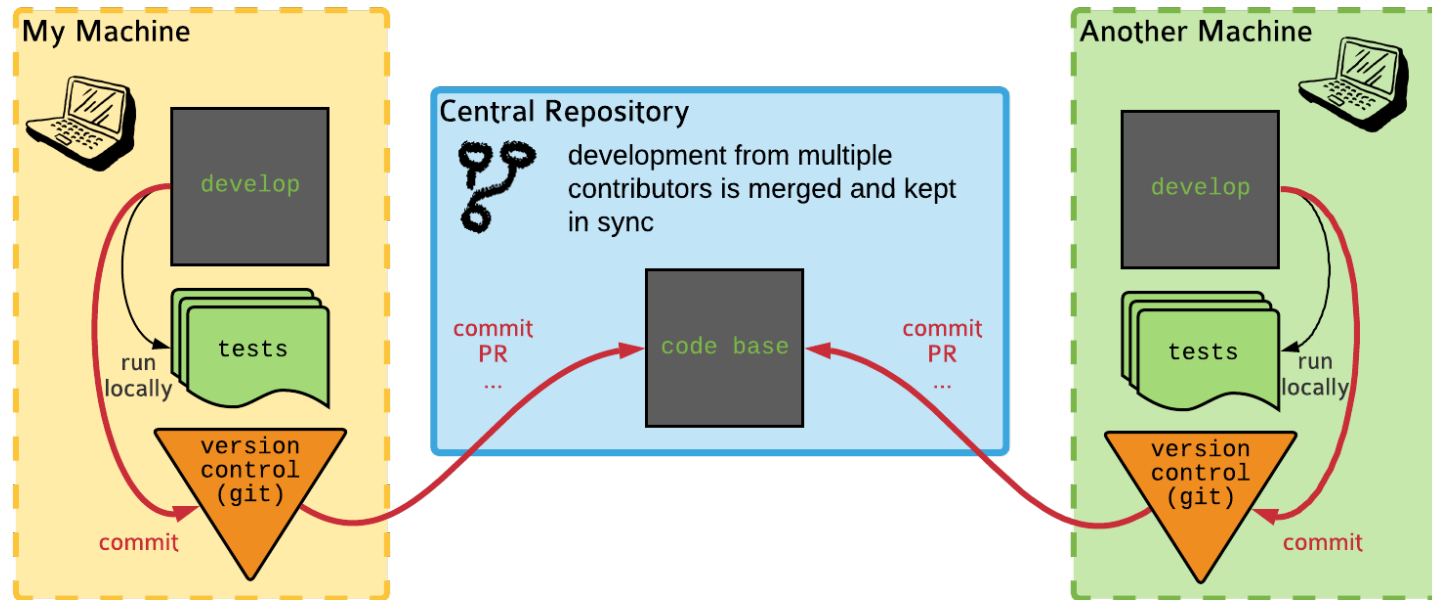


Continuous Integration

Because you're worth it, continuously

Lisa Schwetlick and Pietro Berkes

Collaborative Development without CI



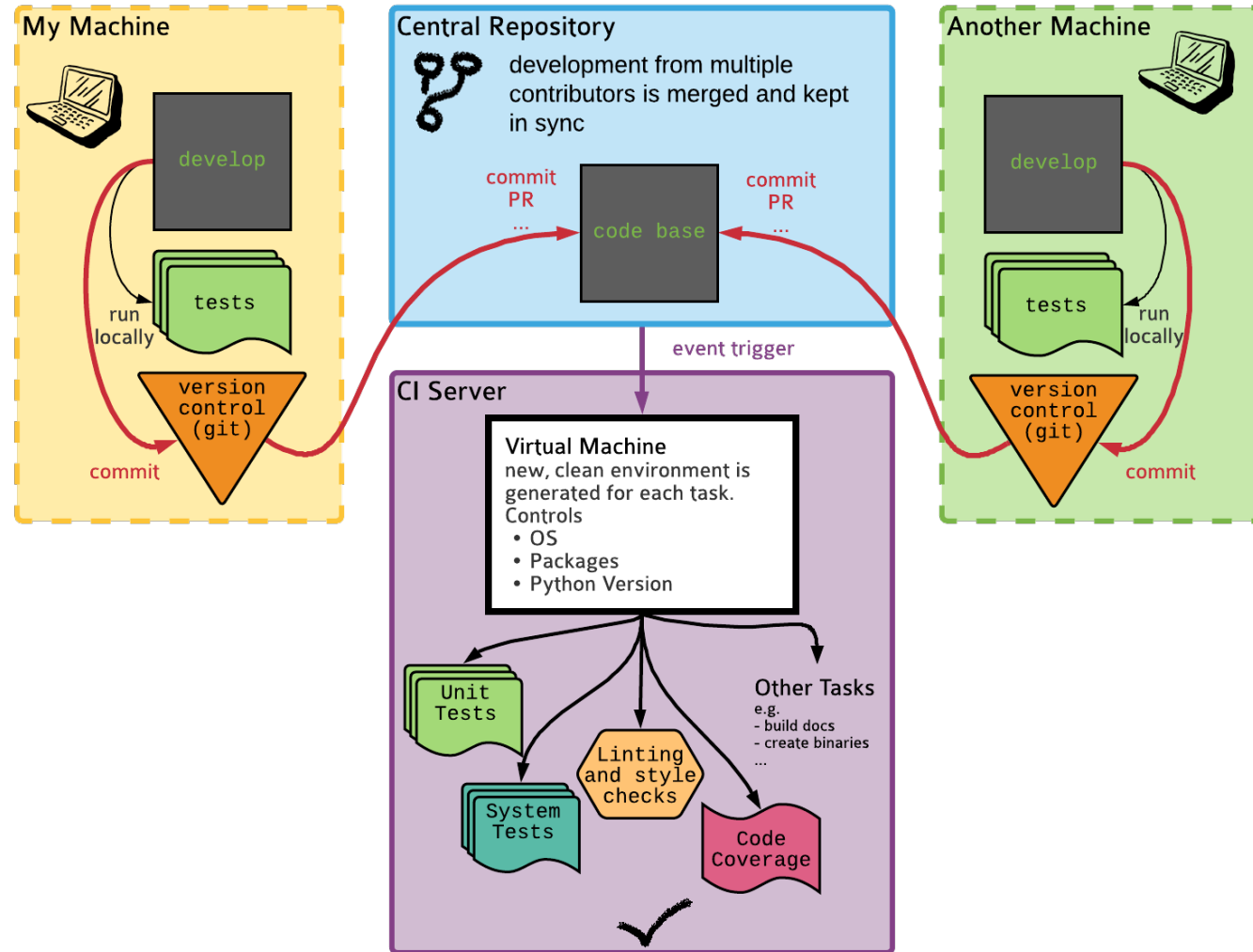
Potential issues

- The tests might pass on one machine and/or the other, but not in a third-party environment (versions, OS, etc.)
- A maintainer needs to ensure that the software works on all the supported combinations of versions / OSs
- A maintainer needs to create and upload artifacts like binary packages, documentation, etc

Continuous Integration

- Continuous Integration is a set of tools and practices to make sure that a project with many contributors (≥ 1) runs smoothly
- One goal is to automatize the non-coding tasks:
 - make sure that the tests always pass
 - check for style consistency
 - build packages for distribution on multiple architectures
 - build documentation
- Another goal is to solve the “it works on my machine” problem

Collaborative Development with CI



The CI tasks that you'll find 95% of the time

- **Task 1: Run test when a PR is created**
 - **Event trigger:** PR is created
 - **Action:** Run all tests for different Python versions

- **Task 2: Release package when version is bumped**
 - **Event trigger:** Version is bumped
 - **Action:** Create binary packages for Linux, Mac, Windows and upload them to a package repository

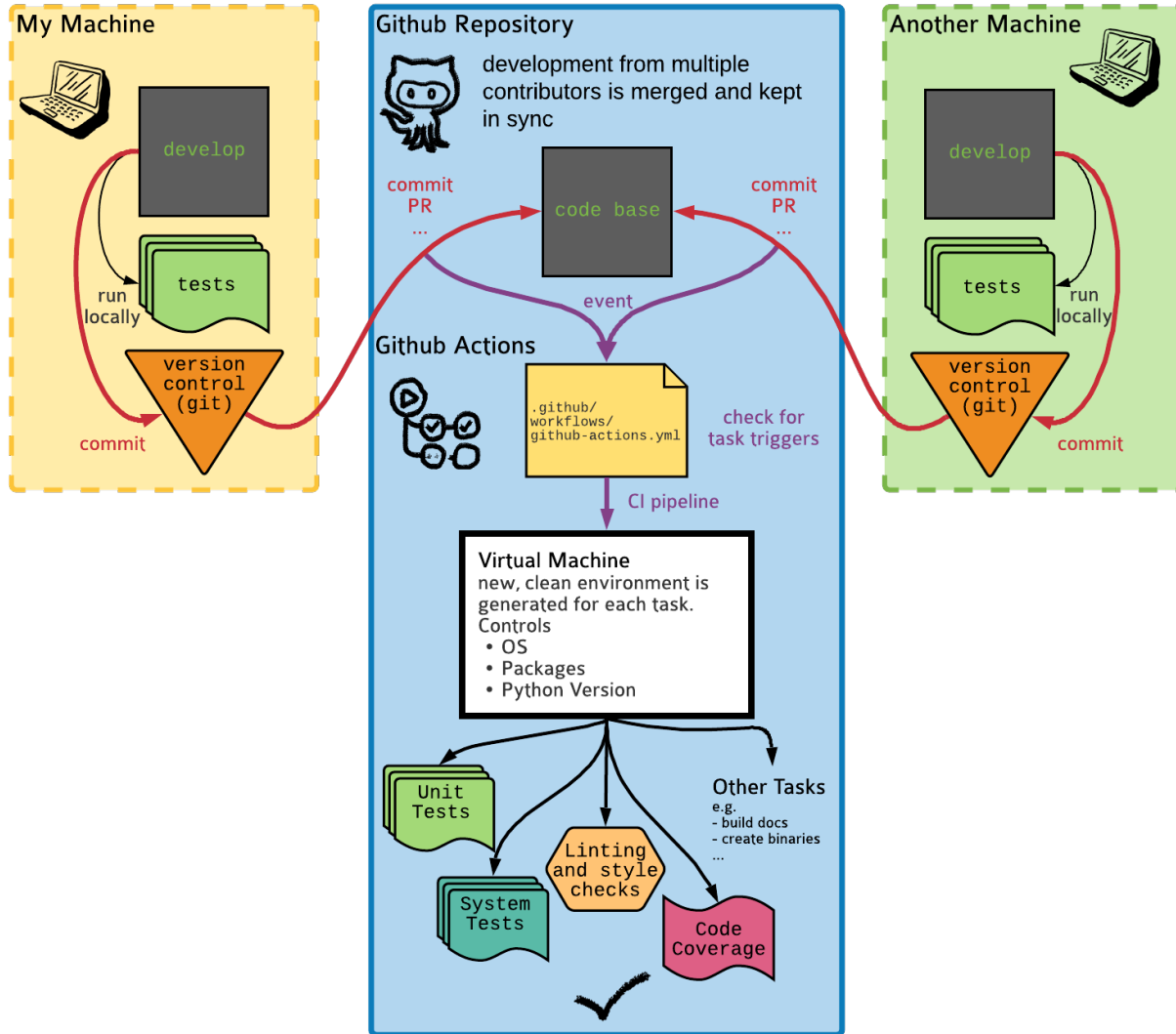
- **Task 3: Publish documentation on request**
 - **Event trigger:** Repository is tagged in a certain way
 - **Action:** Build and publish the documentation

CI options



GitHub Actions is at the moment the preferred choice for many open source projects. It is very flexible and well integrated with GitHub.

Collaborative Development with GitHub Actions



GitHub acts as both the central repository and the CI server, but the rest is the same

GitHub Actions basic ideas

An event occurs, it has an associated commit SHA (e.g., a PR is opened or a commit tag is pushed)



GitHub searches for config files in `.github/workflows` at that SHA, and looks if there is a trigger that matches the event



It then creates a virtual machine as specified in the config file and runs the commands listed there

GitHub Actions basic ideas

- The outcome is logged and if the job exits cleanly it is marked as “passed” otherwise “failed”

The image shows two screenshots of the GitHub Actions interface. The top screenshot shows a workflow that has passed, with a green checkmark and the text "All checks have passed" and "36 successful and 2 neutral checks". Below this, a list of jobs is shown, all with green checkmarks and "Successful" status. The bottom screenshot shows a workflow that has failed, with a red circle and the text "Some checks were not successful" and "35 successful, 1 cancelled, 1 failing, and 2 neutral checks". Below this, a list of jobs is shown, with one job marked with a red 'X' and "Failing" status, and another marked with a blue 'i' and "Cancelled" status.

Passed Workflow:

- ✓ All checks have passed (36 successful and 2 neutral checks)
- ✓ Build_Test / lint (pull_request) Successful in 19s
- ✓ Pull Request Labeler / pr-labeler (pull_request_target) Successful in 3s
- ✓ Build_Test / smoke_test (pull_request) Successful in 6m
- ✓ Build_Test / basic (3.8) (pull_request) Successful in 8m
- ✓ Build_Test / basic (3.9) (pull_request) Successful in ...
- ✓ Build_Test / basic (3.10.0-beta.3) (pull_request) Su...
- ✓ This branch has no conflicts with the base branch

Failed Workflow:

- ⓘ Some checks were not successful (35 successful, 1 cancelled, 1 failing, and 2 neutral checks)
- ✓ Build_Test / smoke_test (pull_request) Successful in ...
- ✓ Build_Test / basic (3.7) (pull_request) Successful in 6m
- ⓘ Build_Test / basic (3.9) (pull_request) Cancelled after 9m — basic (3.9)
- ✗ Build_Test / basic (3.10.0-rc.1) (pull_request) Failing after 8m — basic (3.10.0-rc.1)
- ✓ Build_Test / debug (pull_request) Successful in 7m
- ✓ Build_Test / blas64 (pull_request) Successful in 7m
- ✓ This branch has no conflicts with the base branch

GitHub Actions

The screenshot displays the GitHub Actions interface. At the top, a navigation bar includes links for Code, Issues, Pull requests, Actions (highlighted), Projects, Wiki, Security, Insights, and Settings. Below this, the 'Workflows' section is visible, featuring a 'New workflow' button and a list of workflows, with 'All workflows' selected. A specific workflow, 'Run all the tests for PRs', is partially visible. The main area is titled 'All workflows' and shows 'Showing runs from all workflows'. A search bar for 'Filter workflow runs' is present. Below the search bar, a table lists '2 workflow runs' with columns for Event, Status, Branch, and Actor. The first run, 'fix workflow error', is successful (green checkmark) and occurred 1 minute ago on the 'master' branch. The second run, 'add workflow', failed (red X) 4 minutes ago on the 'master' branch.

Event	Status	Branch	Actor
fix workflow error	Success	master	...
add workflow	Failure	master	...

GitHub config file: Example

Run tests every time a PR is opened or a commit is pushed

The configuration file is saved in `.github/workflows` , with a name related to its task, e.g. `run-tests.yml`

```
name: Run all the tests for PRs
```

```
on:
```

```
[push, pull_request]
```

Specifies the events that trigger the jobs below

```
jobs:
```

```
run-tests:
```

```
runs-on: ubuntu-latest
```

The type of virtual machine used to run the workflow

```
steps:
```

```
- uses: actions/checkout@v2
```

```
- name: Set up Python
```

```
  uses: actions/setup-python@v2
```

```
  with:
```

```
    python-version: 3.9
```

```
- name: Install dependencies
```

```
  run:
```

```
    python -m pip install pytest numpy
```

```
- name: Test with pytest
```

```
  run:
```

```
    pytest -sv hands_on/pyanno_voting
```

Multiple steps are used to set up the environment so that we can run the tests.

Notice the use of community actions

The command that we wanted to execute all along

GitHub Actions reference

- **Introduction:**

<https://docs.github.com/en/actions/learn-github-actions/introduction-to-github-actions>

- **Events that can trigger actions, and their config options:**

https://docs.github.com/en/actions/reference/events-that-trigger-workflows#pull_request

- **Catalog of community actions:**

<https://github.com/marketplace?type=actions>

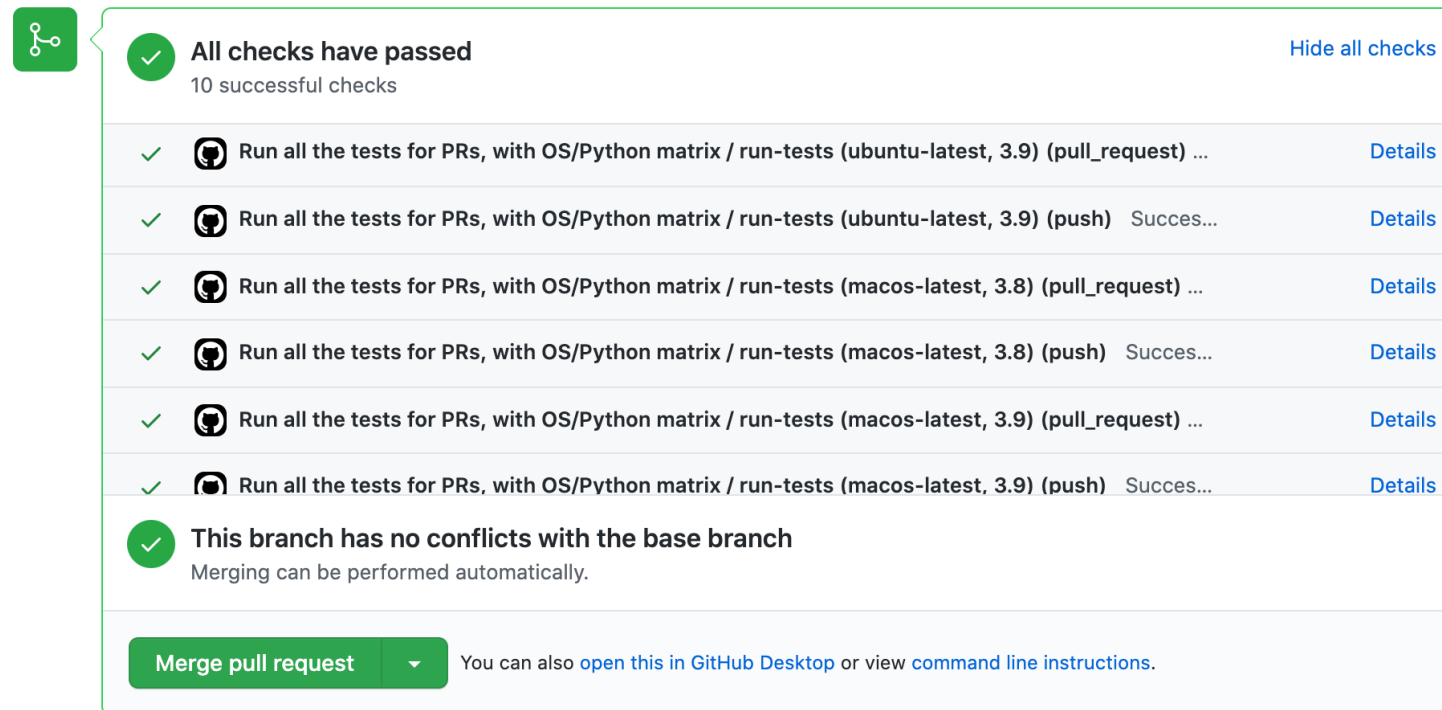
Hands On!

Add a CI pipeline to your logistic function project!


1. In your local version of the project make a folder `.github/workflows`
 2. Create a file called `run_test_on_push.yml`
 3. Write your configuration file to run the tests every time someone pushes some commits or every time someone creates a pull request
 4. Commit and push the changes to GitHub
 5. Check the actions tab of your GitHub repo to see if it worked
- Bonus: check the GitHub actions documentation and modify the configuration file so that the tasks run only for pushes and PRs against the branch `main`













Matrix configuration

- If your project supports multiple OSes, Python versions, and library version, you might want to run our tests on all the combinations of those



The screenshot displays a GitHub Actions workflow status for a pull request. At the top left is a green icon with a white branching diagram. The main status is a green checkmark followed by the text "All checks have passed" and "10 successful checks". To the right of this status is a link "Hide all checks". Below this are seven individual check items, each with a green checkmark, a GitHub Actions logo, and a description of the test run. The descriptions include the workflow name "Run all the tests for PRs, with OS/Python matrix / run-tests" and the specific configuration (OS, Python version, and event type). Each item has a "Details" link to its right. The last item in the list is "This branch has no conflicts with the base branch" with the subtext "Merging can be performed automatically." At the bottom of the status box is a green button labeled "Merge pull request" with a dropdown arrow, followed by the text "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)."

 **✓ All checks have passed** [Hide all checks](#)
10 successful checks

-  **✓**  Run all the tests for PRs, with OS/Python matrix / run-tests (ubuntu-latest, 3.9) (pull_request) ... [Details](#)
-  **✓**  Run all the tests for PRs, with OS/Python matrix / run-tests (ubuntu-latest, 3.9) (push) Succes... [Details](#)
-  **✓**  Run all the tests for PRs, with OS/Python matrix / run-tests (macos-latest, 3.8) (pull_request) ... [Details](#)
-  **✓**  Run all the tests for PRs, with OS/Python matrix / run-tests (macos-latest, 3.8) (push) Succes... [Details](#)
-  **✓**  Run all the tests for PRs, with OS/Python matrix / run-tests (macos-latest, 3.9) (pull_request) ... [Details](#)
-  **✓**  Run all the tests for PRs, with OS/Python matrix / run-tests (macos-latest, 3.9) (push) Succes... [Details](#)

✓ This branch has no conflicts with the base branch
Merging can be performed automatically.

[Merge pull request](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

GitHub Actions workflow with matrix config

```
Name: Run all the tests for PRs, with OS/Python matrix
```

```
on:  
  [push, pull_request]
```

```
jobs:  
  run-tests:  
    runs-on: ${{ matrix.os }}
```

```
strategy:  
  matrix:  
    os: [ubuntu-latest, macos-latest]  
    python-version: [3.8, 3.9]
```

The strategy/matrix section specifies lists of parameters. The workflow is run for all combinations

```
steps:  
- uses: actions/checkout@v2  
- name: Set up Python ${{ matrix.python-version }}  
  uses: actions/setup-python@v2  
  with:  
    python-version: ${{ matrix.python-version }}  
- name: Install dependencies  
  run:  
    python -m pip install pytest numpy  
- name: Test with pytest  
  run:  
    pytest -sv hands_on/pyanno_votin
```

GitHub Actions workflow with matrix config

Name: Run all the tests for PRs, with OS/Python matrix

```
on:  
  [push, pull_request]
```

```
jobs:
```

```
  run-tests:
```

```
    runs-on: ${{ matrix.os }}
```

This is how we refer to the matrix parameters in the config file

```
  strategy:  
    matrix:  
      os: [ubuntu-latest, macos-latest]  
      python-version: [3.8, 3.9]
```

```
  steps:
```

```
    - uses: actions/checkout@v2
```

```
    - name: Set up Python ${{ matrix.python-version }}
```

```
      uses: actions/setup-python@v2
```

```
      with:
```

```
        python-version: ${{ matrix.python-version }}
```

```
    - name: Install dependencies
```

```
      run:
```

```
        python -m pip install pytest numpy
```

```
    - name: Test with pytest
```

```
      run:
```

```
        pytest -sv hands_on/pyanno_votin
```


GitHub Actions reference

- **Types of virtual machines available on GitHub Actions:**

<https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners#supported-runners-and-hardware-resources>

- **setup-python community action, all available Python flavors and versions:**

<https://github.com/marketplace/actions/setup-python>

Hands On!

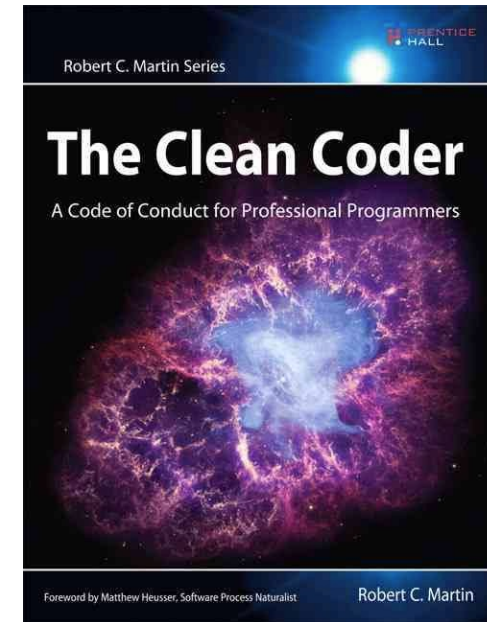
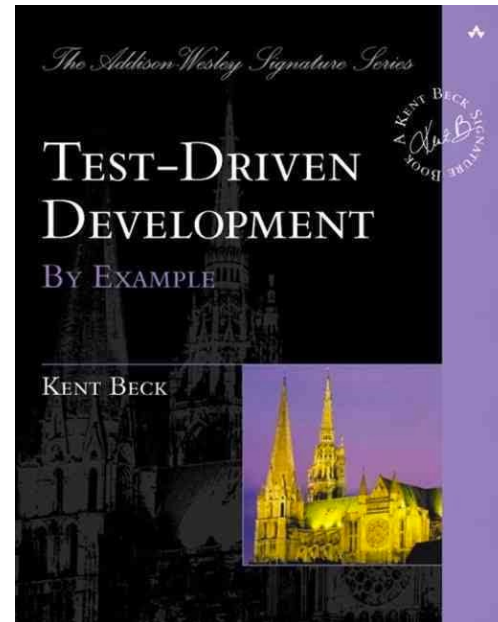
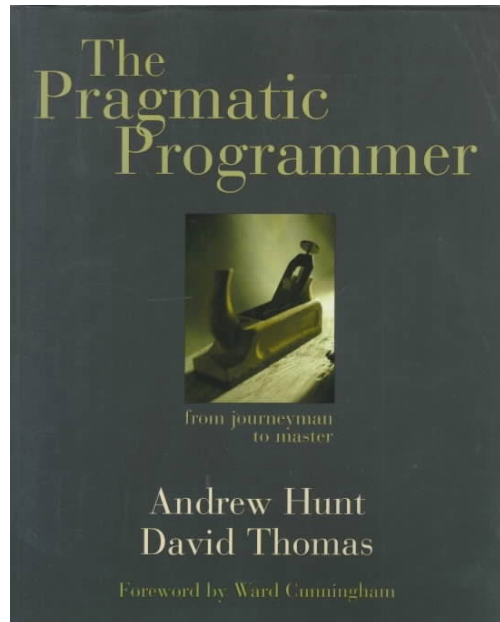
- Adapt your git actions configuration file `run_test_on_push.yml` to run the testing workflow on Python 3.7, 3.8, 3.9, and on Linux and Windows

Conclusions

- It takes a bit of time to set up and debug a Continuous Integration workflow, but it's a good investment that can save you a lot of time later on!



Recommended reading



Thank you!

Bonus: Security

- Some tasks require “secrets” like usernames and passwords, for instance to upload the documentation to a remote machine.
- Do not push passwords and other sensitive information to a repository, not even a private one! Each CI system has a way to deal with secret safely.

TOP SECRET

Bonus: Security

- Secrets in GitHub actions can be added under `Settings` -> `Secrets`. The secret is stored encrypted by GitHub, and decrypted at the moment of running the workflow
- Secrets can then be referred to in the workflow as

```
steps:  
  - name: Hello world action  
    with: # Set the secret as an input  
      super_secret: ${ secrets.SuperSecret }  
    env: # Or as an environment variable  
      super_secret: ${ secrets.SuperSecret }
```


Bonus: Examples of handling secrets

```
name: Reveal a secret when the repository is tagged as something
starting by secret

on:
  push:
    tags:
      - 'secret*'

jobs:
  reveal-secret:
    runs-on: ubuntu-latest

    steps:
      - shell: bash
        env:
          SECRET_MSG: ${ secrets.TOP_SECRET }
        run: |
          echo The secret is "$SECRET_MSG"
          if [ "$SECRET_MSG" = 'do not tell anyone' ]; then
            echo matches
          fi
```

Details available at

<https://docs.github.com/en/actions/reference/encrypted-secrets>