Quality Assurance

Risk

Product

Tools

Process

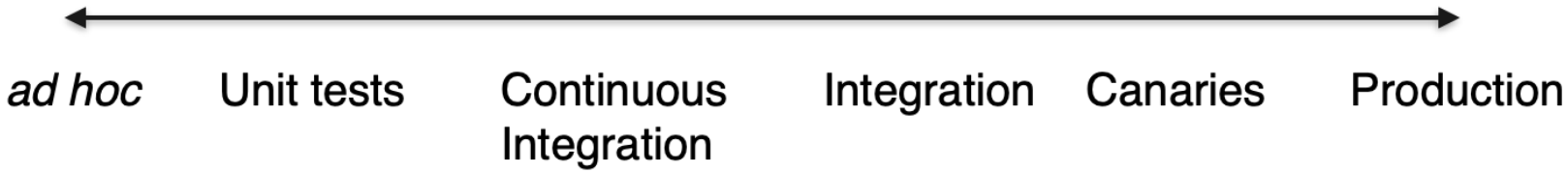
How do you remove risk?

Testing

Testing Myths

- “No tests” arguments
  - I’m a good programmer, I don’t need to prove myself.
  - I already tested this manually!
- Reality:
  - Tests aren’t about now, they are about from now on.
  - You need a fast, consistent, reliable signal for whether later changes have broken/disrupted your current feature.

Ways to Test


ad hoc
Unit tests
Continuous Integration
Integration
Canaries
Production

Lifetime of Code

Production Testing

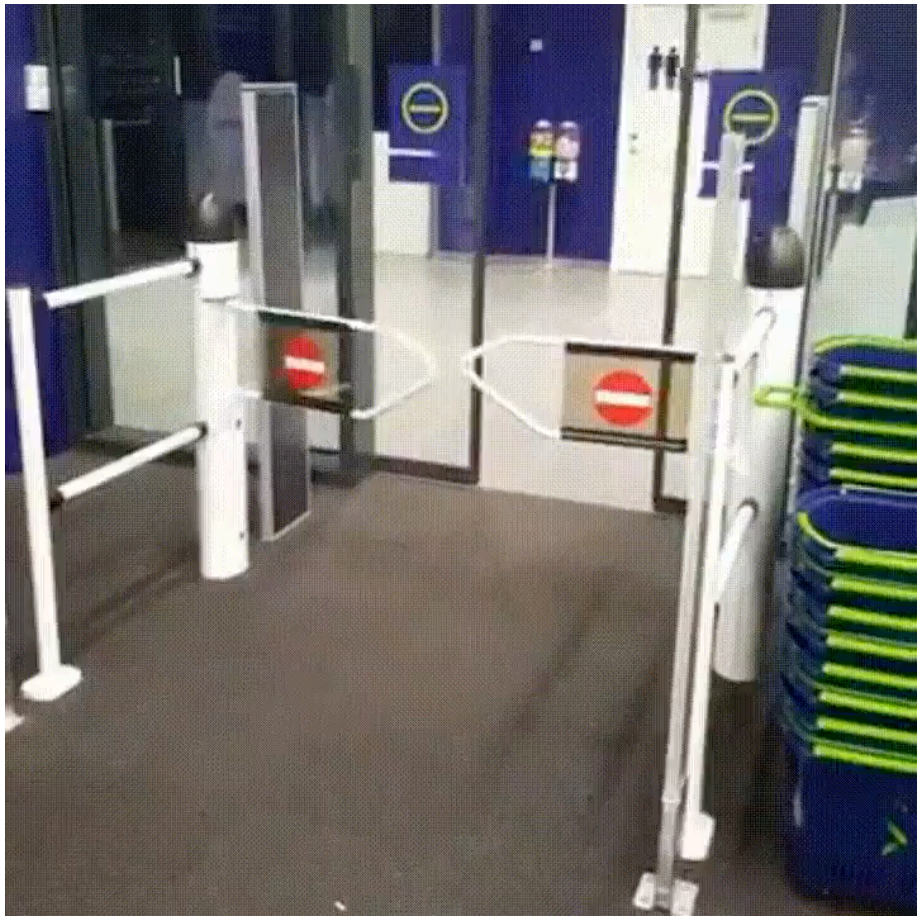
- Most faithfully captures real-world scenarios: it is the real world!
- Relies upon users to detect and report problems.
- Expensive to fix bugs found in production.

Canaries

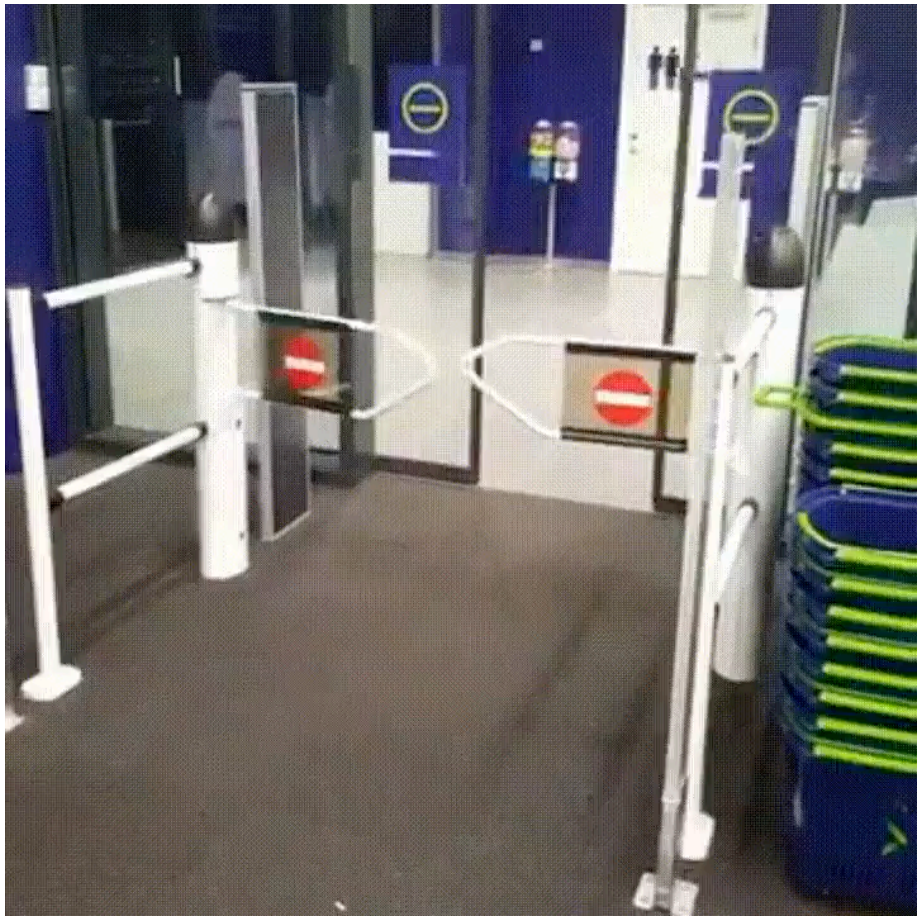
Canaries

- “Staged rollout”
- Still serves production traffic
  - But only a subset
- Sometimes more heavily instrumented
  - Lower performance, but better information about when something goes wrong.
- Still expensive to fix
  - But the users haven’t seen it yet.

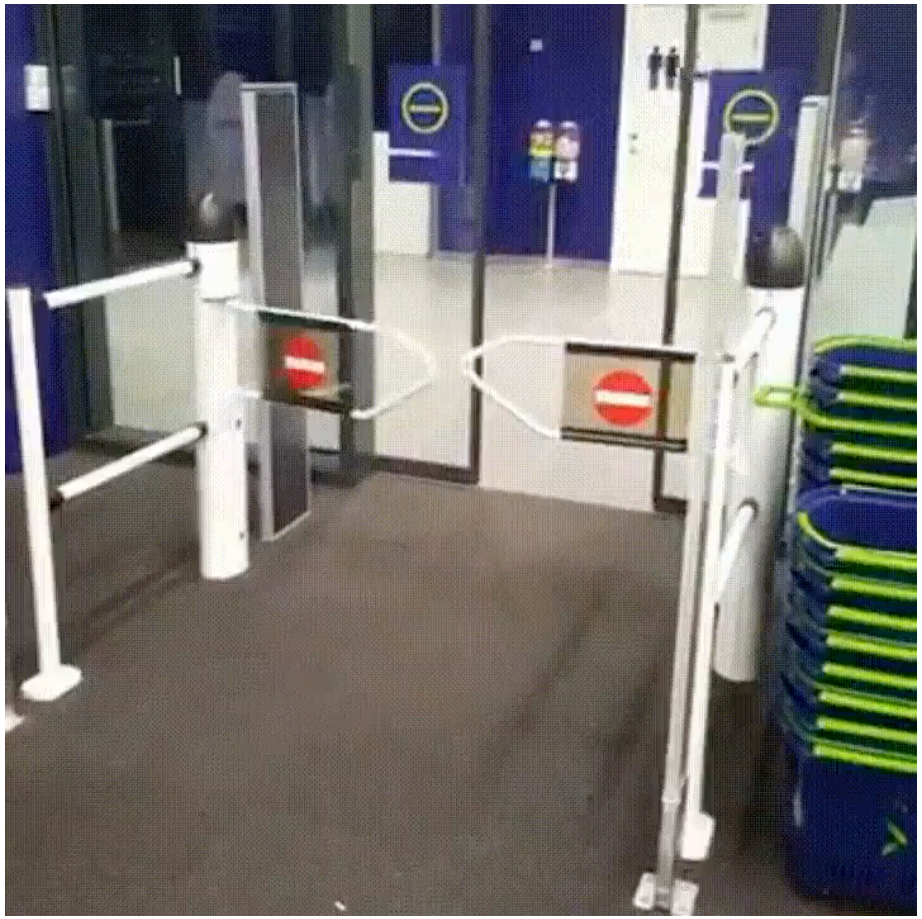
Integration Tests



Integration Tests



Integration Tests



Integration Tests

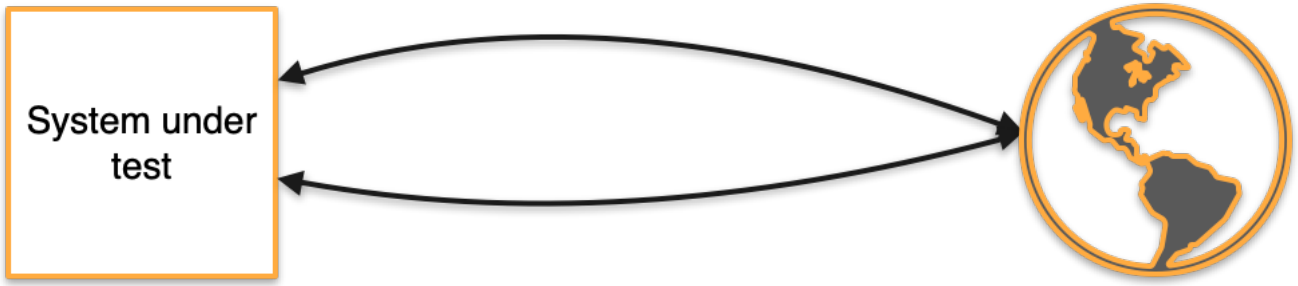
- Tests the combination of smaller components into large ones.
- Individual components can work correctly but fail when put together.
- Still expensive to fix failures detected here
  - Usually involves fixing interfaces between multiple components


Unit tests: Fail
Dolores
RESTAURANT
Integration test: Pass

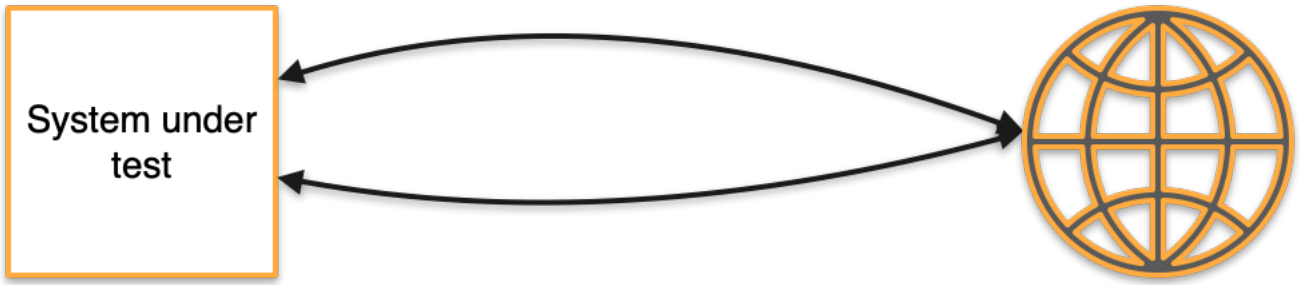
Unit Tests

- Designed to test a small unit of functionality
- Runs very quickly
  - An integral part of the edit-compile-test loop
- Small and self-contained: doesn't require interaction with external systems

Fakes and Mocks



Fakes and Mocks



ad hoc testing

- “Kicking the tires” basic testing
- Easy to setup, not guaranteed to catch much

If you find yourself running the same tests manually multiple times, it’s good sign that it’s time to invest in better testing infrastructure.

Continuous Integration

- Submitting pull requests early and often, and running tests over the resulting branch.
- Integrates directly with the version control system to run existing tests.
- Catches failures earlier, but fidelity is not a good as production.

- Presubmit vs. Postsubmit

Continuous Integration

“Continuous Integration doesn’t get rid of bugs, but it does make them dramatically easier to find and remove.”

— Martin Fowler, Chief Scientist, ThoughtWorks

Sample CI Workflow (Github + Travis)


Create Pull Request


GitHub tells Travis CI build is mergeable


It builds and passes tests

Travis updates PR


PR is merged

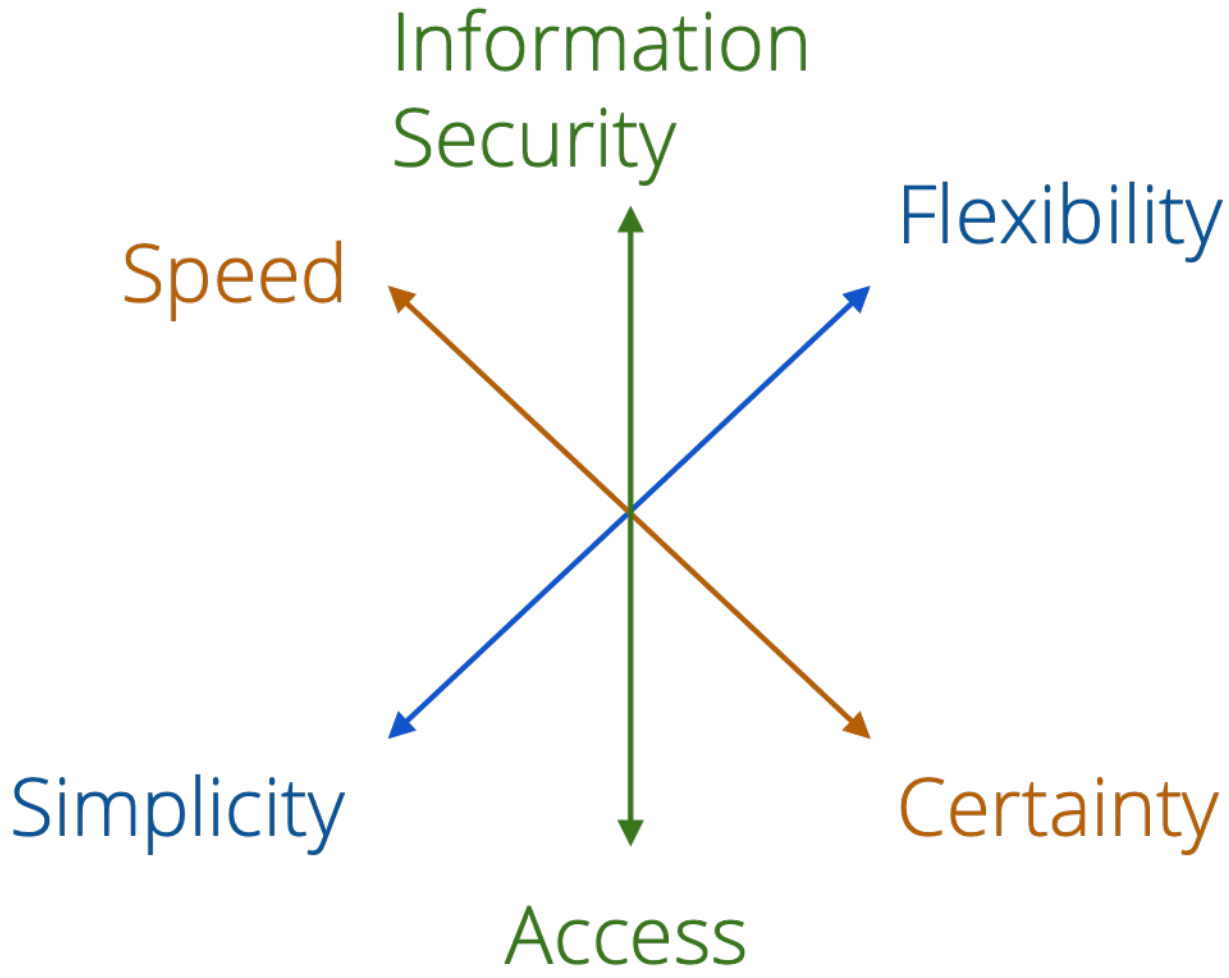
Continuous Integration

Travis CI

circleci

GitHub Actions

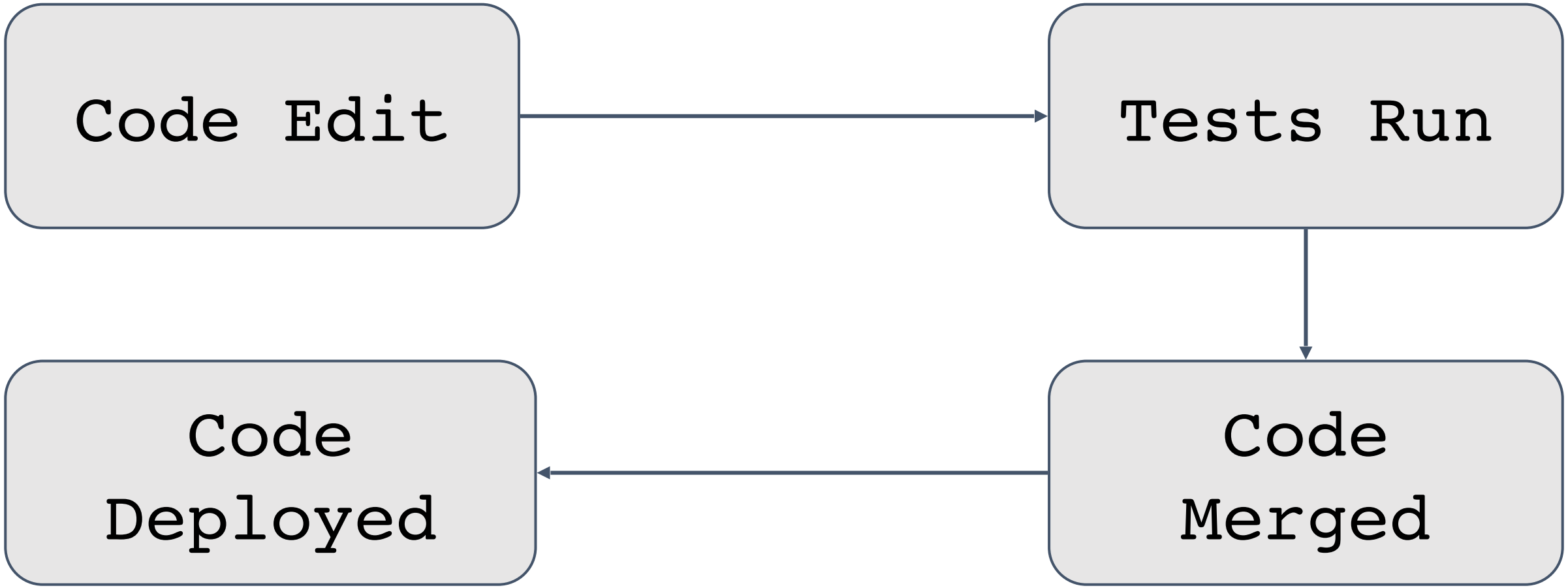
Conflicts:


Information Security
Flexibility
Speed
Simplicity
Certainty
Access

Most of the benefits of CI come from running tests

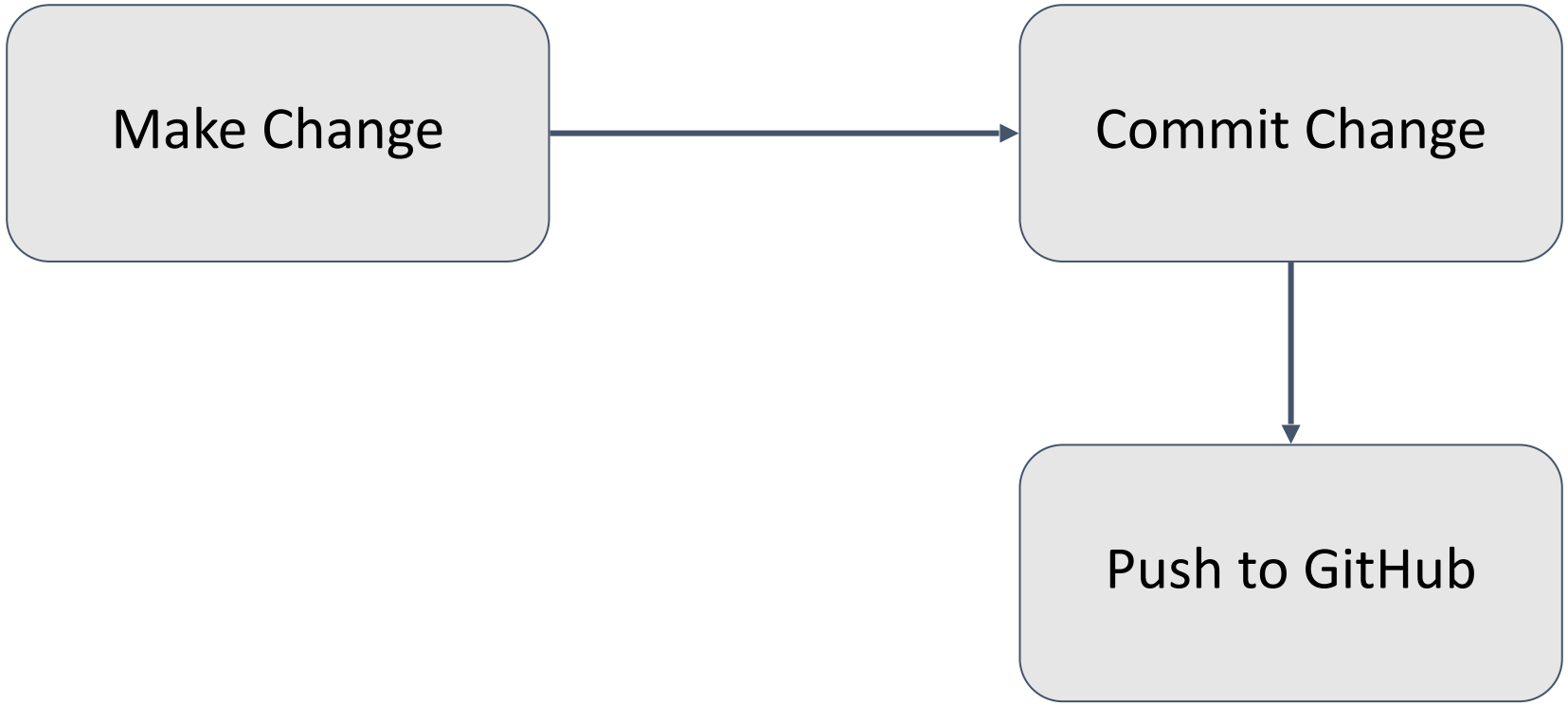
Observation

CI/CD Pipeline overview

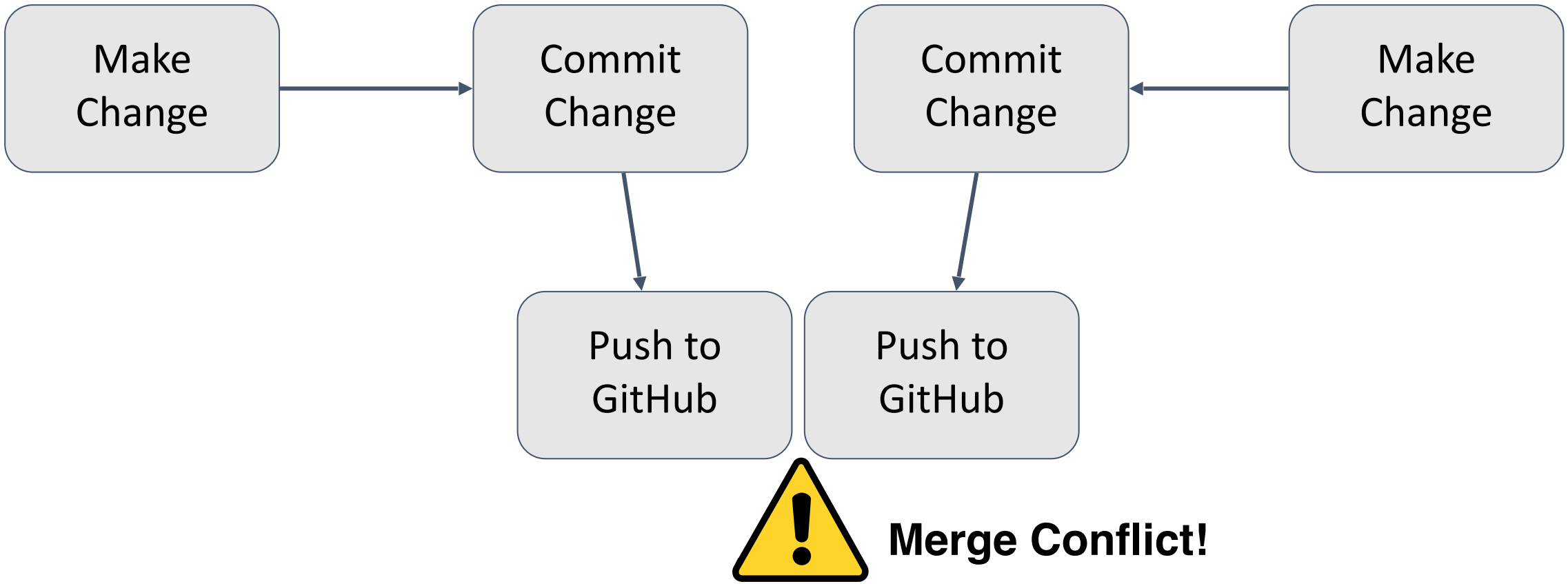


Github Flow

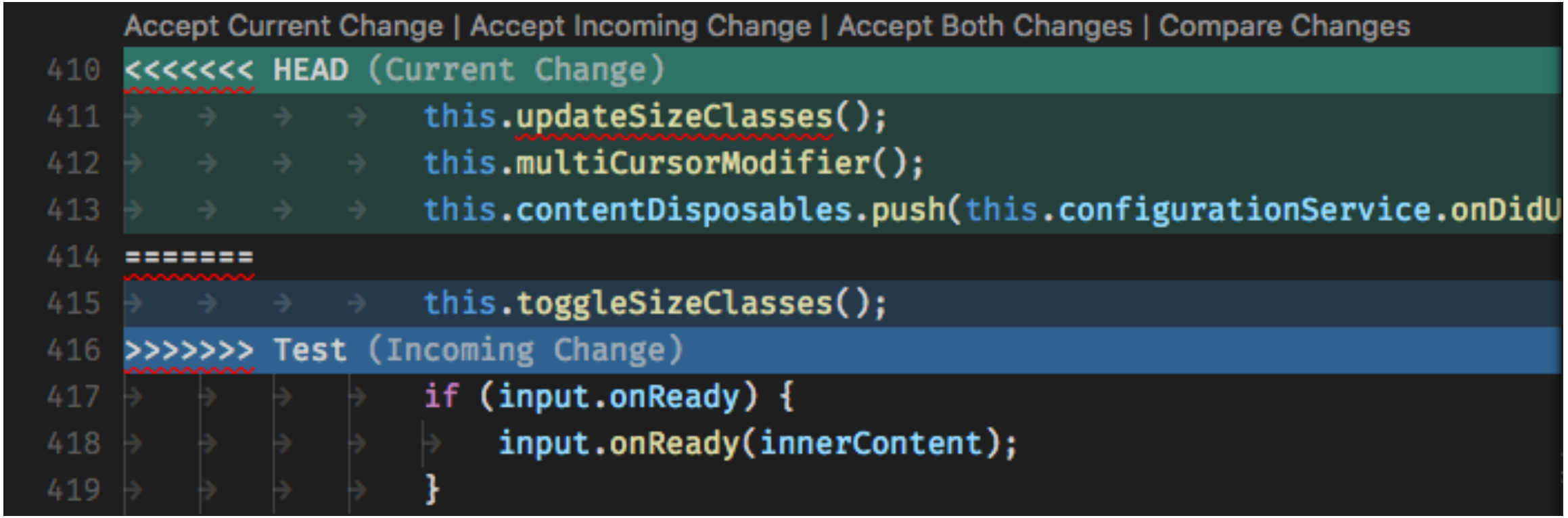
Simple workflow works fine for a single dev



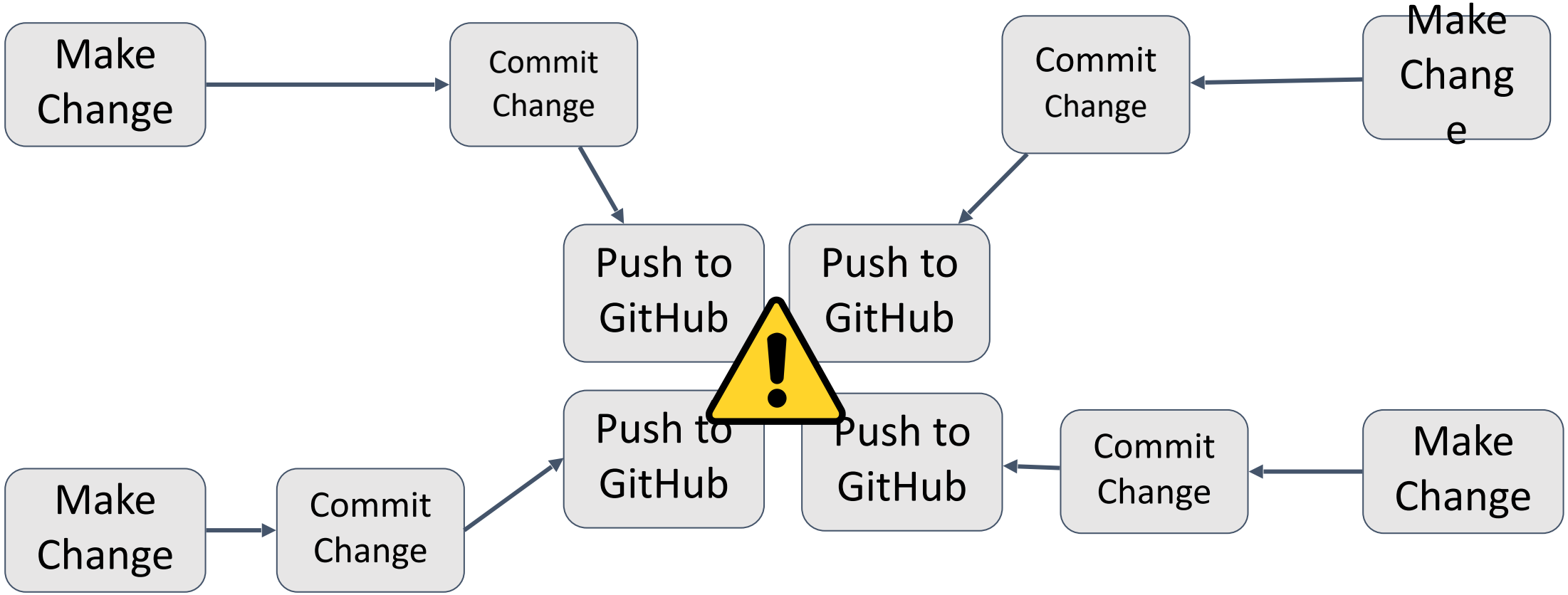
Quickly breaks down with multiple users



Merge Conflict (plus VS Code)



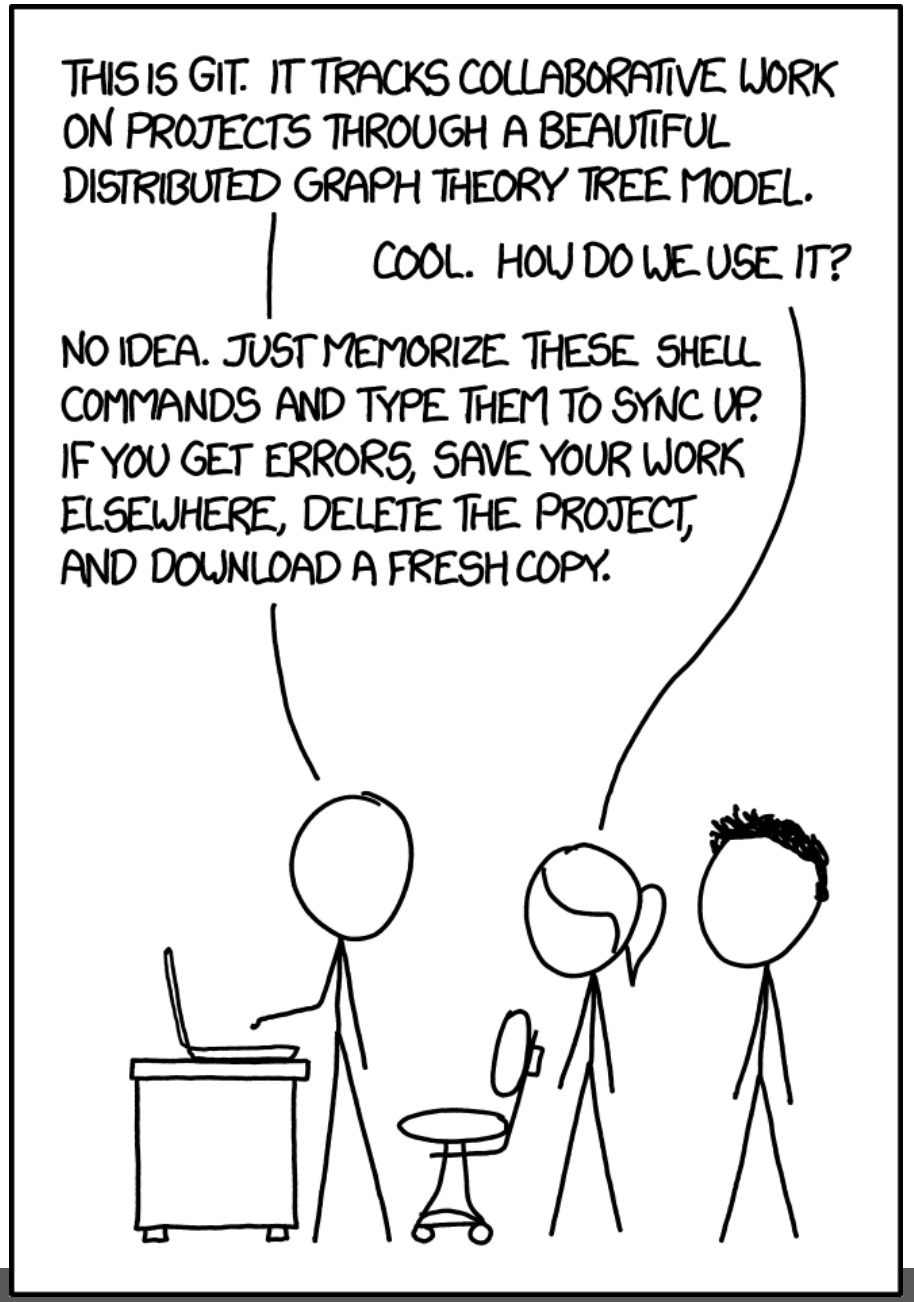
Quickly breaks down with multiple users



GitHub Flow

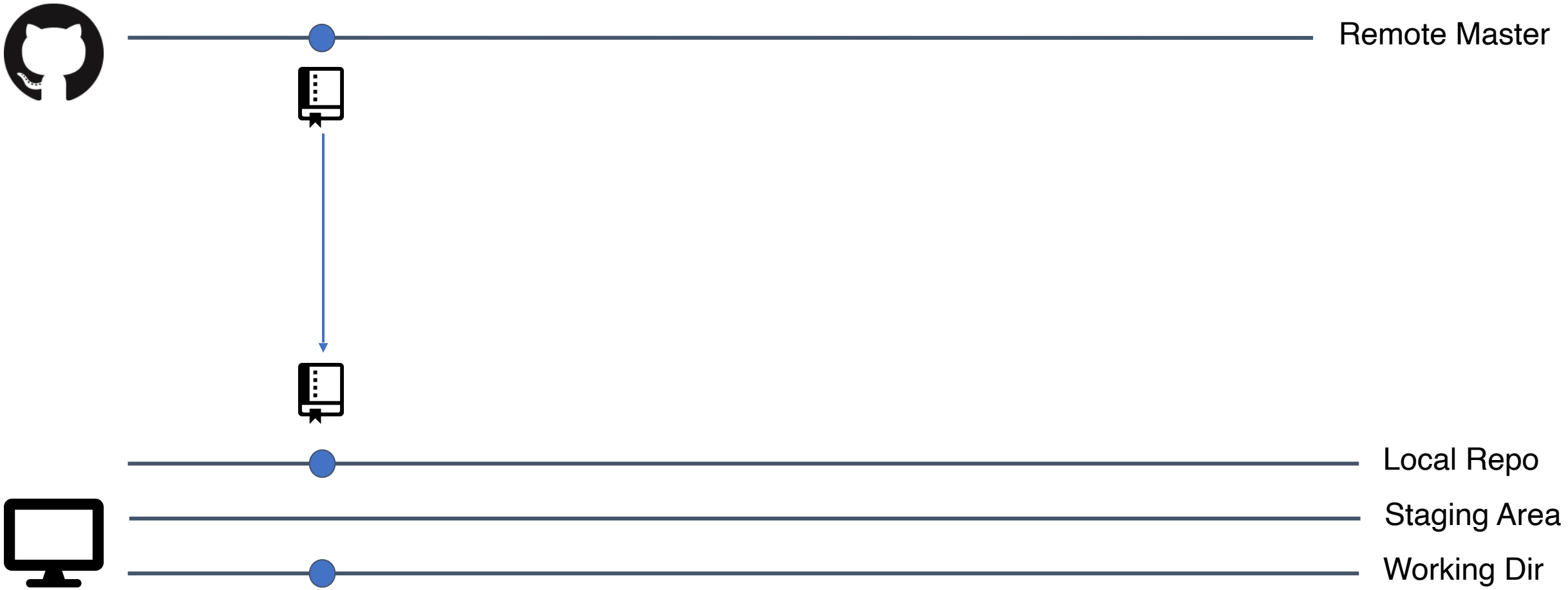
or: branch based development

Git Fundamentals

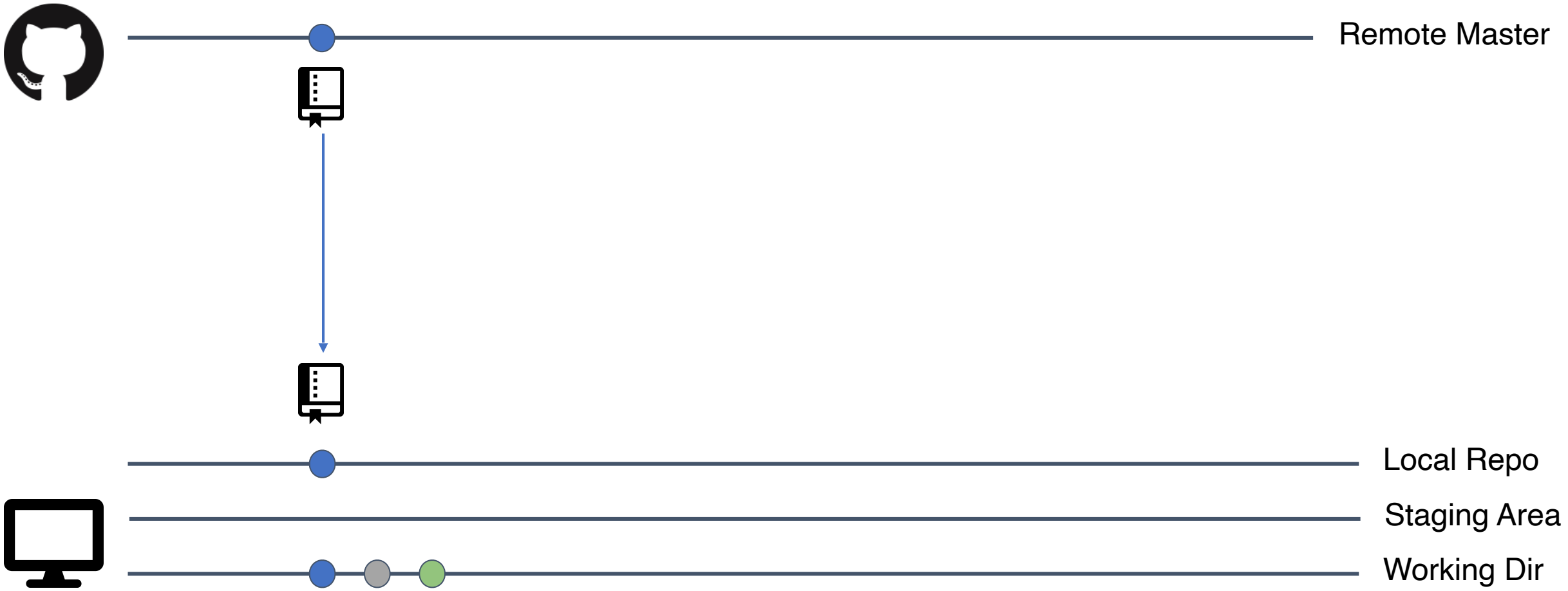


> git clone REPO

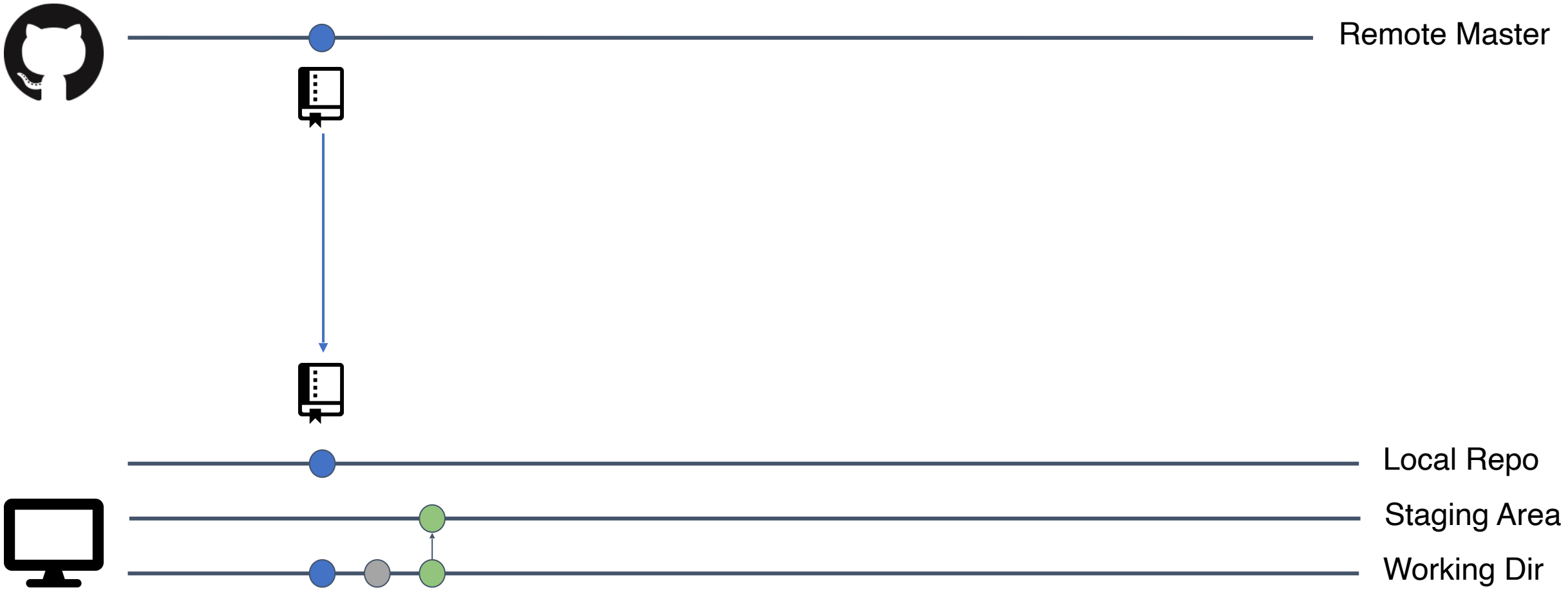
> git clone REPO



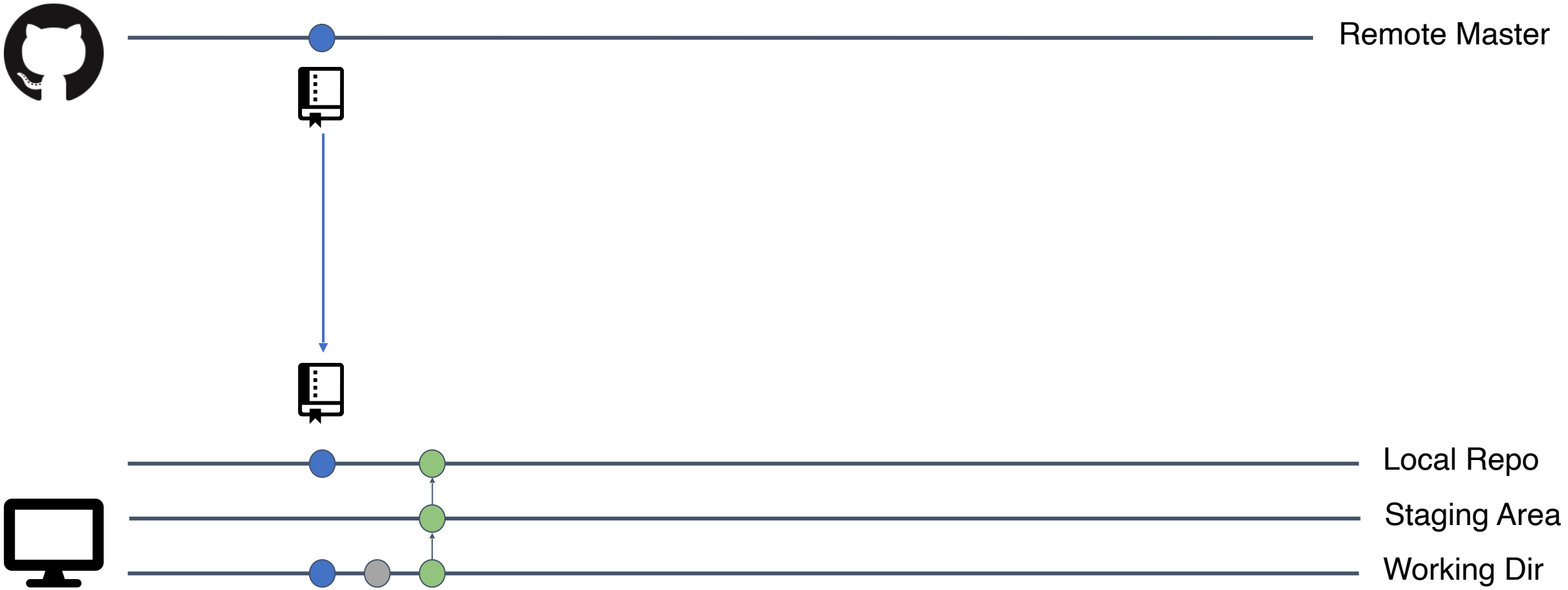
Edit file in working directory



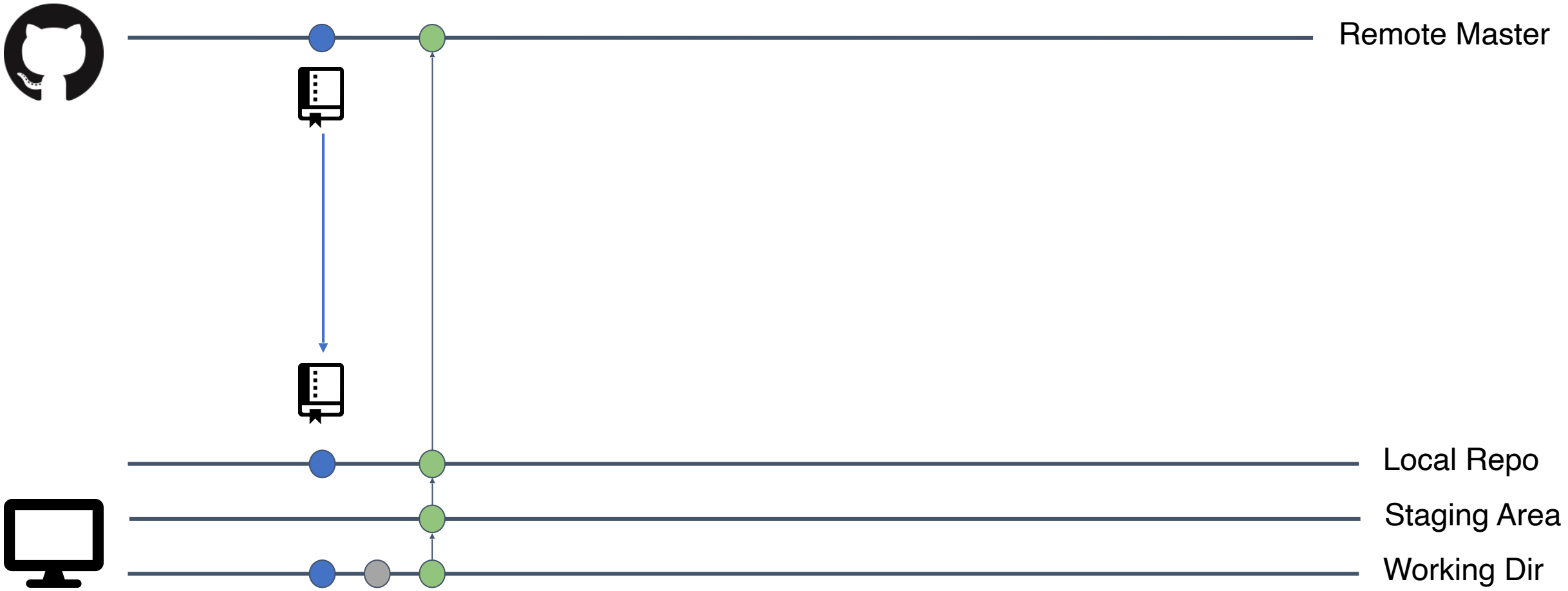
> git add File



> git commit -m “added green edit”



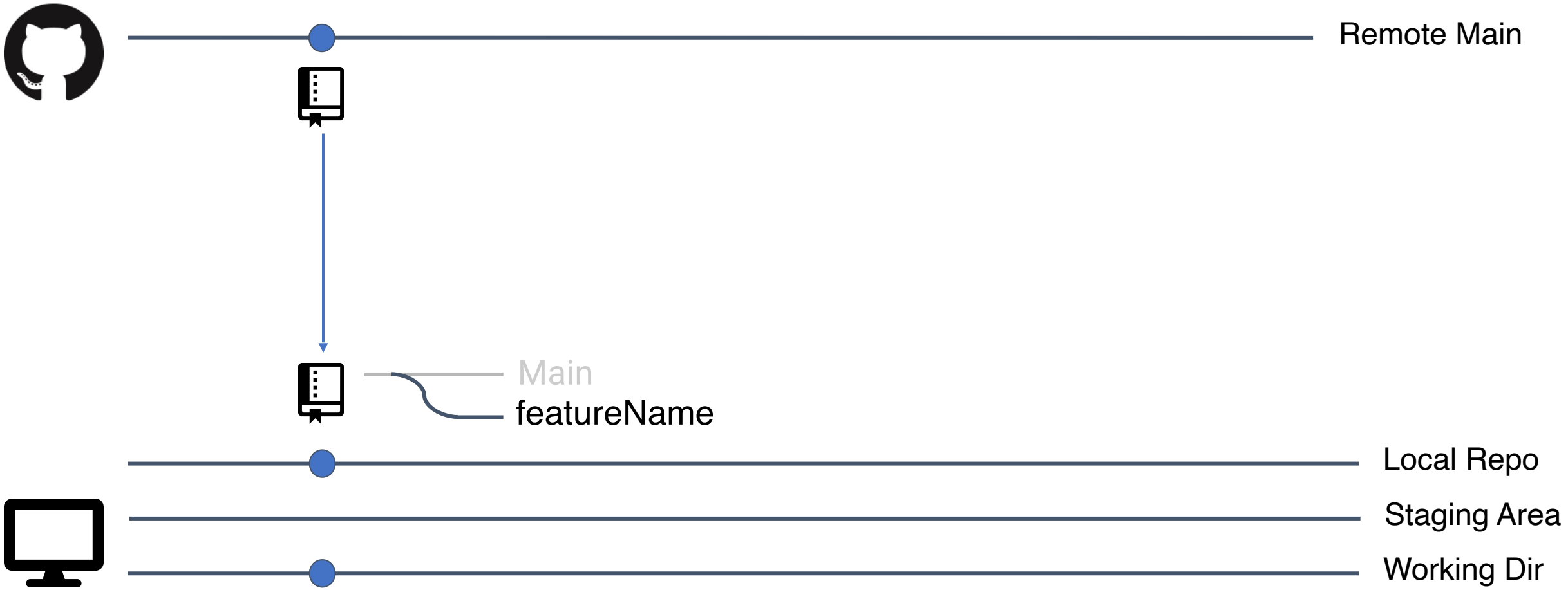
> git push origin master



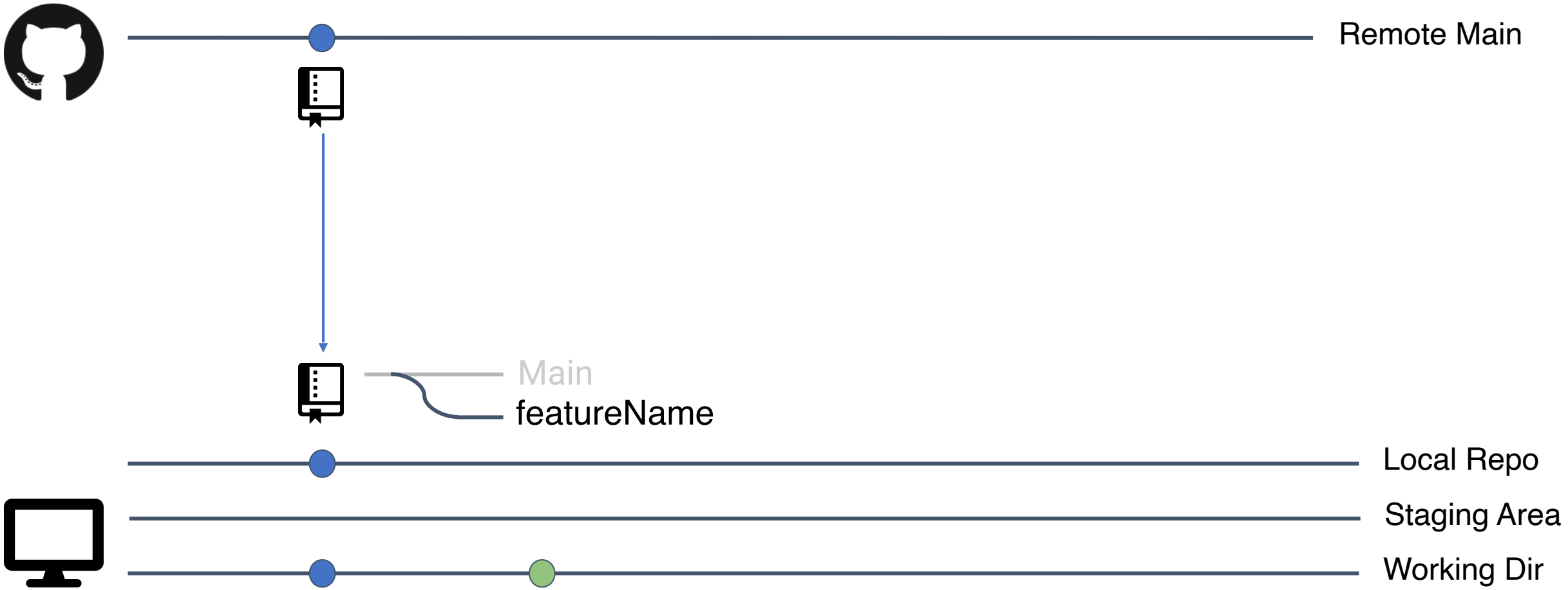
> git push origin master

Branch Based Development

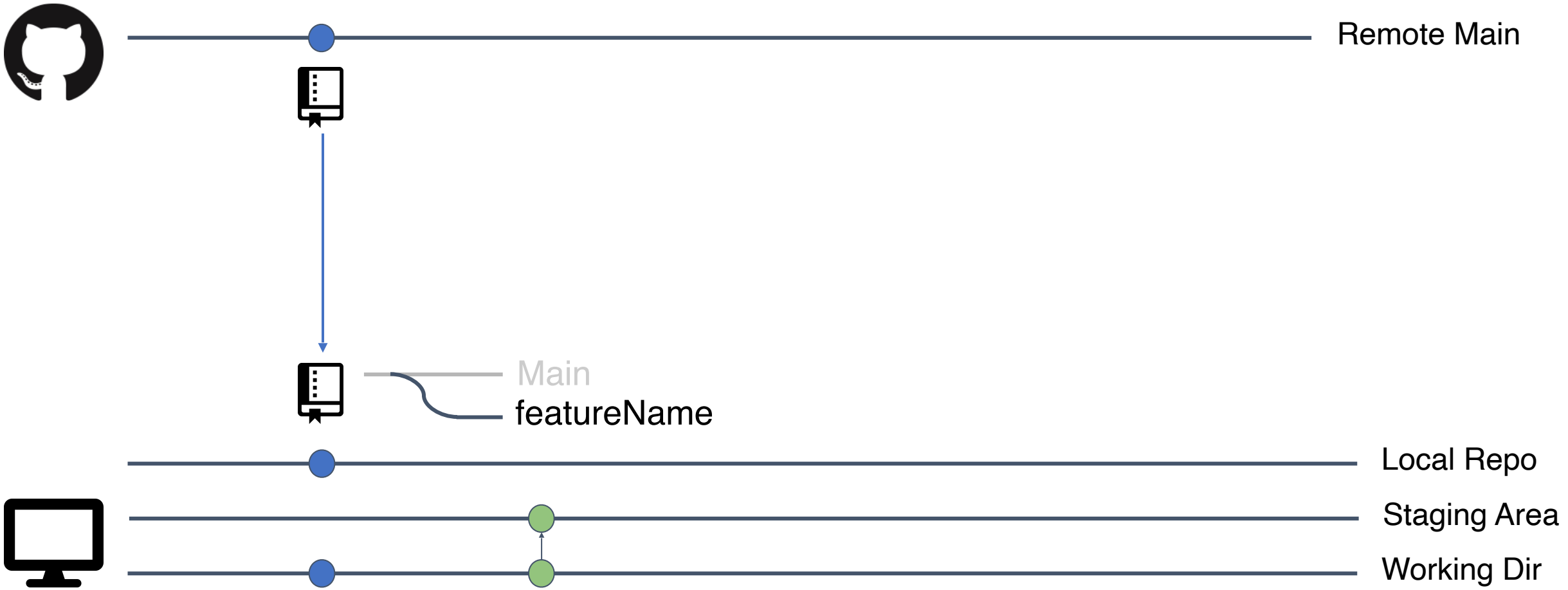
> git checkout -b featureName



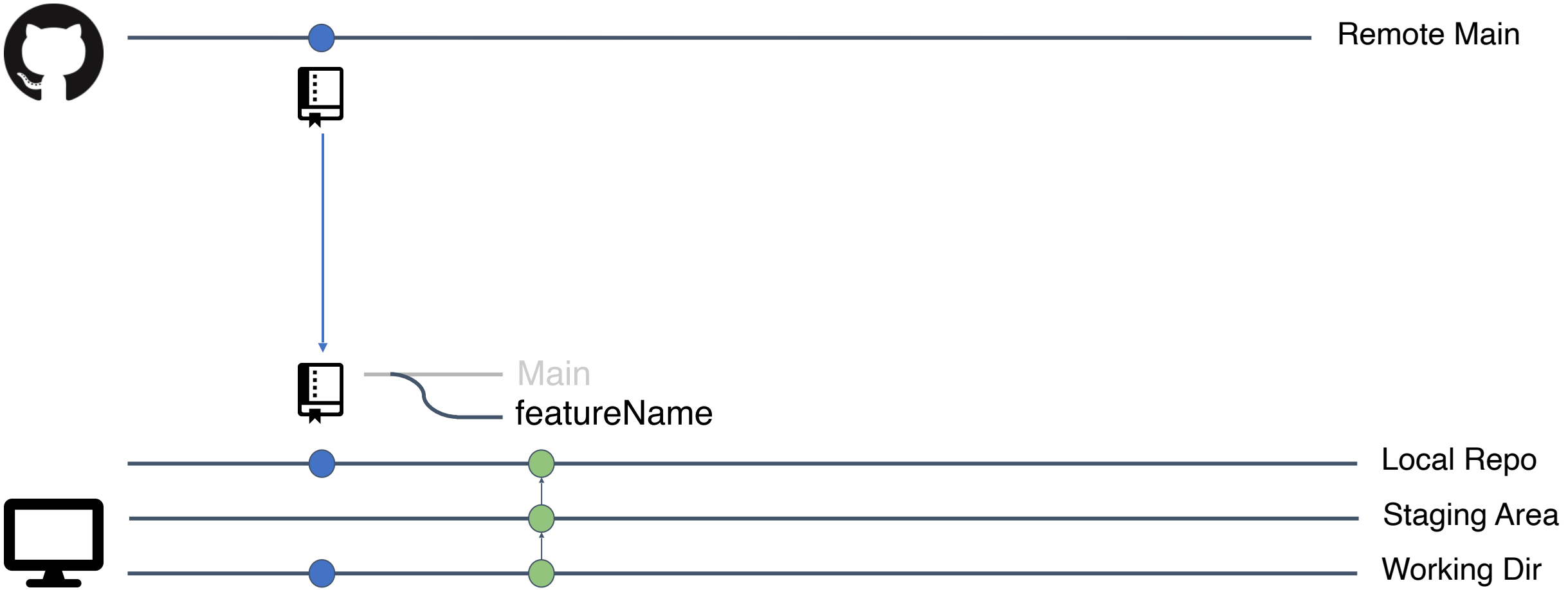
edit file(s) in working directory



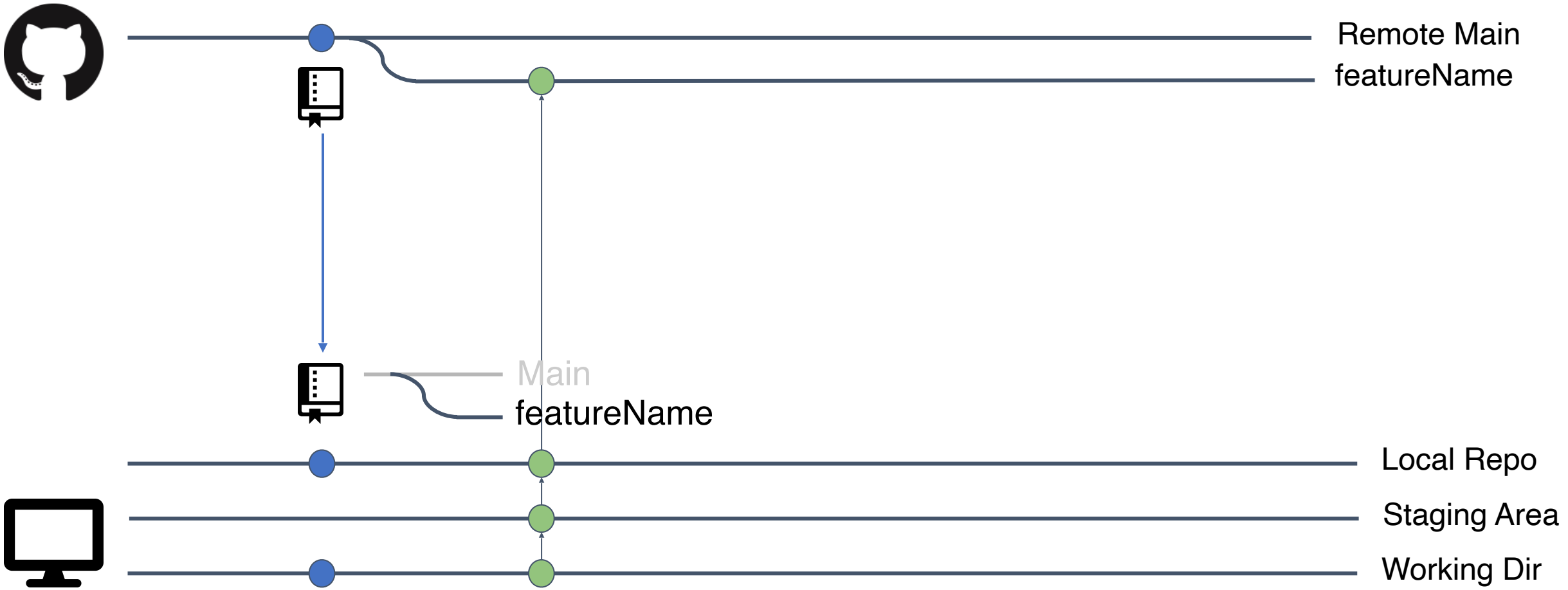
> git add File


Remote Main
Main
featureName
Local Repo
Staging Area
Working Dir

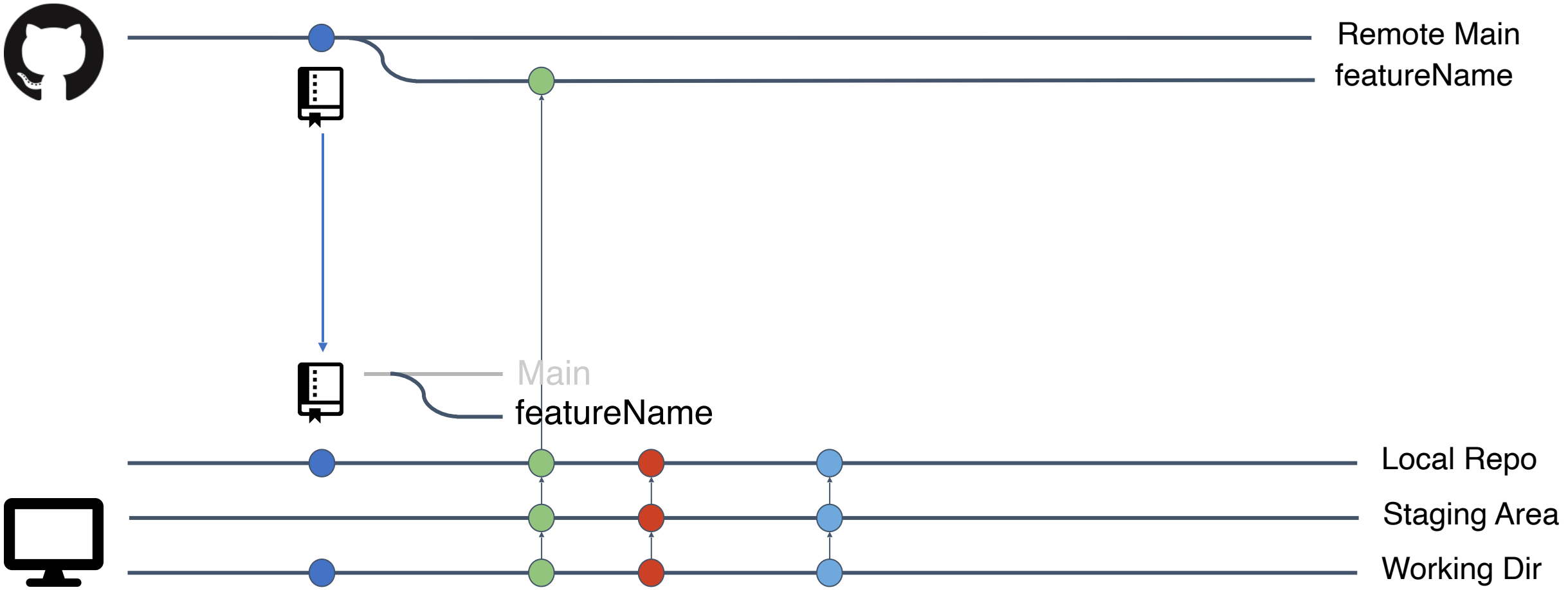
> git commit -m “added green edit”



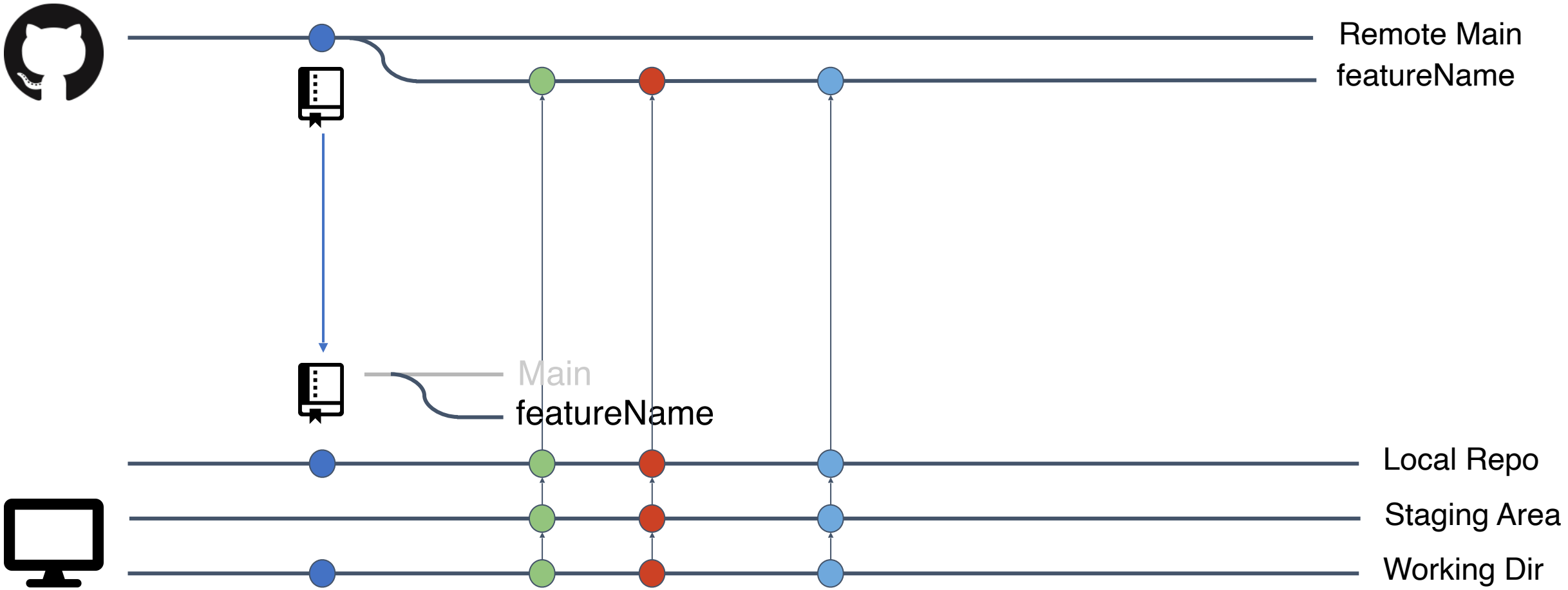
> git push origin featureName



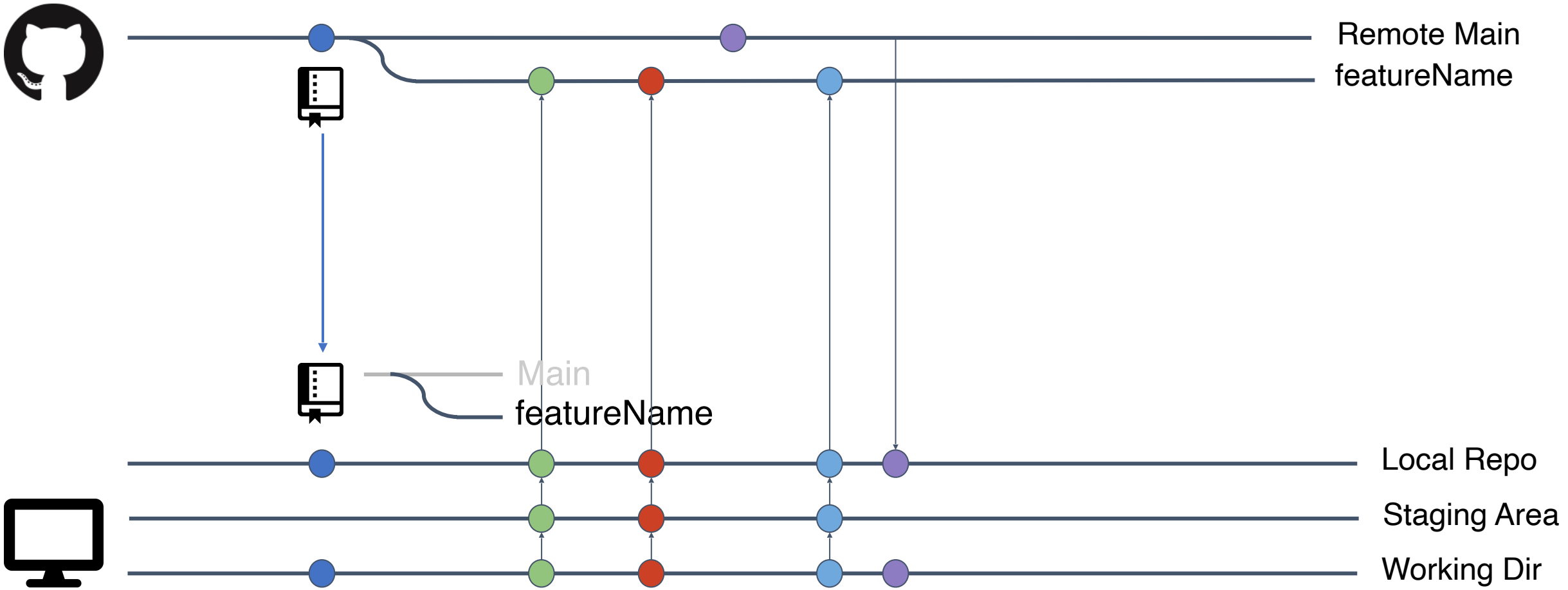
Continue work



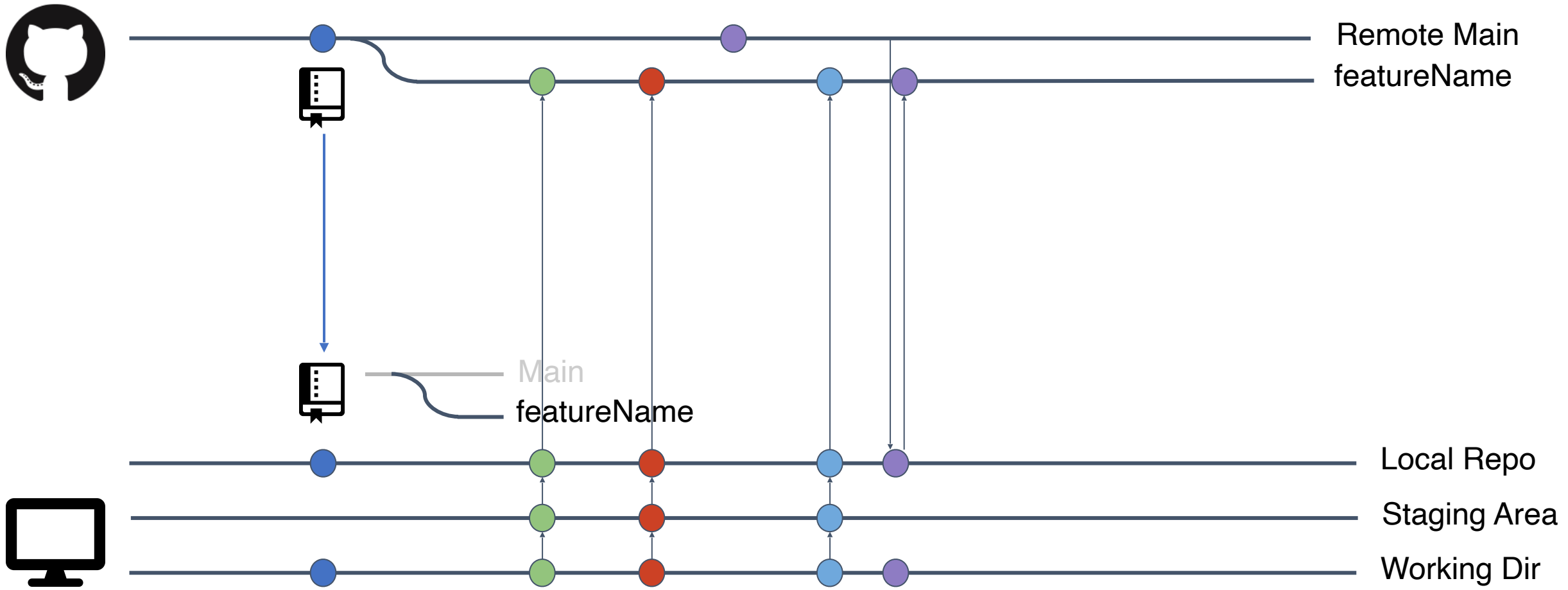
> git push remote featureName



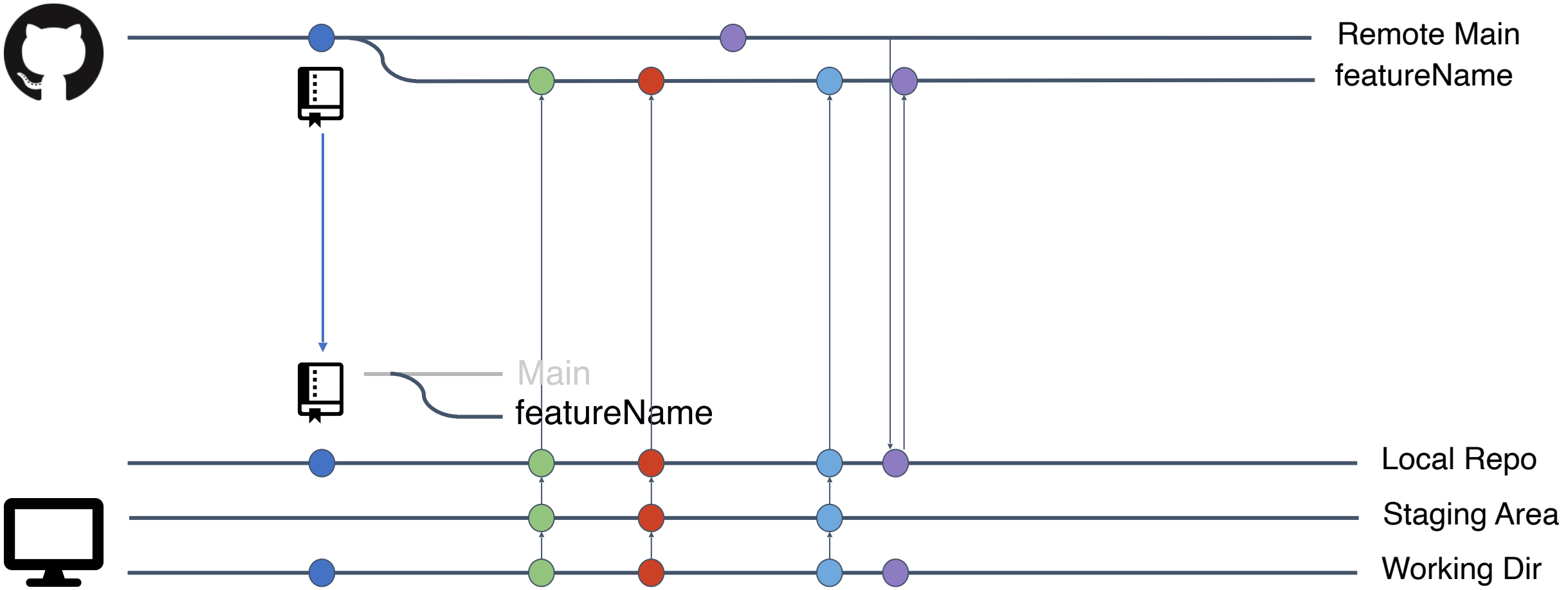
> git pull origin master //Resolve any conflicts



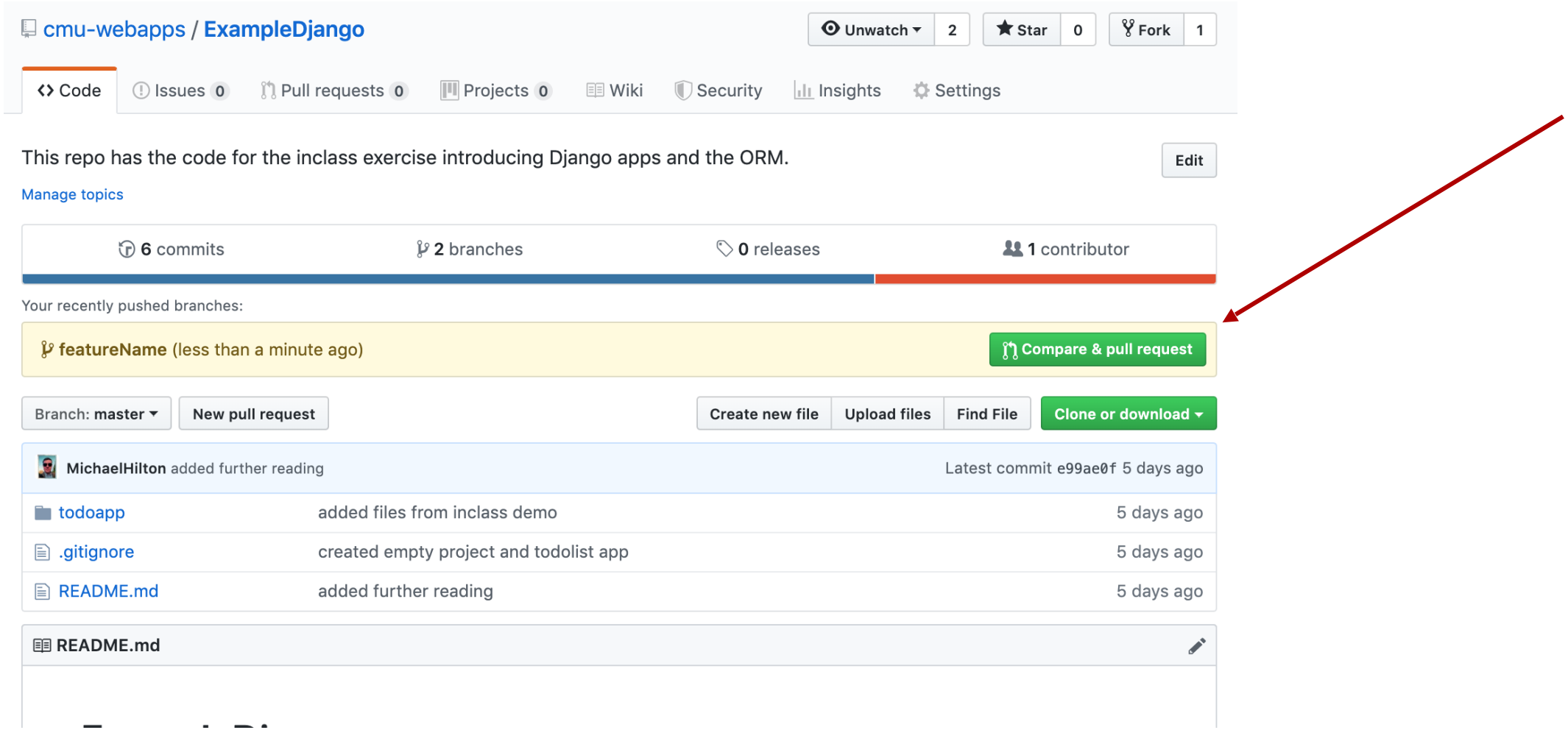
> git push featureName



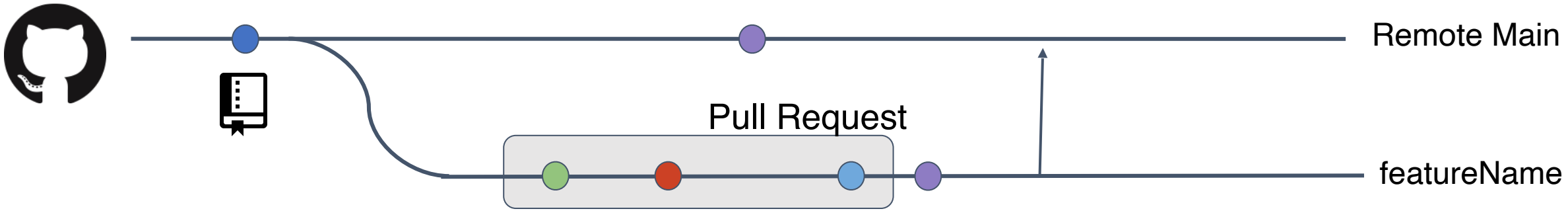
Open Pull Request on GitHub



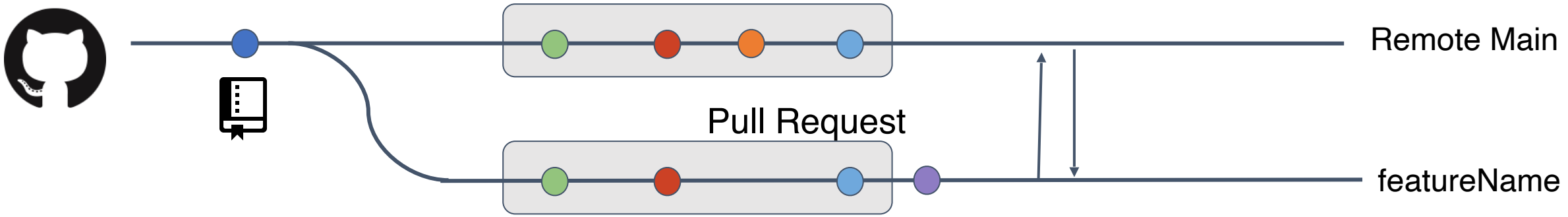
Open Pull Request on GitHub



Pull Request Created



Pull Request Accepted



Test Driven Development (TDD)

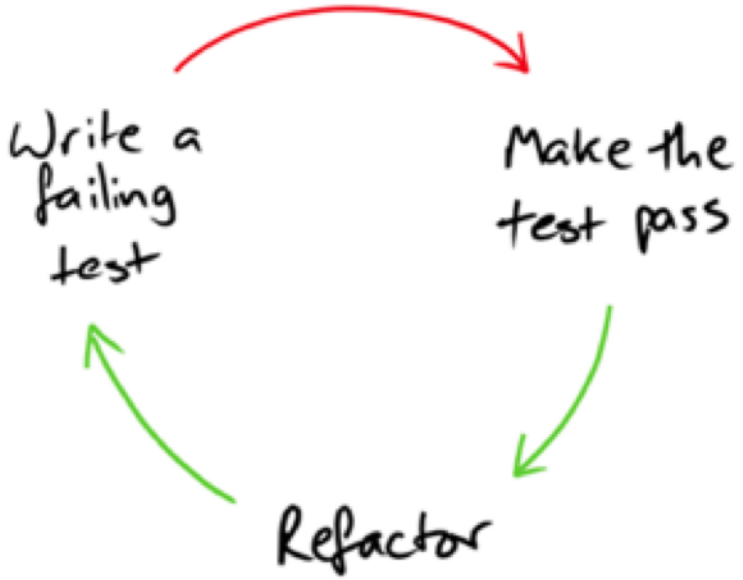
Three Simple Rules

1) You are not allowed to write any production code unless it is to make a failing unit test pass.

2) You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

3) You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

Test Driven Development (TDD)



From Growing Object-Oriented Software by Nat Pryce and Steve Freeman
http://www.growing-object-oriented-software.com/figures.html

@sebrose

http://cucumber.io

Why TDD

“The act of writing a unit test is more an act of **design** than of verification.

It is also more an act of **documentation** than of verification.

The act of writing a unit test closes a remarkable number of feedback loops, the least of which is the one pertaining to **verification** of function”.

Advantages of TDD

Clear place to start

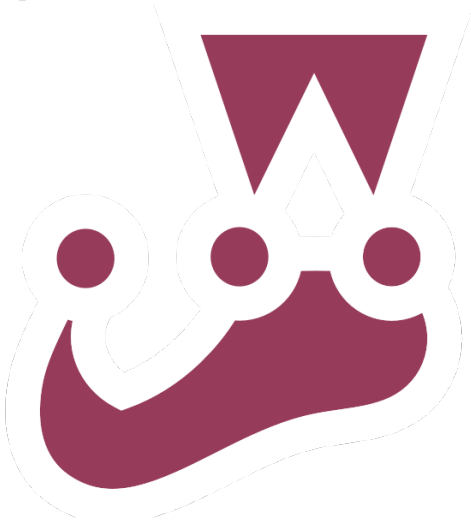
Much less code thrown away, less wasted effort

Less Fear

Side Effect: Robust test suite

Testing Frameworks

- Designed to make writing and executing tests easier
- Often integrate with other systems, such as continuous integration
- Takes care of much of the boilerplate of testing



Quality Assurance

- Take risk on the product, not the process or tooling
- Build testing into your process early, and use it throughout