Continuous Integration with GitHub Actions
Software Engineering for Scientists

https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

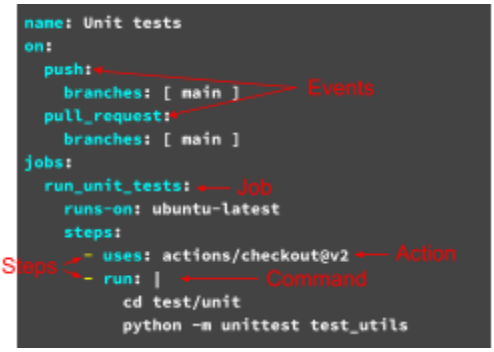
Continuous Integration (CI) is a development strategy where changes are continuously added to the main code base as you are working on them, instead of waiting until some milestone is reached and many features are added at once. CI reduces the pain in attempting to integrate many changes from many developers at once and uses automated building and testing to ensure that new code does not break the existing system. CI relies on the developer to include adequate tests for the features that they add and is best suited for a test-driven environment.

Ideally, before the code is added to the remote repository's main branch, it is tested in an independent environment to ensure that successful tests do not depend on some state that exists only in the developer's environment. An independent test environment also ensures that the developer has committed all the updates required for the new feature. GitHub actions give us both automated tests and the independent test environment.

GitHub Actions

GitHub actions is a platform for automating developer workflows. While GitHub actions can do many things (e.g., deploy code, update conda or pip repositories, create docker images, etc.), we will use it to ensure that code committed to our remote repository passes our tests. More specifically, we will create a workflow that will run our unit and functional tests every time someone makes a pull request (or pushes code to the remote master, but we aren't doing that in this class) to the remote repository. From the results of these tests we can make a decision about whether to accept or reject the pull request. When a commit fails a test it will still change the remote repository, which is why we do not commit directly to our remote main repository.

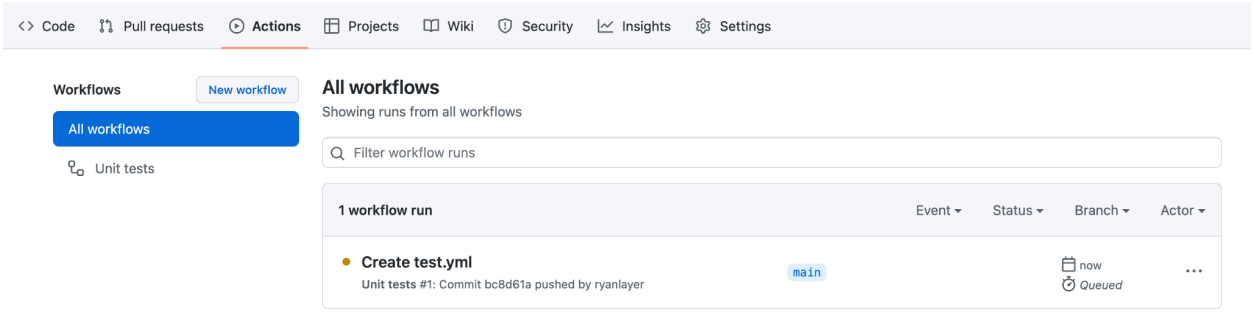
With GitHub actions, **events** trigger **workflows**. Workflows can also be triggered manually or run on a schedule. Workflows contain **jobs**, and each job runs inside of its own virtual machine and are completely independent of each other. Jobs have **steps** that either run code you wrote or some predefined **action**.



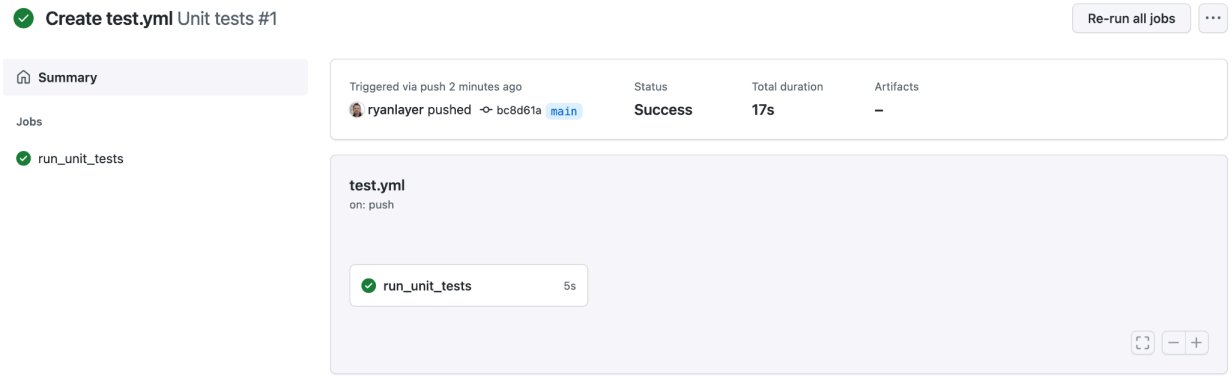
Workflows are individual YAML files stored in the `.github/workflows` directory in your repository. As soon as a properly formatted file is committed, the workflows will start running according to the actions they are bound to.

Monitoring workflows

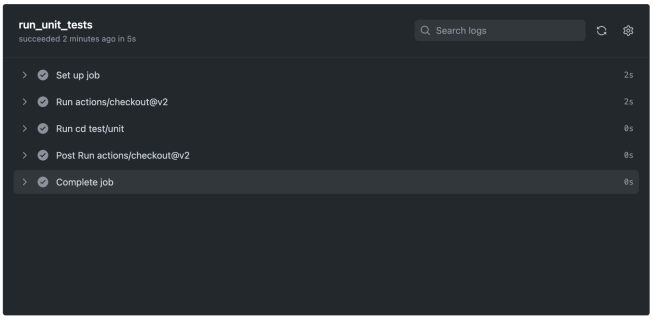
You can track the progress of your workflows by clicking on the Actions tab at the top of the page. This will give you a chronologically sorted list of every time a workflow was attempted. The one given below is associated with a push and has not started running yet. Once it starts, you can watch the progress by clicking on the commit message.



The next screen gives the status of each job.



Here we only have one and it has finished successfully. Clicking on the job name pulls up a list of every step. By default the commands are grouped and collapsed. To get more detail on an individual step, simply expand the step.



Anatomy of a workflow file

The status and result of workflows are organized by name to help you quickly understand what failed and what passed. Other than for display and organizational purposes, the name does not affect the workflow

```
name: Unit tests
```

When the workflow runs is defined by the set of events listed here. In the following example, the workflow is triggered by a push or pull requests to the main branch. There are many events. The full lists is here https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows

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

A workflow can have many jobs, each is organized under the job's name. Jobs are run in parallel (unless configured otherwise) and in independent virtual environments.

```
jobs:
  run_unit_tests:
```

Each job runs on a base operating system. Here we are choosing a popular Linux distribution. Other options include macOS and Windows. A job can be configured to run on more than one OS.

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

The actual commands associated with a job are listed under `steps`. There are two classes of steps.
Actions (uses) are specified by the `uses` key and provide a means to simplify and share complex steps. For example, `actions/checkout@v4` checks out the repository. Many actions have been made and shared by the GitHub community. You can browse them in the actions marketplace. https://github.com/marketplace?type=actions

Commands (run) are specified by the `run` key and provide command line access to the job. A multiline command is indicated with a pipe in the first line

```
    steps:
      - uses: actions/checkout@v4
      - run: |
          cd test/unit
          python -m unittest test_utils
```

When using an action, you can pass configuration parameters through the `with` key. These parameters are specific to the action being used. For example, the `mamba-org/setup-micromamba@v2` action supports setting an environment file and shell initialization:

```
      - uses: actions/checkout@v4
      - uses: mamba-org/setup-micromamba@v2
        with:
          environment-file: environment.yml
          init-shell: bash
```

By default, commands in `run` steps execute in the runner's default shell. You can override this with the `shell` key.

```
      - run: pycodestyle $(git ls-files "*.py")
        shell: bash -l {0}
```

This example forces the step to run inside a *login-enabled bash shell* (`-l`), which is needed because Micromamba writes its setup steps and environment modifications to the shell initialization files (e.g., `~/.bash_profile`, `~/.bash_login`, `~/.profile`) . With `bash -l {0}`, the login shell reads the init script, Micromamba's hooks are loaded, and the environment behaves correctly. Without `bash -l {0}`, the environment may not be activated; commands could fail with "command not found" or use the wrong Python.

test.yml

```yaml
name: Unit tests

on:
  push:
  pull_request:

jobs:
  run_style_check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: mamba-org/setup-micromamba@v2
        with:
          environment-file: environment.yml
          init-shell: bash
      - run: pycodestyle $(git ls-files "*.py")
        shell: bash -l {0}
  run_unit_tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: mamba-org/setup-micromamba@v2
        with:
          environment-file: environment.yml
          init-shell: bash
      - run: |
          cd test/unit
          python -m unittest test_utils
        shell: bash -l {0}
```