

Software Design for Data Science

Continuous Integration

*Naomi Alterman
University of Washington
February 17, 2026*



A common problem

- I change some code
- I make a PR
- Riyosha approves my PR
- I merge my PR

- Whoops! I forgot to run the tests and now all the code in main is broken

A solution

eg install packages

Tests (eg unit tests)
Style checkers (eg pylint)

Automated **building** and **testing** of a code base

aka **Continuous Integration**

“CI”

Building and testing is
done “continuously” as
the code is developed

“Integrating” changes
from developers into
code

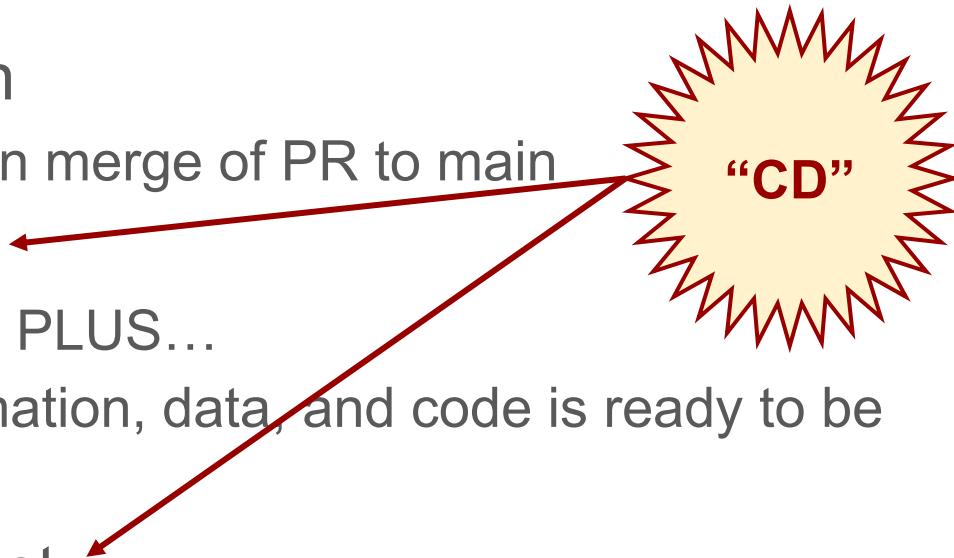
Actually...

Commit is pushed => Test/Build

So more like “semi continuous”

Going a step further...

- Continuous Integration
 - Automated build/test on merge of PR to main
- Continuous Delivery
 - Continuous integration PLUS...
 - All configuration information, data, and code is ready to be pushed to production
- Continuous Deployment
 - Continuous delivery PLUS...
 - The application is automatically deployed to production



Continuous Delivery vs Deployment

CONTINUOUS DELIVERY



CONTINUOUS DEPLOYMENT



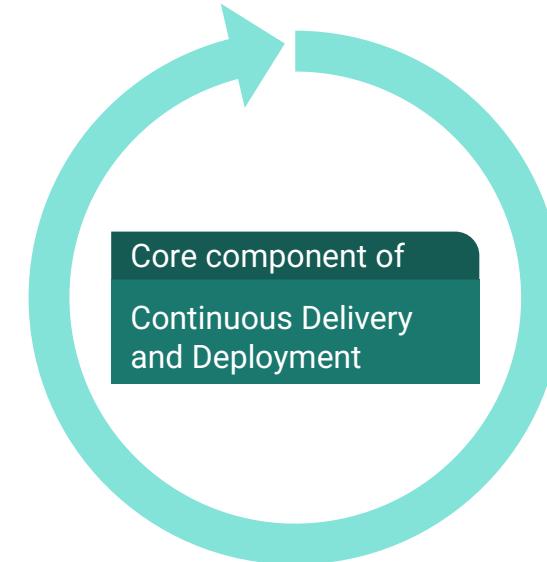
Why CI is good



Rigorous quality checks on main branch



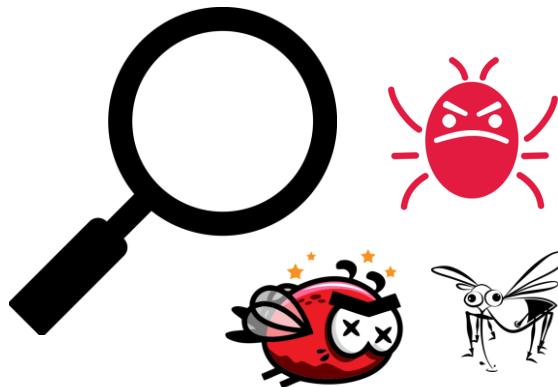
More frequent “integrations” (even before a PR is merged - stopping breakages before they happen!)



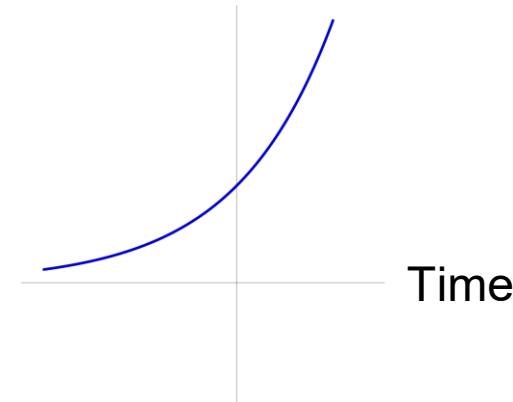
Evidence shows conflicts are resolved more rapidly

Why CI is good

Identifying bugs early!

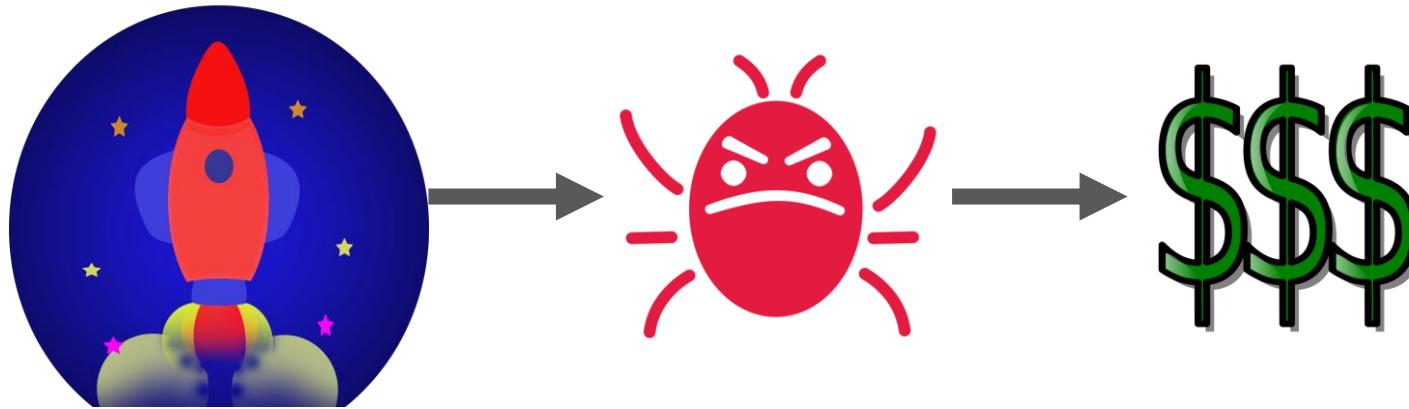


Difficulty of
fixing bug



Why CI is good

When a defect is identified after delivery the cost is enormous.



- Bug tracking, testing, re-testing, release notes, customer notification, down time
- Damage to reputation or liability

Options for CI

- TravisCI
 - Since 2011
 - Cloud-hosted
 - First company to provide CI to open-source projects for free
- CircleCI
 - Since 2011
 - Cloud-hosted
 - Very similar to TravisCI, but slightly more expensive for non-open-source
- Jenkins
 - Since 2011
 - Open-source software
 - You have to run it yourself
- GitHub Actions
 - Since 2018
 - Integrated into the rest of GitHub

By the way, technology review...

A screenshot of a Google search results page. The search query "github actions" is entered in the search bar. The top result is a blog post titled "GitHub Actions Is Slowly Killing Your Engineering Team" by Ian K Duncan, dated 2026-02-05. The post's URL is <https://www.iankduncan.com/engineering/2026-02-05-github-actions-killing-your-team>. The snippet of the post text discusses the author's experience with CI systems and their opinion on GitHub Actions.

github actions

Google Search I'm

| [2026-02-05]

GitHub Actions Is Slowly Killing Your Engineering Team

I was an early employee at CircleCI. I have used, in anger, nearly every CI system that has ever existed. Jenkins, Travis, CircleCI, Semaphore, Drone, Concourse, Wercker (remember Wercker?), TeamCity, Bamboo, GitLab CI, CodeBuild, and probably a half dozen others I've mercifully forgotten. I have mass-tested these systems so that you don't have to, and I have the scars to show for it, and I am here to tell you: GitHub Actions is not good. It's not even fine. It has market share because it's *right there* in your repo, and that's about the

How does it work?

1 Make a commit

You make some commits that form a coherent body of work, e.g. implement a feature.

2 Push to GitHub (any branch)

GitHub triggers a “workflow”

GitHub detects the push event, finds all workflows corresponding to that event, and starts up a virtual machine(s).

4 GitHub runs the build/test workflow

GitHub runs the commands you specify that do unit testing, coverage reports and style checks.

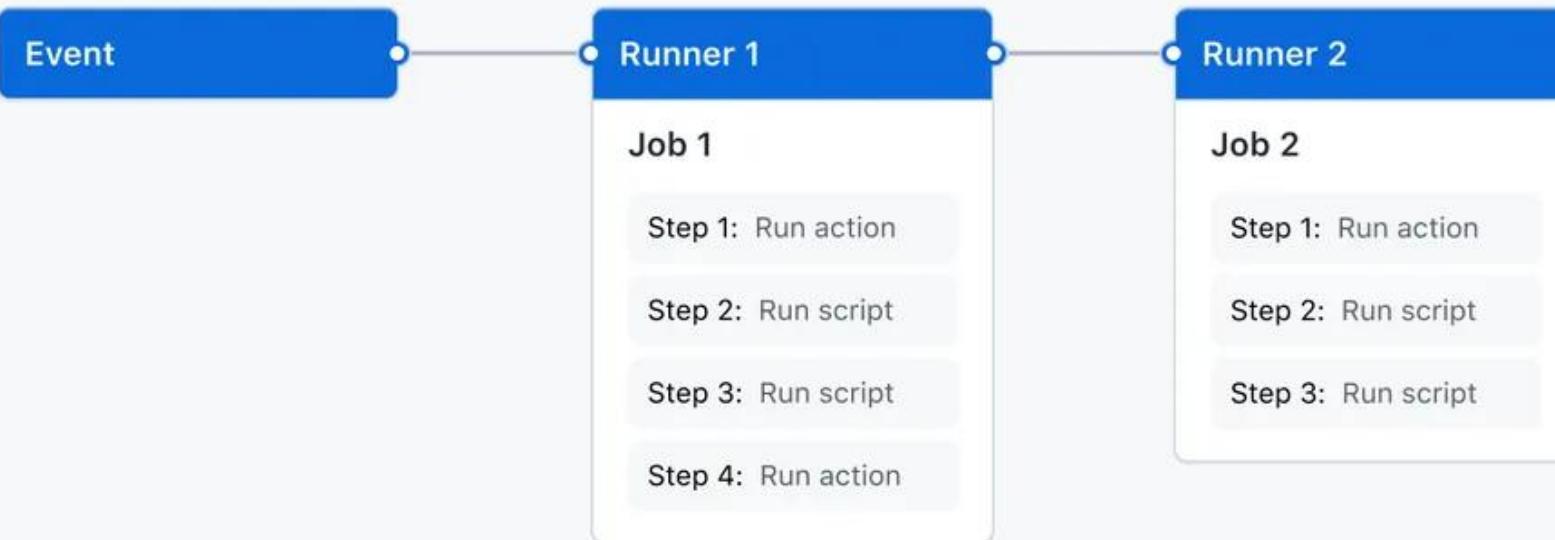
5 Notification of result

The team is notified of the result, usually in email. If the commands fail with an error, the build is reported as failing.

GitHub Actions Concepts

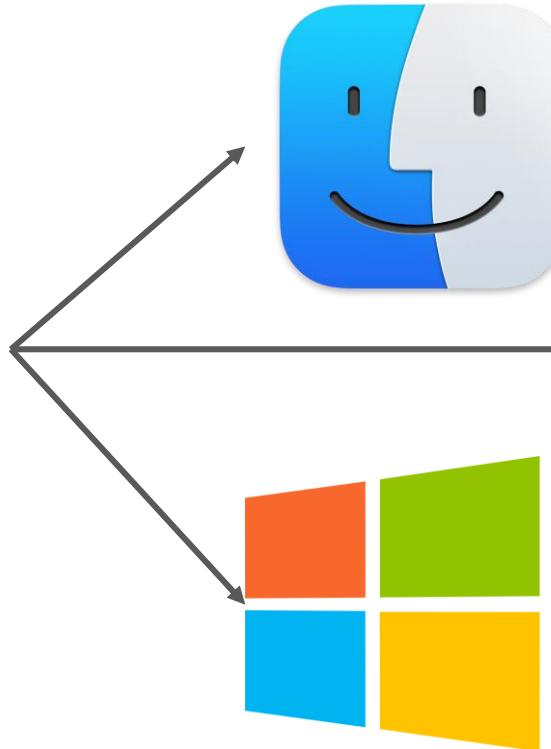
- **Workflow:** an automated process that will run
- **Event:** a specific activity that triggers a workflow (eg push, PR creation, etc)
- **Job:** a sequential set of tasks that is a part of a workflow
 - Jobs can have dependencies on each other
- **Step:** a single task within a job
- **Runner:** a server (VM) that runs a workflow when it is triggered
- **Action:** a complex, reusable step that has been coded separately
 - There are 3rd party actions, but you can write your own

GitHub Actions Workflow



GitHub runners use virtual machines

GitHub  **Ubuntu**

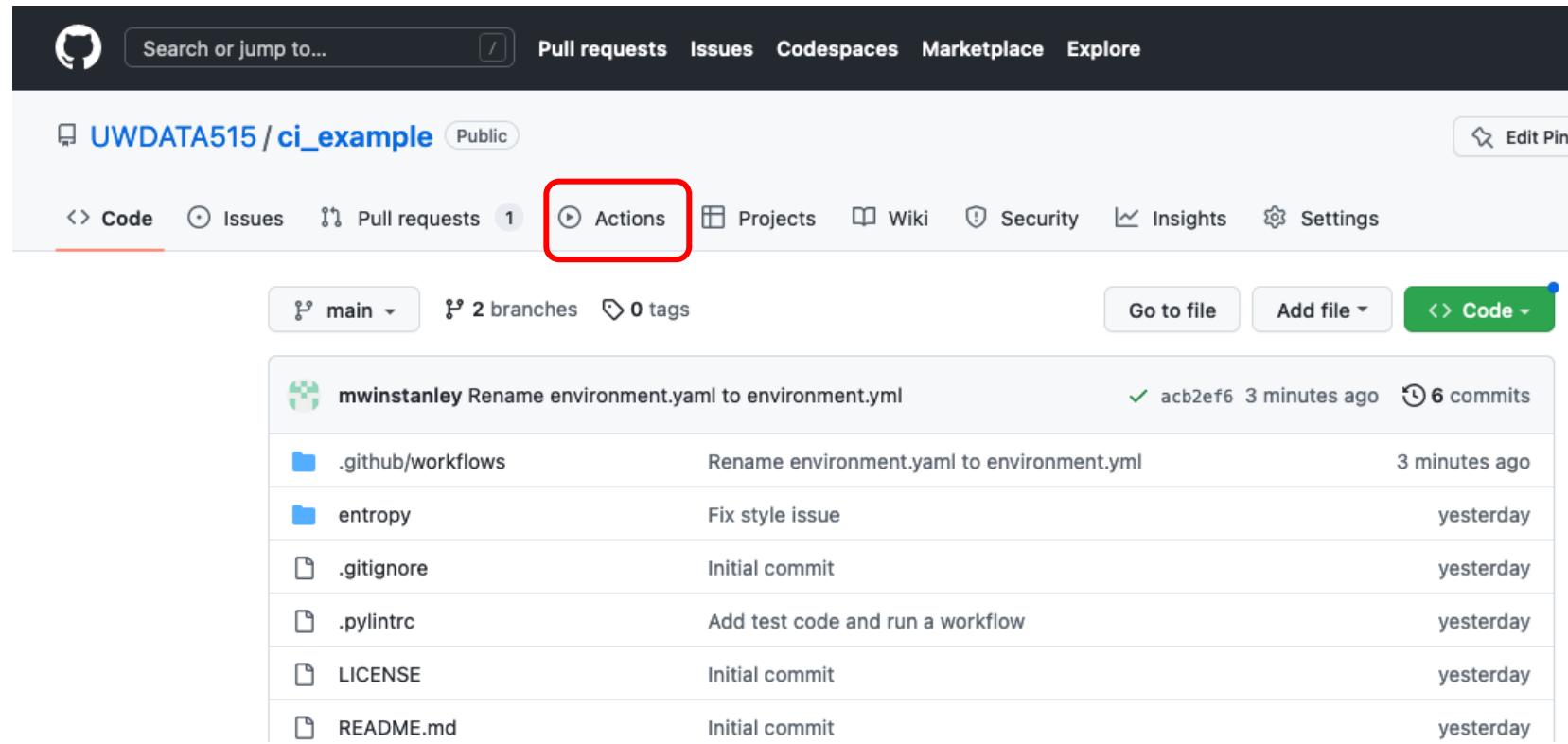


Why do I need more than one job?

- Jobs encapsulate tasks
- Jobs can be run in parallel
 - Eg run your tests on all platforms (Windows, MacOS, Ubuntu)
- Jobs can be configured to run on specific branches
 - So you may run the build/test job on ALL branches
 - But only run the deployment or package push job on the `main` branch

But it's also ok to only have one job if you have a simple CI plan.

Where do I see the results of the workflow runs?



The screenshot shows a GitHub repository page for "UWDATA515 / ci_example". The top navigation bar includes links for Pull requests, Issues, Codespaces, Marketplace, and Explore. Below the repository name, there are buttons for Code, Issues, Pull requests (with a count of 1), Actions (which is highlighted with a red box), Projects, Wiki, Security, Insights, and Settings. The main content area displays the repository's structure: main (branch), 2 branches, 0 tags. It also shows a list of recent commits and their associated actions:

Commit	Message	Time
acb2ef6	mwinstanley Rename environment.yaml to environment.yml	3 minutes ago
	.github/workflows Rename environment.yaml to environment.yml	3 minutes ago
	entropy Fix style issue	yesterday
	.gitignore Initial commit	yesterday
	.pylintrc Add test code and run a workflow	yesterday
	LICENSE Initial commit	yesterday
	README.md Initial commit	yesterday

Enforcement with branch protection rules

Protect matching branches

Require a pull request before merging

When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

Require status checks to pass before merging

Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

Require branches to be up to date before merging

This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

Search for status checks in the last week for this repository

Status checks that are required.

build_test (3.8)

 GitHub Actions ▾ X

build_test (3.10)

 GitHub Actions ▾ X

build_test (3.9)

 GitHub Actions ▾ X

build_test (3.7)

 GitHub Actions ▾ X



Enforcement with branch protection rules



Some checks were not successful

[Hide all checks](#)

3 cancelled and 1 failing checks

! build_test / build_test (3.7) (push) Cancelled after 36s

[Required](#)[Details](#)

! build_test / build_test (3.8) (push) Cancelled after 34s

[Required](#)[Details](#)

! build_test / build_test (3.9) (push) Cancelled after 33s

[Required](#)[Details](#)

✗ build_test / build_test (3.10) (push) Failing after 34s

[Required](#)[Details](#)

Required statuses must pass before merging

All required [statuses](#) and check runs on this pull request must run successfully to enable automatic merging.

A Workflow Example

https://github.com/UWDATA515/ci_example

More powerful features

Check out the documentation to learn all about what you can do!

<https://docs.github.com/en/actions>

Or for Python:

<https://docs.github.com/en/actions/tutorials/build-and-test-code/python>

I can't cover everything in this lecture, and you won't remember anyways.

If you want to use another CI platform...

....you can, but we won't provide active support.

Coverage

- What % of your code is *covered* by tests.
 - High coverage is good
 - High coverage does not *guarantee* your code works but it helps!
- Calculated using a package that we use to run the tests
 - For us: the `coverage` package
 - `coverage run -m unittest discover`
- We often use a tool to report and analyze coverage
 - For us: [Coveralls](#)
- Goal: don't decrease your test coverage when you make a PR!

Exercise: Set up CI for your project repository

THIS IS REQUIRED FOR YOUR PROJECT (by the time it's due)!

1. Create a new working git branch.
2. Create a GitHub Actions workflow to run on push and/or PRs.
3. Push the workflow configuration to GitHub.
4. Make sure the workflow runs successfully.
5. Create a PR to add the workflow and merge it after acceptance.
6. Enforce that tests pass with branch protection.
7. Add **badges** for CI status and code coverage.
8. Improve your test coverage! Aim for as close to 100% as possible.

AT A MINIMUM, the workflow should run tests & code coverage (style next week!)

Special applause for continuous delivery/deployment (*future lecture*)

Pushing to PyPI (publishing your Python package), deploying a web application