

The Code Testing Paradigm

Or

How I Learned to Stop Worrying*

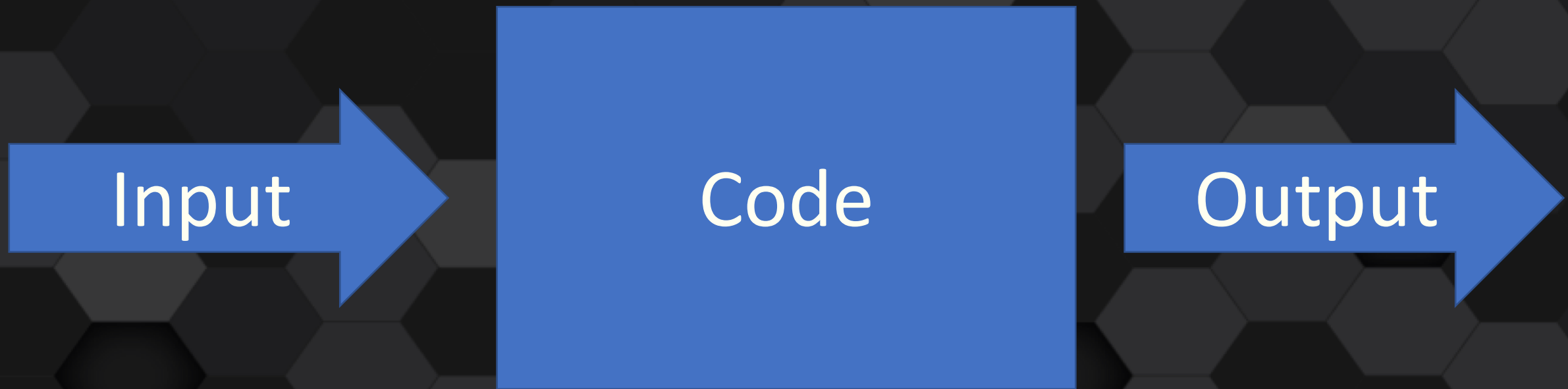
Jacob Seiler

*At least about this

Why Test?

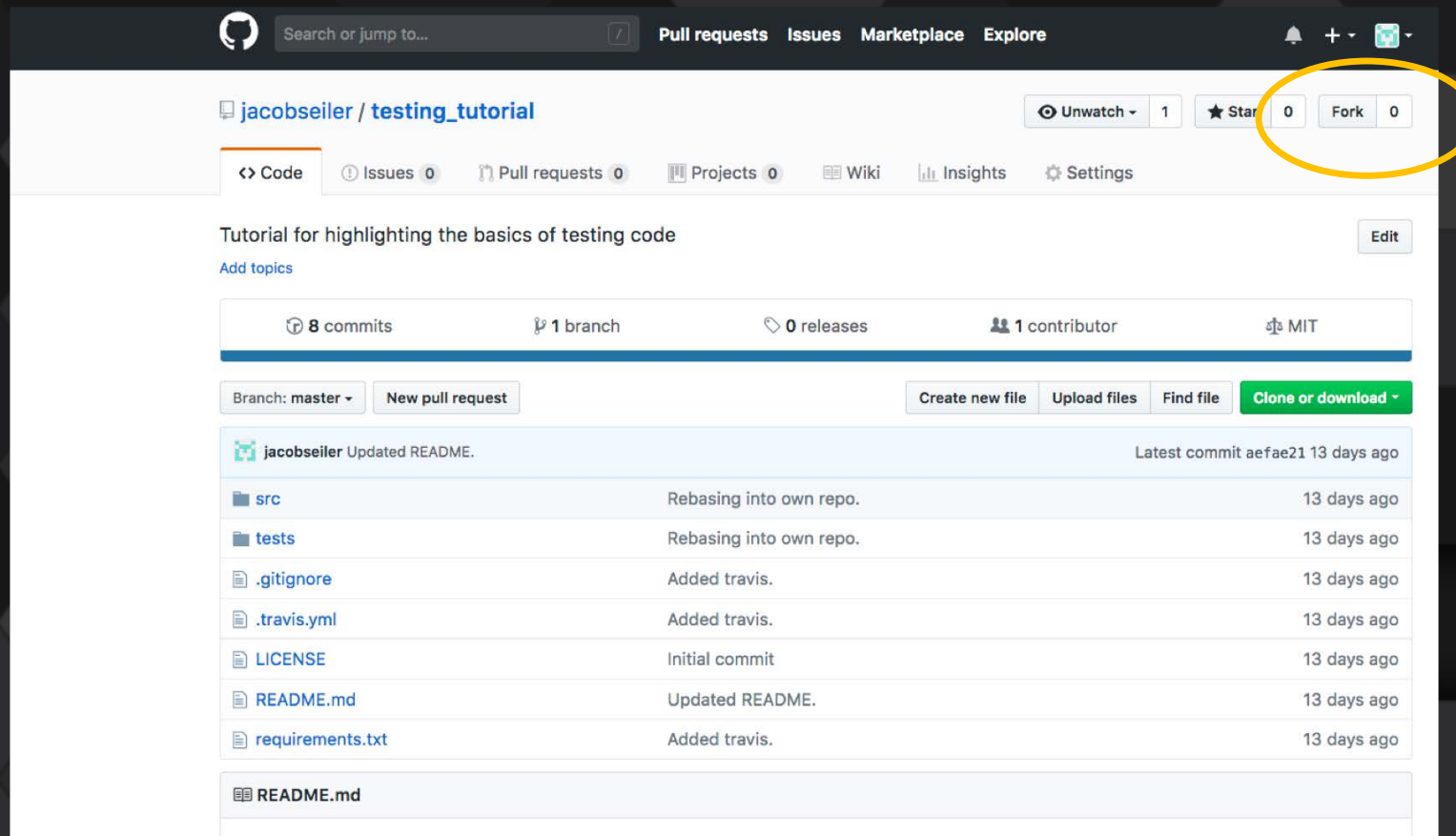
- Have you ever woken up in a cold sweat at the thought that all your results may be wrong because your code isn't actually running correctly and it's only through "sheer dumb luck" you've succeeded so far?
- Well you will now.
- Testing gives you confidence that your code is running exactly how you want it after EVERY single commit.

Testing Paradigm



Setting Up your Repo

- Create your own fork of https://github.com/jacobseiler/testing_tutorial



The screenshot shows the GitHub repository page for `jacobseiler / testing_tutorial`. The repository description is "Tutorial for highlighting the basics of testing code". The repository statistics show 8 commits, 1 branch, 0 releases, 1 contributor, and the MIT license. The repository is currently on the `master` branch. The repository contains the following files and folders:

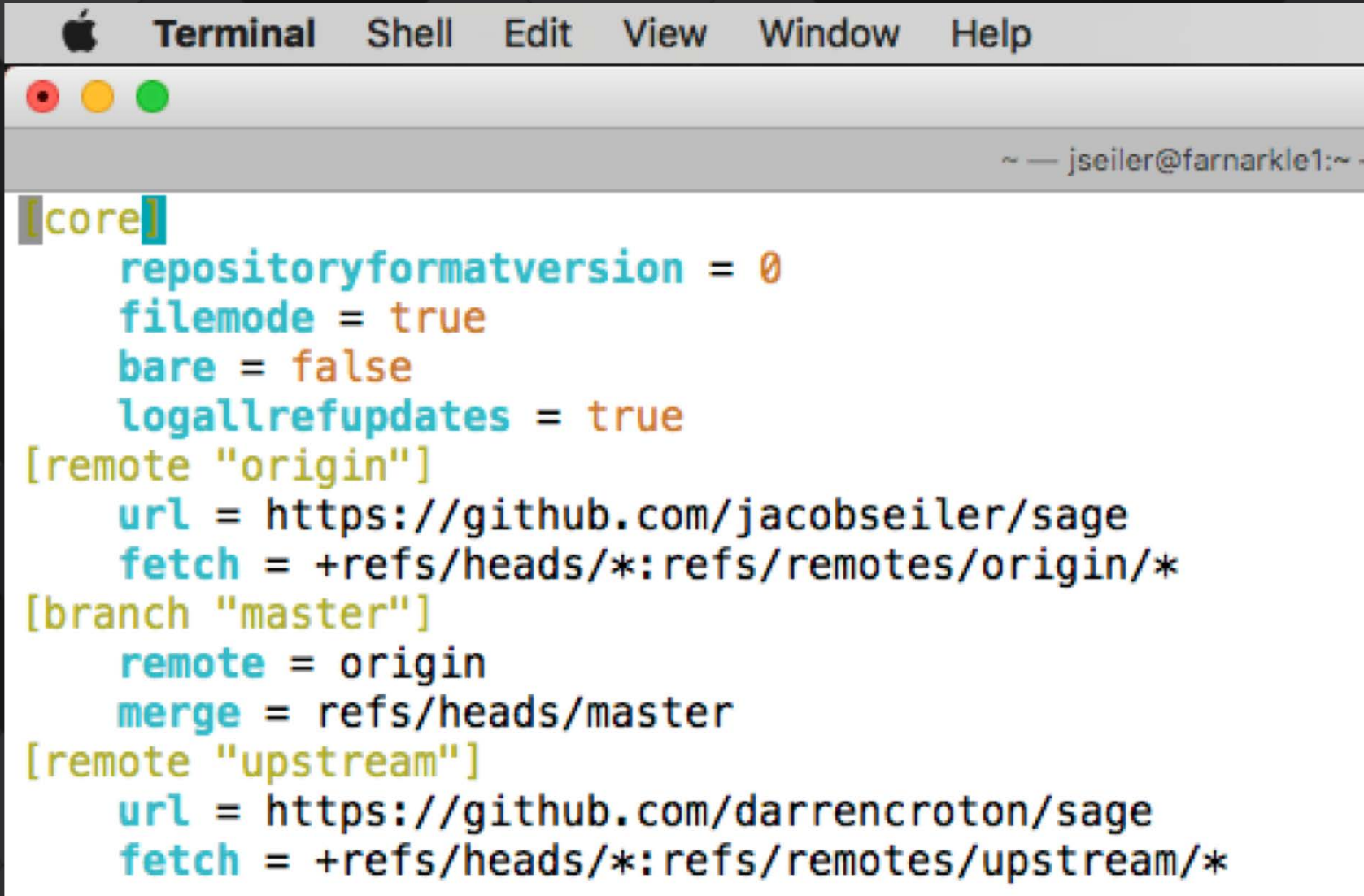
File/Folder	Description	Latest commit
<code>src</code>	Rebasing into own repo.	13 days ago
<code>tests</code>	Rebasing into own repo.	13 days ago
<code>.gitignore</code>	Added travis.	13 days ago
<code>.travis.yml</code>	Added travis.	13 days ago
<code>LICENSE</code>	Initial commit	13 days ago
<code>README.md</code>	Updated README.	13 days ago
<code>requirements.txt</code>	Added travis.	13 days ago

The repository also has a `README.md` file. The repository is currently on the `master` branch. The repository is currently on the `master` branch.

Setting Up your Repo

- Clone your fork.
- Set my repo as the upstream repo of your clone.
 - Open `‘.git/config’`
 - Copy paste the “[remote “origin”]; url = ...; fetch = ...”
 - Replace “origin” with upstream.
 - Replace my Github URL with your Github URL.

Setting Up your Repo

A screenshot of a macOS Terminal window. The title bar shows the Apple logo, the word "Terminal", and menu items "Shell", "Edit", "View", "Window", and "Help". The window has three colored window control buttons (red, yellow, green) on the left. The status bar at the bottom right shows the user "jseiler" and the host "farnarkle1". The terminal content shows the configuration of a git repository using the [core], [remote "origin"], [branch "master"], and [remote "upstream"] sections.

```
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
[remote "origin"]
  url = https://github.com/jacobseiler/sage
  fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
  remote = origin
  merge = refs/heads/master
[remote "upstream"]
  url = https://github.com/darrencroton/sage
  fetch = +refs/heads/*:refs/remotes/upstream/*
```


The Code

- Given a grid with a large size (e.g., $256 \times 256 \times 256$), we wish to downsample it to a smaller size (e.g., $64 \times 64 \times 64$). However we want to ensure that averages are kept during the process.
- Using a sliding cube with side $(256/64) = 4$, we use scipy to generate a sliding sum across the input grid.
- Then we only use every 4th cell and normalize it's value by $4^3 = 64$.

The Tests

- There are three tests contained in 'tests.py'.
 - For an input grid containing a homogenous input, the output grid should be homogenous of the same value.
 - For an input grid containing input every $(256/64)$ th cell with value $(256/64)^3$, the output grid should contain all values of 1.
 - For an input grid with random values generated from a known seed, the output grid should **exactly** match the 'correct grid' saved in the repo.
- Check that these all pass by invoking 'python tests.py'

The Automation

- We've now ensured that our code is working as intended. Any time we now make changes to our code we should ensure that the tests pass; if they don't, then we know we've broken something critical.
- This can be cumbersome hence we want to automate the process.
- This process is called **Continuous Integration (CI)** whereby we set up an automated pipeline to execute our tests for any commit we make.

Travis

- Travis is a CI service that we will use for the tutorial.
- <https://docs.travis-ci.com/user/getting-started/> Contains most of the information required to get your repo set up and ready for automated testing.

The .travis.yml File

- The .travis.yml file contains instructions to tell Travis exactly what to do once it has cloned your repo.
- There is an example .travis.yml file in the repo called 'travis_template.yml'. However it's missing a few key pieces of information!
 - Who should it email and when?
 - What is the Python versions of the builds you're testing?
 - What is the script to be executed after everything is installed?
- Finally, you will also need to generate a 'requirements.txt' file. This what Pip will install from to gather all your packages.

Making Pull Requests

- One of the most powerful aspects of CI is the ability to ensure any Pull Requests do not break the code.
- When a Pull Request is issued, Travis ensures that the merging branch and the resulting merged branch pass the tests outlined by the `.travis.yml` file.
- Make a change to the repo and issue a PR to the upstream repo. Watch before your very eyes as Travis works its magic!

Wrapping Up

- Testing is a critical step of writing code that is often overlooked. Hours upon hours of work and frustration can be saved by implementing robust tests.
- By having your tests automated, you can always be certain that your code is executing correctly after EVERY commit.
- CI also simplifies collaborative code. There's no need to have endless pull request discussing about "have you checked this works?".