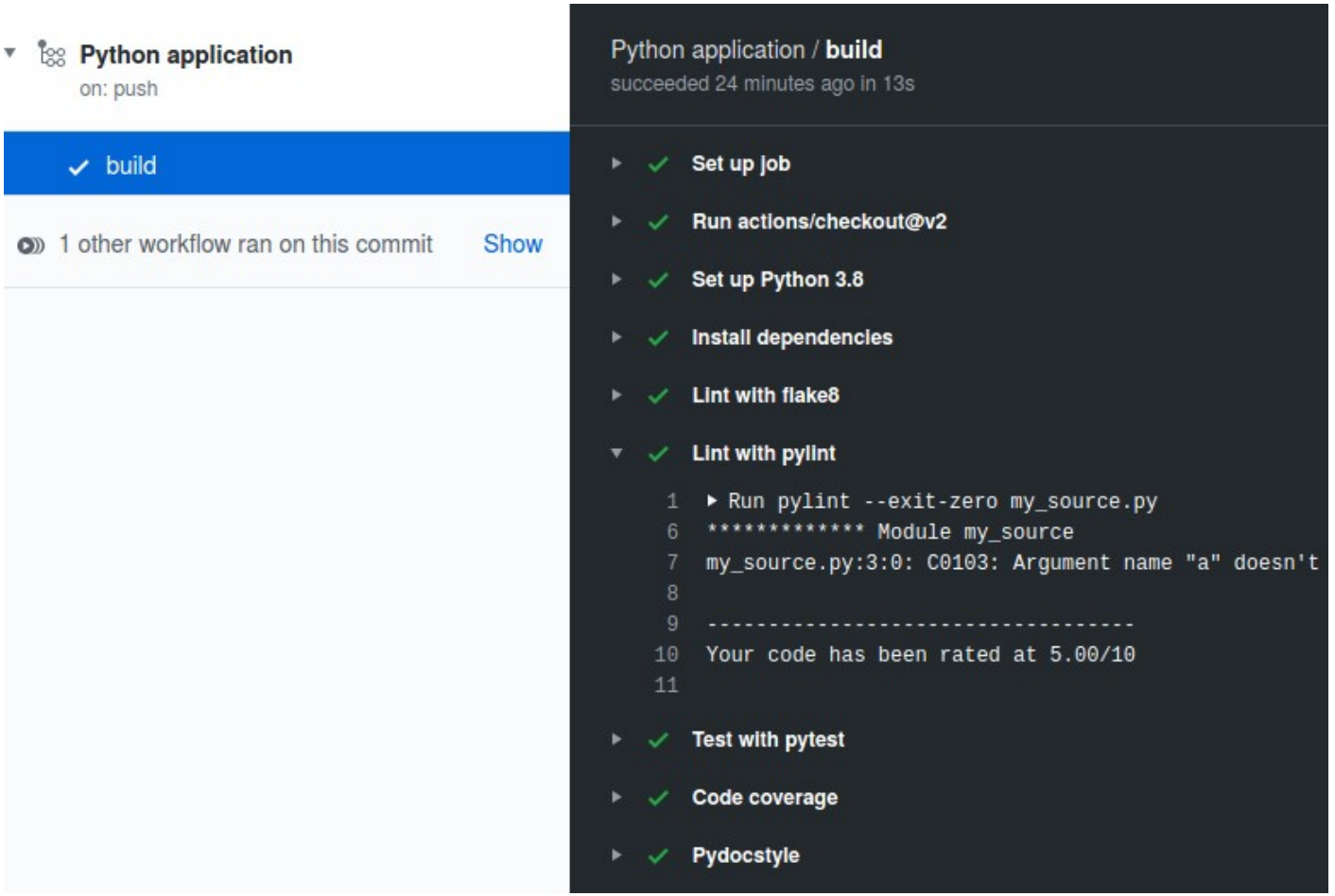
And finally append these lines to `pythonapp.yml` to trigger additional build steps.

```
yml
    - name: Lint with pylint
      run: |
        pylint --exit-zero my_source.py
    - name: Test with pytest
      run: |
        pytest
    - name: Code coverage
      run: |
        pytest --cov-report term-missing --cov
    - name: Pydocstyle
      run: |
        pydocstyle
```

Check CI on GitHub

Now commit and push these changes and take a look at the actions field in GitHub.

- Does the build run?
- Check your emails, you should receive notifications from GitHub.
- What kind of messages are provided?



Try a bug

- Now, try to change one assertion in `test_my_source.py`. Does now the build fail?

Improve pylint rating

- Take a look at pylint field. It gives a hint to improve the code.
- Try to resolve it.

Solution

You can compare your solution with this repository

https://github.com/StephanRhode/solution_ci (https://github.com/StephanRhode/solution_ci)

Books on CI

Please find here two books on continuous integration topic.

- The Hitchhiker's Guide to Python [Schlusser2016] (../references.bib).
- Continuous integration improving software quality and reducing risk [Duvall2013] (../references.bib).

Tutorial on Python CI

Add to this, there are numerous web courses out there and one great and free tutorial which explains how to apply CI for Python is on realpython.com https://realpython.com/python-continuous-integration/ (https://realpython.com/python-continuous-integration/).