COR 5

DISCUSSION QUESTION

You write tests for a project, but they keep breaking because no one runs them. How would you fix this?

Type your answer, but wait for our cue to send it.

Continuous integration

It's well and good to have tests, but how can you ensure that those tests actually run? You could try your best to remember to run them after each change—or you could deploy a *continuous integration* system to run them for you:

- Term originally coined to refer to a development process where changes get added to the main branch soon after they're made.
- Has shifted over time to refer to automated tooling that ensures the main branch stays correct (a.k.a. *green*) as this happens.
- CI systems automatically run builds and test suites for every commit (and often every merge request, too!)

Popular CI tools

- Jenkins
- Travis CI
- GitHub Actions
- GitLab CI/CD
- CircleCI
- Buildkite
- SourceHut Builds

GitHub Actions

- Free of charge and tightly integrated with GitHub
- Well-documented and relatively easy to get started with
- Not open source—can't use if your project isn't on GitHub
  - Although the act project is attempting a fully open reimplementation
- We used GitHub Actions to test your VCS Constructive and BLD Constructive submissions.

Using GitHub Actions

Organized into *workflows*, each of which consists of one or more *jobs*, which are themselves made up of *steps*.

A *step* can either run a shell command directly or delegate to a predefined *action*, which performs a higher-level task all in one go (like a function).

Configured using YAML files in `.github/workflows/`.

```yaml
name: Build and test

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

jobs:
  build:
     runs-on: ubuntu-latest
     steps:
     - uses: actions/checkout@v2

     - name: Build
       run: make

     - name: Run tests
       run: make test
```

demo time!

Continuous integration vs build systems

The defining feature of a CI system is to run tasks *remotely* when certain events occur. This is outside the scope of most (but not all) build systems. However, there's still more overlap than you might expect:

- CI systems generally have their own language to define tasks.
- This language often supports running arbitrary shell commands and setting up dependencies between multiple tasks, much like a build system.
- Some build systems (like Bazel and Buck) can use remote servers to speed up builds.
- **Rule of thumb: make CI tasks a thin a wrapper around your build system**

Security considerations for CI

- CI, especially for pull requests, runs untrusted code
- If the server running CI jobs doesn't clean up properly, jobs can mess with future jobs
  - e.g. one job could add lines to `.bashrc` that get run whenever a shell starts for future jobs
- Even if it does, jobs still have the full privileges of the ruler environment
  - e.g. if you use the same environment to deploy releases as to test PRs, a malicious PR could upload its own release with no vetting
- Lesson: limit access as much as possible when testing untrusted code
- GitHub Actions does this fairly well by default
  - Each job run in a totally clean environment (new container each time, perhaps?)
  - `pull_request` event is not allowed to push to the repo

Distributed CI

The CI systems we've talked about are all centralized, requiring servers to run tasks and often tying you to a specific platform, like GitHub. But there are other ways to run tests

- Option 1: automating local test running
  - Git supports *hooks*, which automatically run shell scripts on commits, pushes, etc.
  - Hook to automatically run tests on each contributor's machine whenever they commit
  - Not fool-proof: can't install hooks by default on a fresh clone for security reasons
- The Linux Kernel model
  - Linux has no single, "blessed" CI system.
  - Several systems run asynchronously, each owned and sponsored by a separate entity.
  - Each one pulls from Linux's mainline repository and reports failures to the mailing list.

Testing your CI

- Making sure your CI does what you want
  - yaml-commit-push-cry loop
- Getting notified when your CI breaks
  - Automatically filing tasks when CI goes *red*
  - But what about silent failures?