



Introduction to Continuous Integration

With unit tests and github actions

Learning Outcomes

By the end of this talk, you should be able to *automate* the *unit testing* of your code.

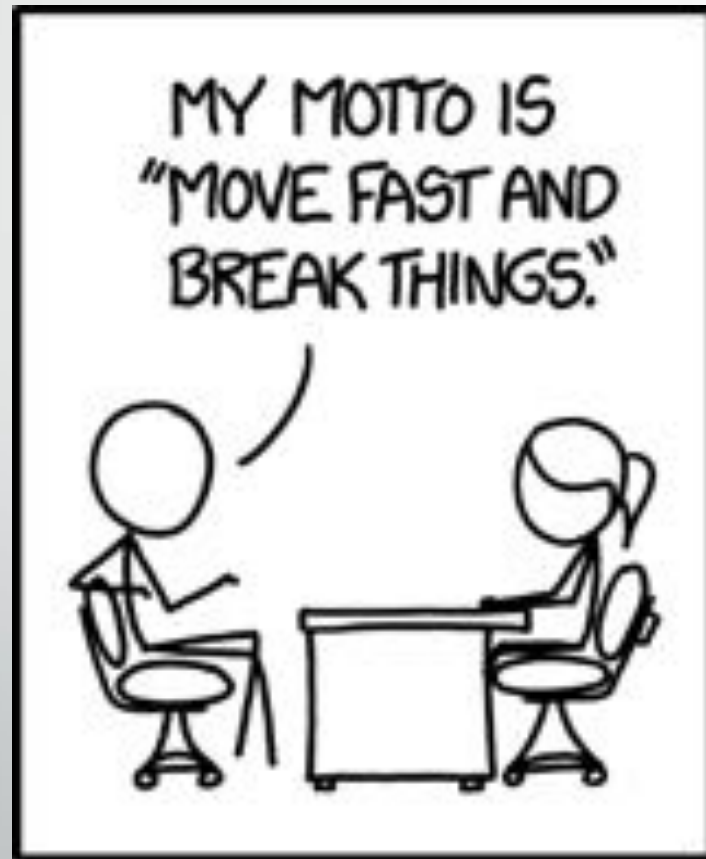
You should also understand how to do *test driven* development.

Literature this talk is based on: C. Chandrasekara, P. Herath *Hands-on GitHub Actions* and DSFP Session 3.

How do you manage a large project?

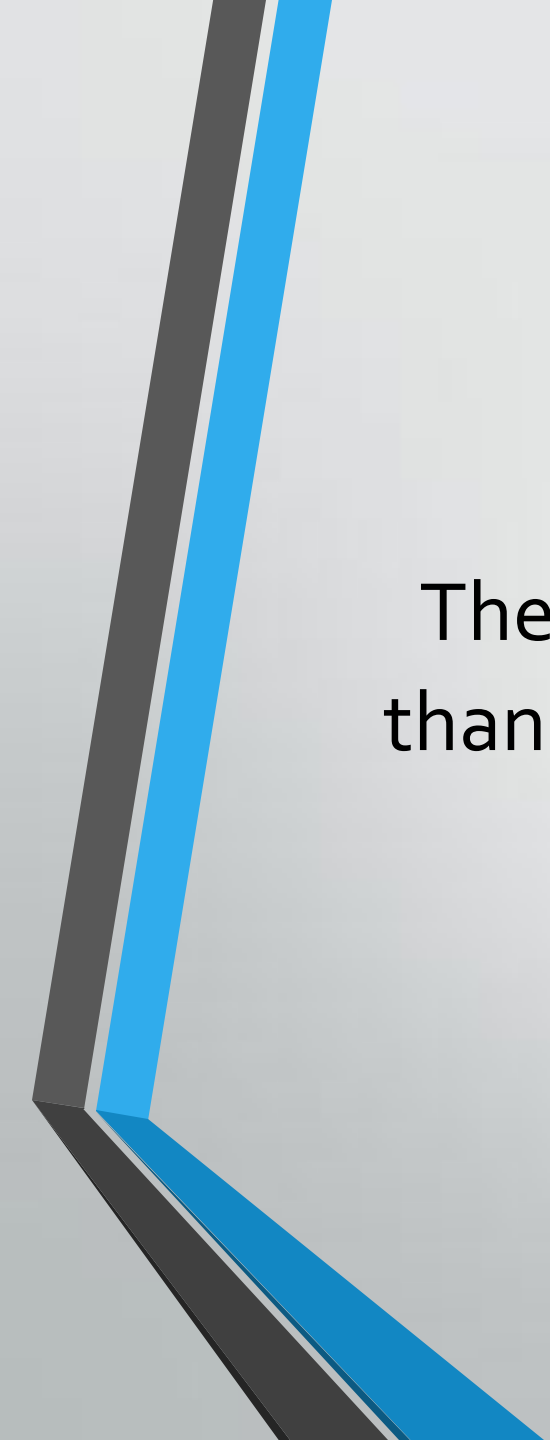
- As we saw yesterday, distributed version control provides a large number of benefits for managing software and ensuring scientific reproducibility
- Historically, software projects were treated like other engineering tasks, a large amount of planning and careful checks prior to *production*.
- This model has largely been replaced with the concept of *agile development*.

"Move fast and break things."



JOBS I'VE BEEN FIRED FROM

FEDEX DRIVER
CRANE OPERATOR
SURGEON
AIR TRAFFIC CONTROLLER
PHARMACIST
MUSEUM CURATOR
WAITER
DOG WALKER
OIL TANKER CAPTAIN
VIOLINIST
MARS ROVER DRIVER
MASSAGE THERAPIST



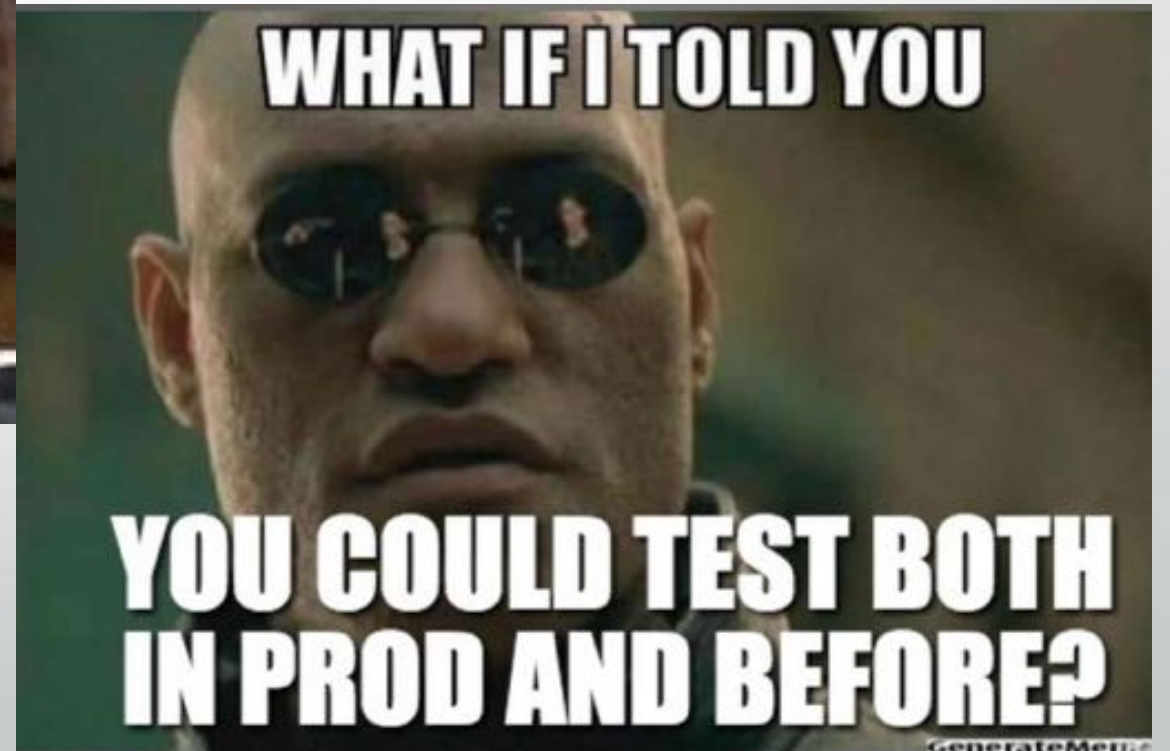
The **big idea** is to build code around testing, rather than planning. Write a test for a method, implement the method, then apply the test.



Continuous Integration

Continuous integration is a development paradigm in which changes to a software project are made and tested often. Usually these tests are automated and distributed.

Scientists Test on Production



Software testing paradigms

- Scientific workflows often involve *closed box* or *integration tests*. We write code and confirm that it reproduces specific results from the literature. We then trust it to give us reliable results when applied to new data, independent of the internal code mechanisms.
- We are less used to 'open box' unit tests – where individual parts of our code are tested early and often.

Principles of good unit tests

- Independence/modularity
- Simple and Fast
- Deterministic

Unit tests can introduce bugs too



Example Unit Test

```
def add(a,b):  
    c = a + b  
    return c
```

```
def test_add(a, b):  
    Added = add(1,1)  
    assert added = 2
```

Pytest, Continuous Integration, and Github Actions



Pytest directory structure

- Pytest is fairly inflexible, it expects a certain directory structure:
 - Packagename/__init__.py
 - Packagename/module_name.py
 - Packagename/test_module_name.py
 - Packagename/tests/test_module_name.py

Checking Coverage

- To have confidence in your code, tests should *cover* as much of your code as possible.
- We can check coverage with the `pytest --coverage` option.

Configuring Github Actions

- Github actions uses YAML - "YAML Ain't Markup Language" configuration files. These have a specific key: value pair syntax. For example,

```
# This workflow will install Python dependencies and run tests with a variety of Python versions
# For more information see: https://help.github.com/actions/language-and-framework-guides/using-python-with-github-actions

name: Python package

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

(More) on Configuring github Actions

Key: value pairs in a nested structure, so for, example, to setup tests to run automatically on a push to the main branch

on:

push:

branches: ["main"]

Can also schedule to run at a certain time every day with

on:

schedule:

cron: * 12 * * *

Configuring Github Actions: Jobs

Jobs are the "content" of a github action – they are the collection of all of the things that you want to occur.
The syntax is:

Jobs:

Build:

runs-on: <virtual machine> #"runner"

Steps:

- name: Install dependencies

run:

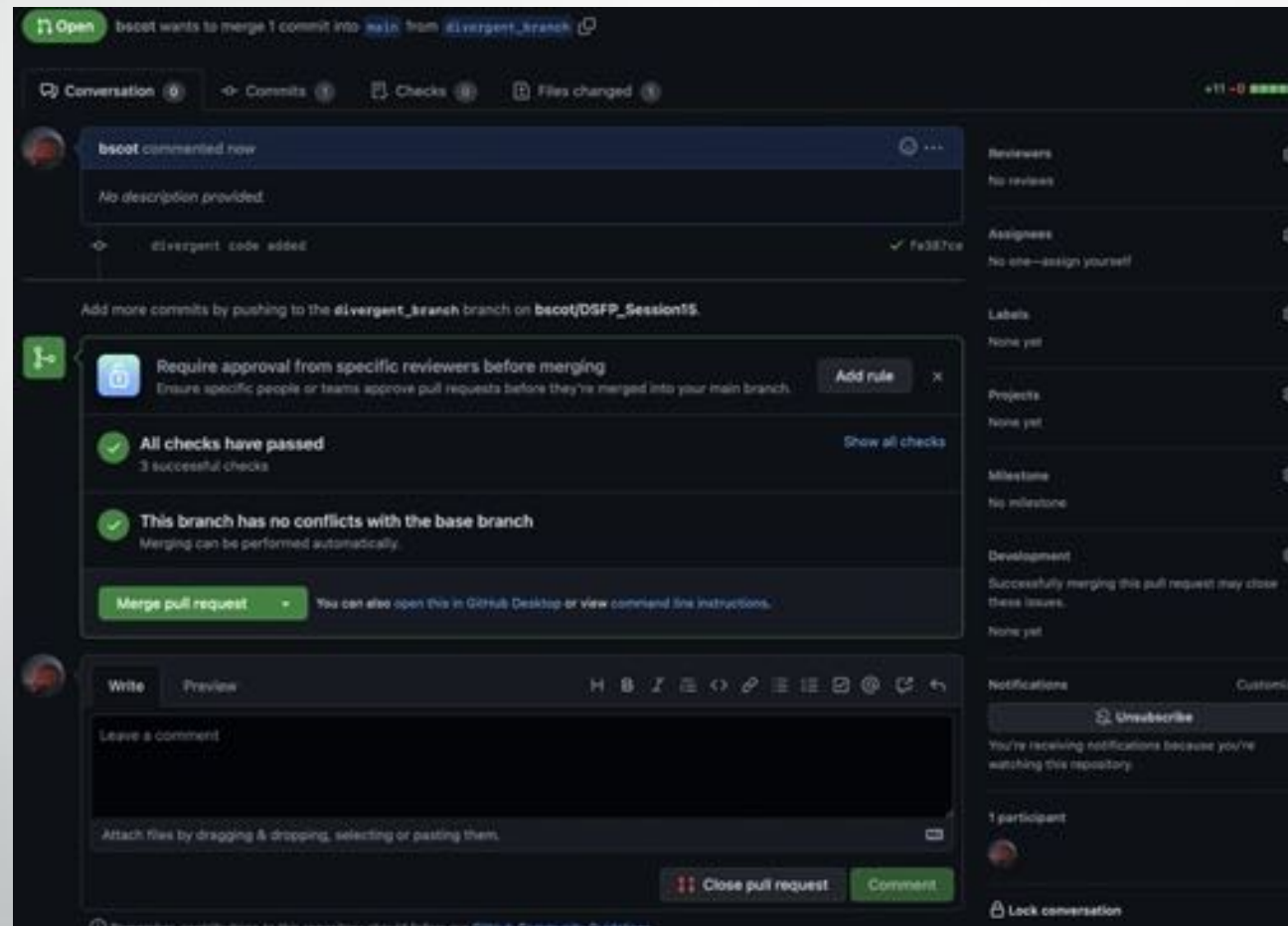
(some code to install dependencies)

- name: run tests

run:

(some code to run tests)

Continuous Integration in Github Pull Requests



Unit Test Exercise

- In the DSFP Session 15 Repository, you'll find a notebook that will walk you through setting up a github actions workflow and writing some unit-tests. You'll also find a template .yml file.
- We will continue to work on the clustering example from yesterday. You should have some cluster centers to work with, but if you do not, you can generate some mock cluster centers.

