Continuous Integration/ Deployment

CS350 Introduction to Software Engineering

Shin Yoo

Site Reliability Engineering (SRE)

Originated from Google circa 2003

- A set of principles and practices that incorporates aspects of SE and IT infrastructure/operations, with the aim of creating highly reliable and scalable software systems (Wikipedia)
  - Core principles:
    - Automate as much as possible
    - Purse just the right amount of reliability
    - Design/engineer the system so that risks of availability, latency, and efficiency are minimized
    - Any aspects of the system should be observable

Push on Green

USENIX magazine *;login:*, 2014

- https://www.usenix.org/system/files/login/articles/login_1410_05_klein.pdf
- Instead of having a fixed release schedule, the “push-on-green” practice aims to be able to release the software whenever all test results are “green” (i.e, pass)
- All changes are tested as they come in
- Pushmasters / release managers observe the testing outcomes, and organizes a push (=release)

Continuous Integration

- The practice of all developers merging their working copies and changes to the main branch several times a day.
- The term was coined by Grady Booch in 1994.
- Extreme Programming (XP) made this one of their main principles.
- Why? Recall Trunk-Based Development
  - The longer you work on your own branch, the more likely that you will run into a conflict/merge hell

Continuous Delivery / Continuous Deployment

- If changes are continuously merged, the follow-up process should also be continuous
- Continuous Delivery: automatically delivers the merged code to the staging/ testing environment
  - Staging Environment: a close replica of the production environment, where you can do end-to-end system testing
- Continuous Deployment: automatically delivers the merged code to the production stage
  - Every change becomes automatically available for the end-user

DevOps

(no clear technical definition but…)

- Development (i.e., making software) + Operations (i.e., running/serving software)
- Under CI/CD, the pipeline between development and release is increasingly handled programmatically: perhaps software engineer can handle the whole thing?
- Advances in infrastructure/technology contributed heavily:
  - Cloud/Elastic Computing
  - Virtualization/Containers

Benefits

- Faster time to market
- Frequent, smaller releases allows us to spot problems earlier on, resulting in more reliable software
- Easier collaborations, because everyone is working on same, or at least very closer versions of, source code
- Cost saving with automation

CI/CD Frameworks

- Jenkins: originally Hudson, being developed at Sun Microsystems by Kohsuke Kawaguchi - after Oracle bought Sun, developers created an open source version. You have to run your own instance. (https://www.jenkins.io/)
- Travis CI: offers cloud-based commercial service, known to be easier to use then Jenkins.
- GitHub Actions: offered as part of GitHub - closer/easier integration with SCM

Jenkins

Travis CI

GitHub Actions

Hands-on: GitHub Actions

https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

- GitHub Actions: a CI/CD platform in GitHub - no need to monitor code changes
- Workflow: a repeatable pipeline that includes one or more jobs
- Event: a repository-related events (such as push) that can trigger a workflow
- Runner: a virtual machine that runs a single job at a time
- Job: a set of individual actions or shell scripts
- Action: a small custom application that performs a repetitive task

Hands-on: GitHub Actions

Preparations

- Create an empty public GitHub repository with the name `cs350-github-actions`
- Let's add a simple Python program called `triangle.py`
- It may not be correct :)
- Commit and push

```python
def triangle(a, b, c):
    if a < 0 or b < 0 or c < 0:
        return -1
    elif a + b < c or b + c < a or c + a < b:
         return -1
    elif a == b or b == c or c == a:
        return 1
    elif a == b and b == c:
        return 2
    else:
        return 4
```

Hands-on: GitHub Actions

Adding Tests

- Now let's add a test!
- Create test_triangle.py
- Try executing the tests:
  - First, install pytest (`pip install pytest`)
  - Second, either execute `pytest`, or `python -m pytest`

```python
from triangle import triangle

def test_invalid1():
    assert triangle(-1, 0, 1) == -1

def test_equilateral():
    assert triangle(3, 3, 3) == 1
```

Hands-on: GitHub Actions

Adding a workflow

- It’d be nice if the tests are automatically executed whenever new commit is pushed to GitHub!
- Create `.github/workflows`
- Inside, create `test.yml`
- Commit & push

```yaml
name: Python Test

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - name: Set up Python 3.11
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install pytest coverage

      - name: Test
        run: python -m pytest
```

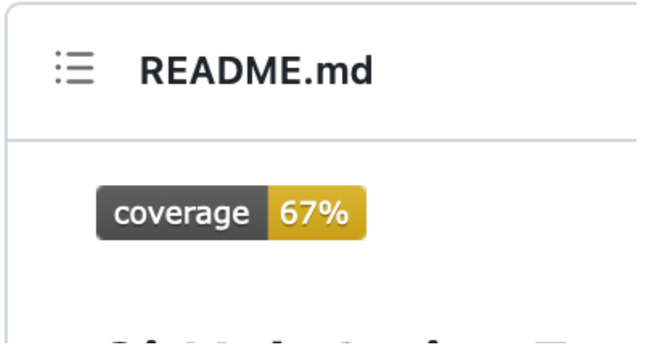
Hands-on: GitHub Actions

Adding Features

- First, add a test case (a new test function in `test_triangle.py`) to reveal the bug in `triangle.py`
- Try pushing the new test
- Second, fix the bug, push the fix, and check the workflow result.
- Finally, see if you can add a branch in `triangle.py` so that it checks for right angle triangles, whose results should be 3.

Hands-on: GitHub Actions

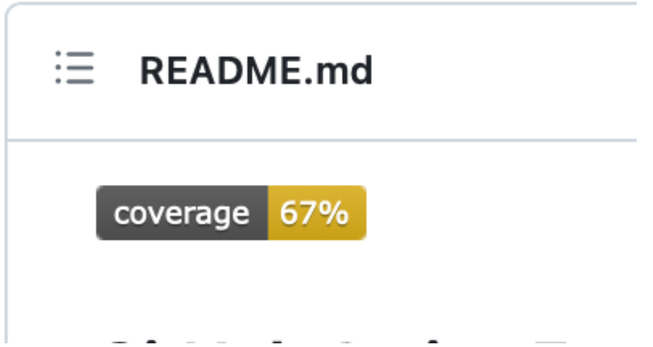
Badge of Honor?



- Let’s try a little hack: have you seen a badge like this on GitHub? How do they do it?
- The workflow, Part 1
  - On push —> checkout new changes —> execute the included tests, but do it while measuring coverage —> execute an action that turns coverage results into badge image

Hands-on: GitHub Actions

Badge of Honor?

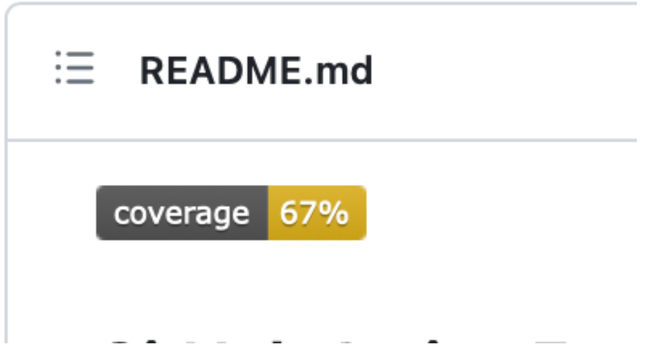



```yaml
- name: Measure coverage with PyTest
  run: |
    coverage run --branch -m pytest
    coverage report
    coverage json

- name: Coverage Badge
  uses: tj-actions/coverage-badge-py@v2
```

Hands-on: GitHub Actions

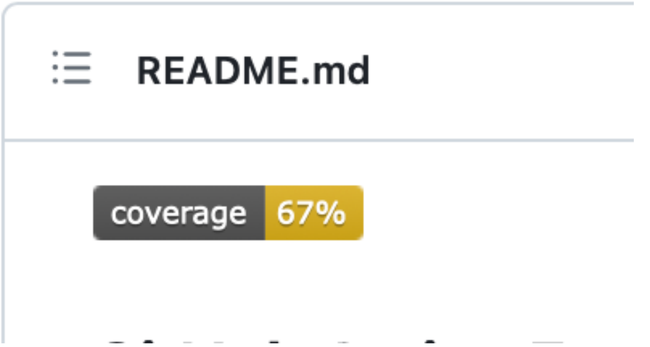
Badge of Honor?



- The workflow, Part 2
  - We now have generated `coverage.svg`, which is the vector image for the badge. It is now in the runner VM.
  - How do we display this in `README.md`??
  - Idea 1: push the `coverage.svg` into our repository, and link the image.
  - What is missing from idea 1? 🤠

Hands-on: GitHub Actions

Badge of Honor?

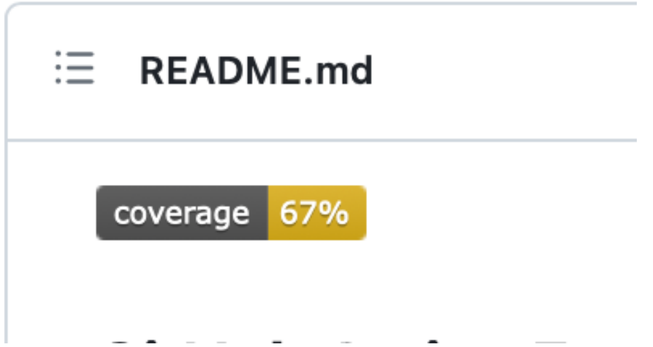


- The workflow, Part 2
  - Idea 2: push the `coverage.svg` into our repository, but on a SEPARATE branch reserved for the badge! This won't mess with whatever branch that we are developing on.
  - Linking can be done via URL:

    `https://rawgithubusercontents.com/[user]/[repository]/[branch]/[filepath]`

Hands-on: GitHub Actions

Badge of Honor?



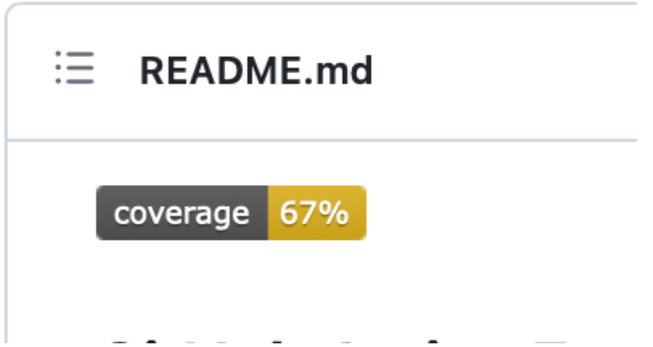

```yaml
- name: Verify Changed files
  uses: tj-actions/verify-changed-files@v13
  id: verify-changed-files
  with:
    files: coverage.svg

- name: Commit files
  if: steps.verify-changed-files.outputs.files_changed == 'true'
  run: |
    git config --local user.email "github-actions[bot]@users.noreply.github.com"
    git config --local user.name "github-actions[bot]"
    git add coverage.svg
    git commit -m "Updated coverage.svg"
```

name: Push changes
if: steps.verify-changed-files.outputs.files_changed == 'true'
uses: ad-m/github-push-action@master
with:
github_token: ${{ secrets.github_token }}
branch: badge
force: true

Hands-on: GitHub Actions

Badge of Honor?

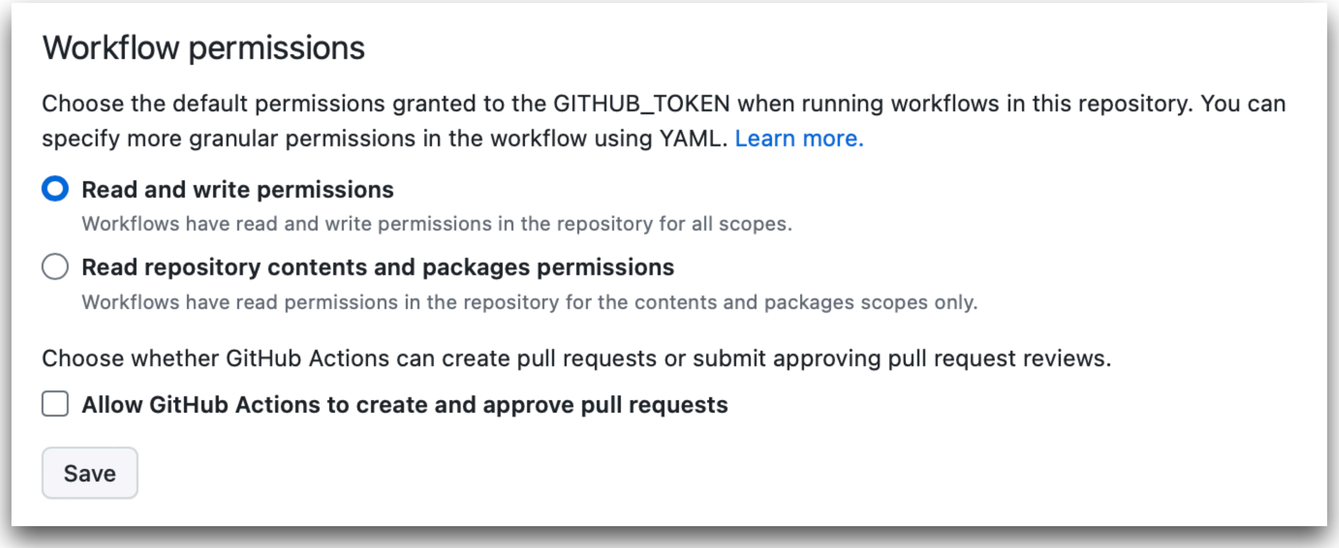



```yaml
- name: Push changes
  if: steps.verify-changed-files.outputs.files_changed == 'true'
  uses: ad-m/github-push-action@master
  with:
    github_token: ${{ secrets.github_token }}
    branch: badge
    force: true
```

Hands-on: GitHub Actions

Badge of Honor?

- You need to allow your actions write permission to the repository
- Configure this under Settings > Actions > General



Hands-on: GitHub Actions

Badge of Honor?

- Add a link to the generated badge in the other branch into `README.md` in your project. Commit & push.
- Finally, add a new test that covers new cases, and see if you can see the badge.

```
![coverage badge](https://
raw.githubusercontent.com/ntrolls/cs350-
github-actions/badge/coverage.svg)
```

A note on the coverage badge hack

- Pushing the badge image into the repository is not REALLY ideal, even if it lives in a separate branch.
- An alternative would be:
  - Move the file to your own web server, which will host the image only, or
  - Depend on a 3rd party service (such as https://coveralls.io/) that will send the results to their own server to be hosted
- We used this hack as a demonstration of push action, etc

Summary

- VCM with automated build completes what we call CI.
- It can be extended to delivery/deployment, resulting in CD.
- CI/CD is a lasting impact from the agile paradigm shift, due to the many benefits it has.
- Know how to configure basic GitHub actions for your project.