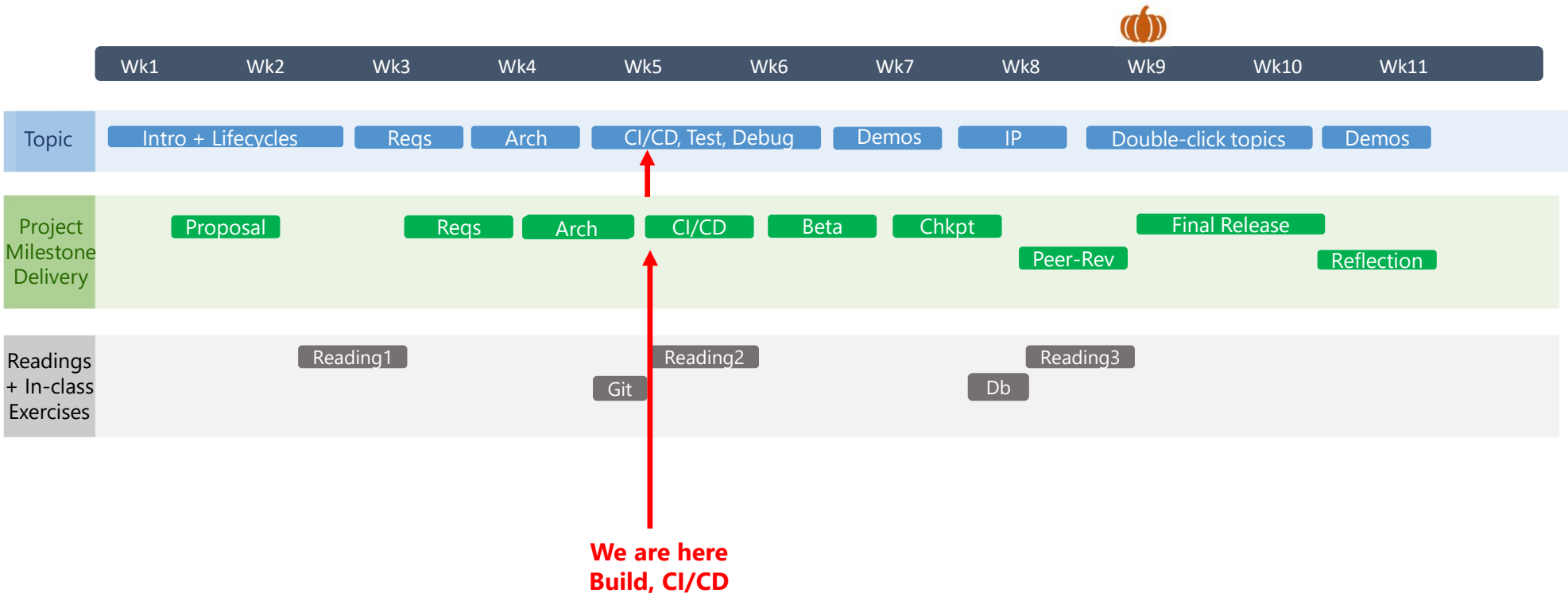
CSE 403 Software Engineering

Build systems & Continuous Integration and Deployment

Autumn 2023

We are moving through the SDLC components



Today's outline

- **Build systems**
- **Continuous integration and deployment systems**
  - What are these
  - How do they relate
  - Best practices
  - Ideas to explore for your projects

Assignment 3:
Git, Testing, and Continuous Integration
+
Reading Reflection 2

Due 10/31

What does a developer do?

The code is written … now what?

- Get the source code
- Install dependencies
- Run static analysis
- Compile the code
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!
- Operate, monitor, repeat

What does a developer do?

The code is written … now what?

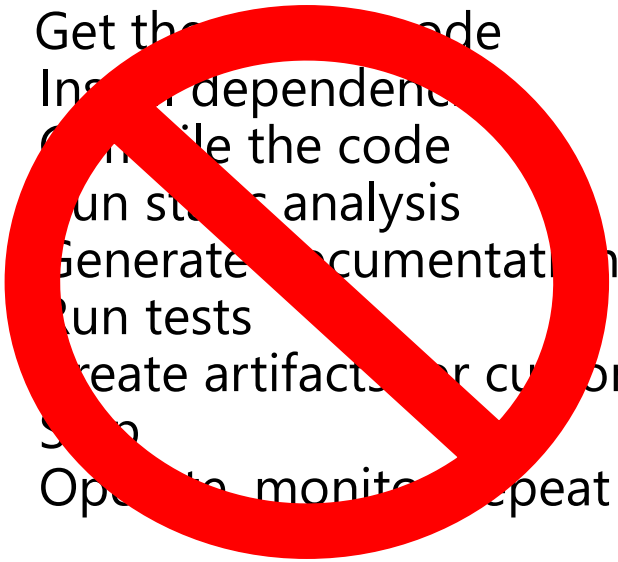
- Get the source code
- Install dependencies
- Run static analysis
- Compile the code
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!
- Operate, monitor, repeat

Which of these tasks should be handled manually?

What does a developer do?

The code is written … now what?

- Get the code
- Install dependencies
- Compile the code
- Run static analysis
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship
- Operate, monitor, repeat



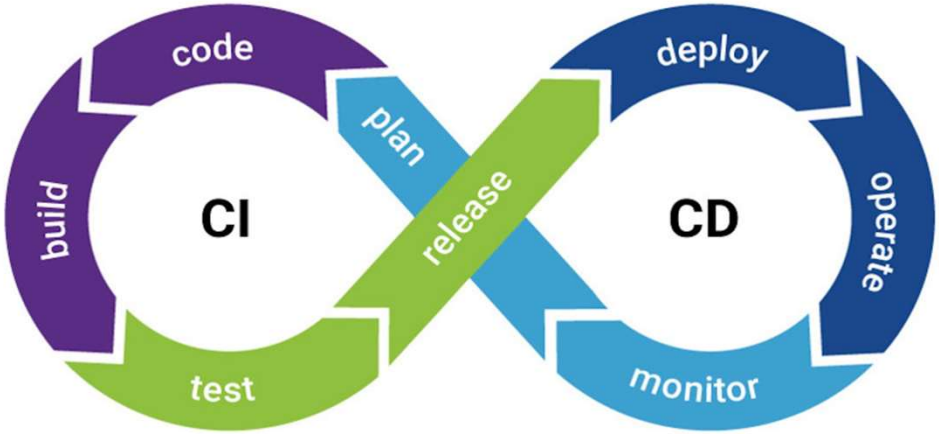
Which of these tasks should be handled manually?

NONE!

Instead, orchestrate with a tool

- **Build system**: a tool for automating compilation and related tasks
- Is a component of a **continuous integration/deployment system** as today we automate more than just the build step of producing shippable software

- ✓ Get the source code
- ✓ Install dependencies
- ✓ Run static analysis
- ✓ Compile the code
- ✓ Generate documentation
- ✓ Run tests
- ✓ Create artifacts for customers
- ✓ Ship!
- ✓ Operate, Monitor, Repeat



Adding to our SE best practices list

- Automate, automate, automate everything!
- Always use a build tool (one-step build) ☺
- Use a CI tool to build and test your code on every commit
- Don't depend on anything that's not in the build file
- Don't break the build!

So how can a build system help us?

1. **Dependency management**
   1. Identifies dependencies between files (including externals)
   2. Runs the compiles in the right order to pick up the right dependencies
   3. Only runs the compiles needed due to dependency changes

2. **Efficiency and reliability**
   1. Automates the build process so that new and old team members, even working in different dev environments, can move quickly from development to shipping code
   2. Eliminates the chance of missing steps due to tribal knowledge and/or simply errors

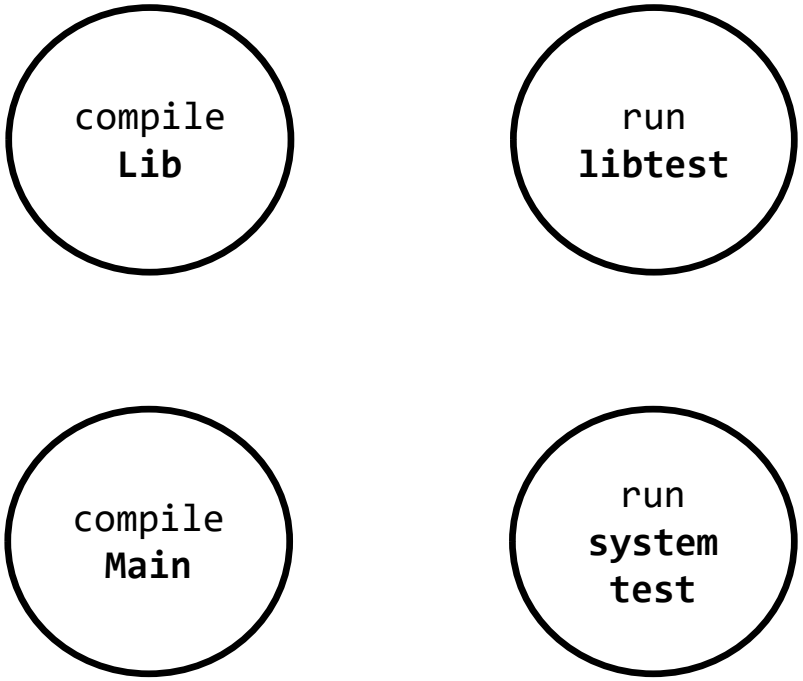
Let's focus on dependency management

Simple example:

```
% ls src/
   Lib.java
   LibTest.java
   Main.java
   SystemTest.java
```

Build systems: dependencies between tasks

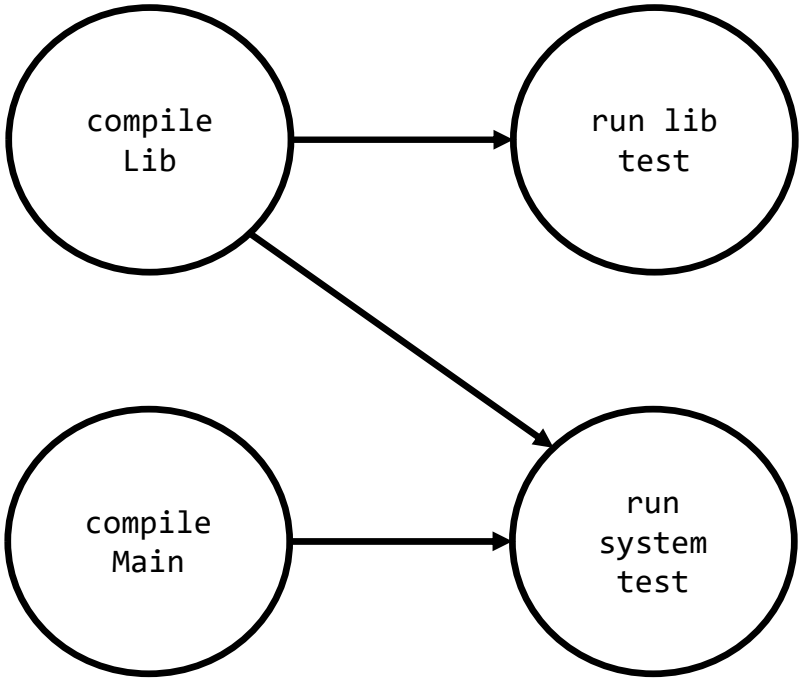
```
% ls src/
    Lib.java
    LibTest.java
    Main.java
    SystemTest.java
```



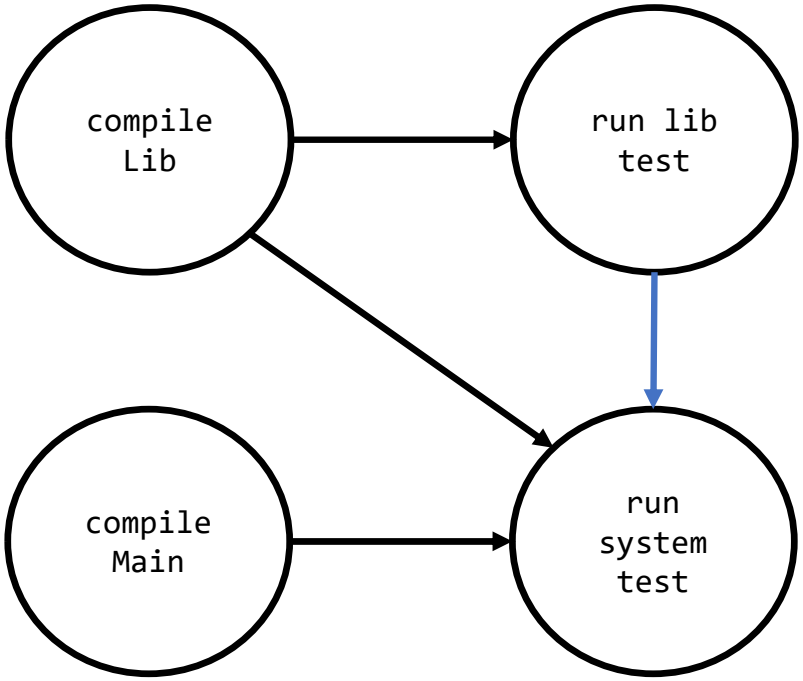
What are the dependencies between these tasks?

And why do I care?

Build systems: dependencies between tasks

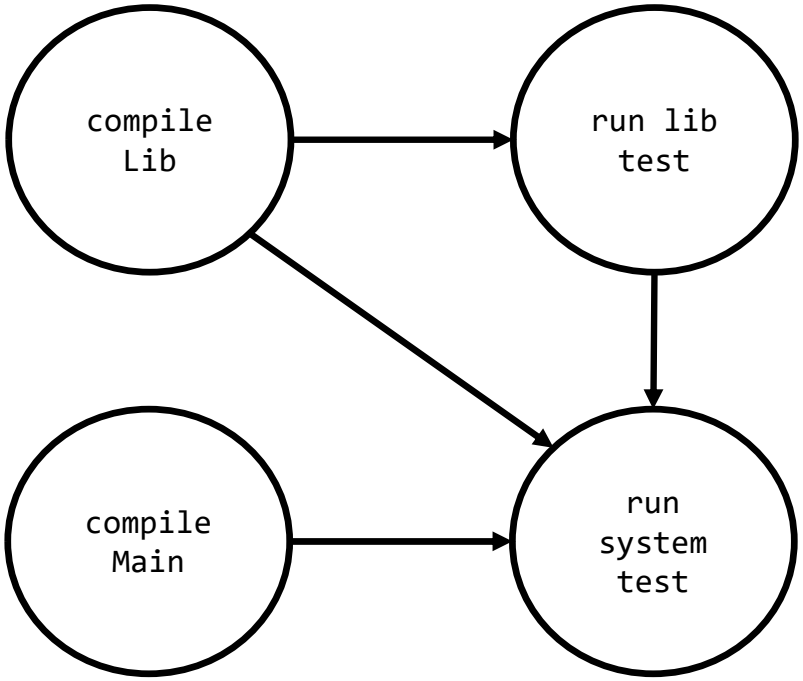


Build systems: dependencies between tasks



Build systems: dependencies between tasks

In what order should we run these tasks?



Build systems can determine task order

Large projects have thousands of tasks

- Dependencies between tasks form a directed acyclic graph
- Build tools use a topological sort to create an order to compiles
  - Order nodes such that all dependencies are satisfied
  - Implemented by computing indegree (number of incoming edges) for each node
  - No dependencies go first and open door to the others
  - See Appendix for example

External code (libraries) also can be complex

- Build systems can manage these dependencies as well!

Let's focus on efficiency and reliability

Actually, I think we understand these ☺

So, let's focus on the opportunity for **static analysis** BEFORE the compile step

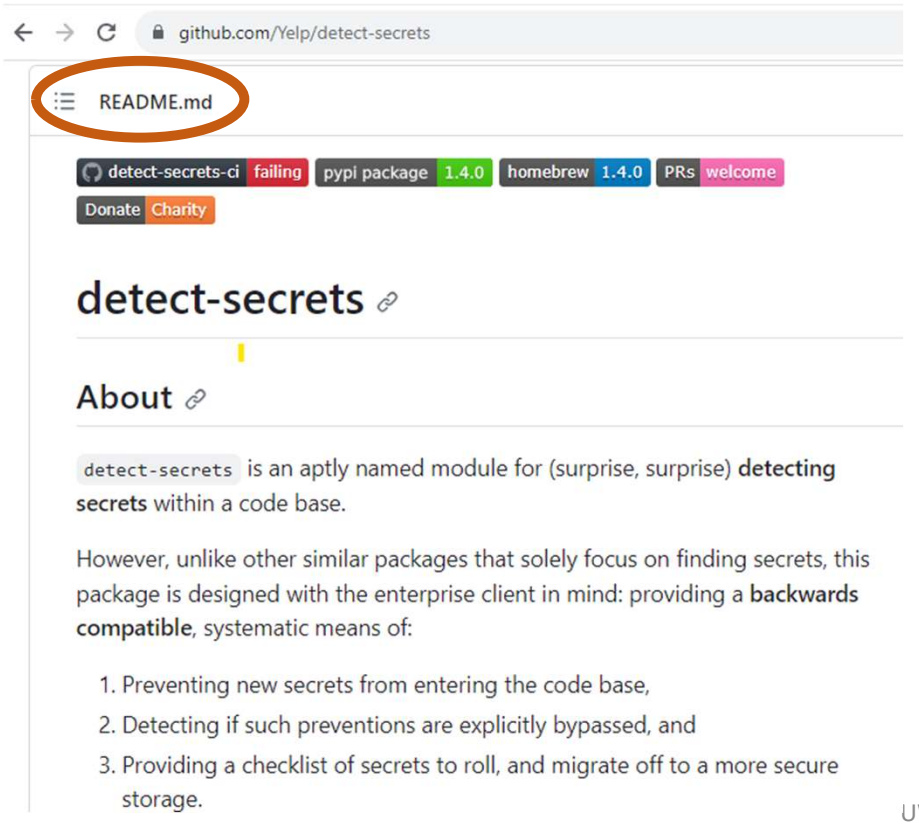
Examples:

- Credential scan
- Date scan
- Sensitive data scan

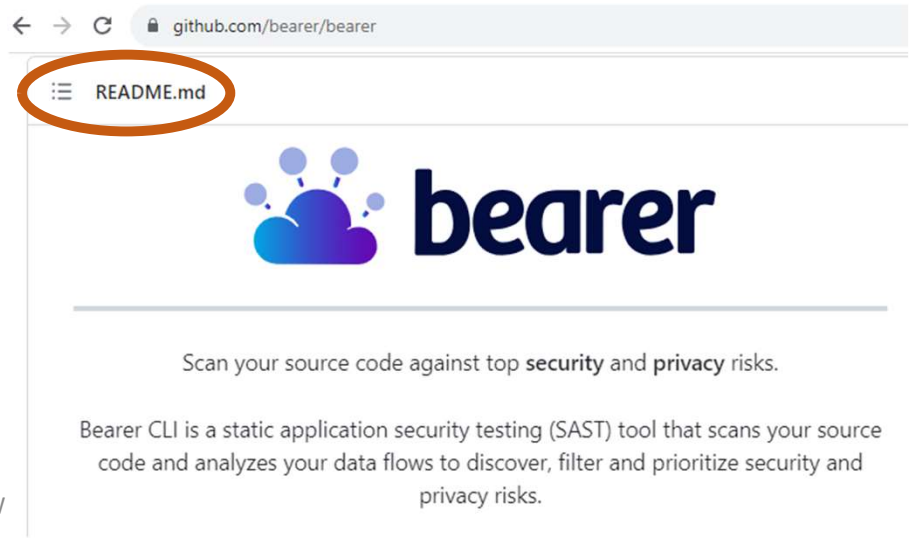
What might be others?

Is this worthwhile?

Build systems: opportunity for static analysis



Could these types of static analysis tools be run earlier than build?



Here's an example build system 'input'

Assignment: evaluate and select a build system

Many other options!

Over to you to research

JAVA+		
	gradle	Open-source successor to **ant** and **maven**
	bazel	Open-source version of Google's internal build tool (blaze)
PYTHON		
	hatch	Implements standards from the Python standard (uses TOML files, has PIP integration)
	poetry	Packaging and dependence manager
	tox	Automate and standardize testing
JAVASCRIPT		
	npm	Standard package/task manager for Node, "Largest software registry in the world."
	webpack	Module bundler for modern JavaScript applications
	gulp	Tries to improve dependency and packing

Today's outline

- Build systems
- **Continuous integration and deployment systems** ⟵ We are here

  - What are these and
  - How do they relate
  - Best practices
  - Ideas to explore for your projects

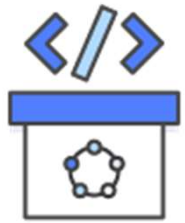
CI/CD: What's the difference?

Continuous Integration (CI)

- Devs regularly integrate code into a shared repository
- System builds/tests automatically with each update
- Complements local developer workflows (e.g., may run diff tests)
- **Goal:** to find/address bugs quicker, improve quality, reduce time to get to working code

Continuous Deployment (CD) [Continuous **Delivery**]

- Builds on top of CI
- Automatically pushes changes to [staging environment and then] production
- **Goal:** always have a deployment-ready build that has passed through a standardized testing process



https://aws.amazon.com/devops/what-is-devops/

Just like build, there are many CI tool options

Assignment: Research, evaluate and choose a CI system

Continuous integration basics

- A CI **workflow** is **triggered** when an **event** occurs in your [shared] repo
  - Example events
    - Push
    - Pull request
    - Issue creation

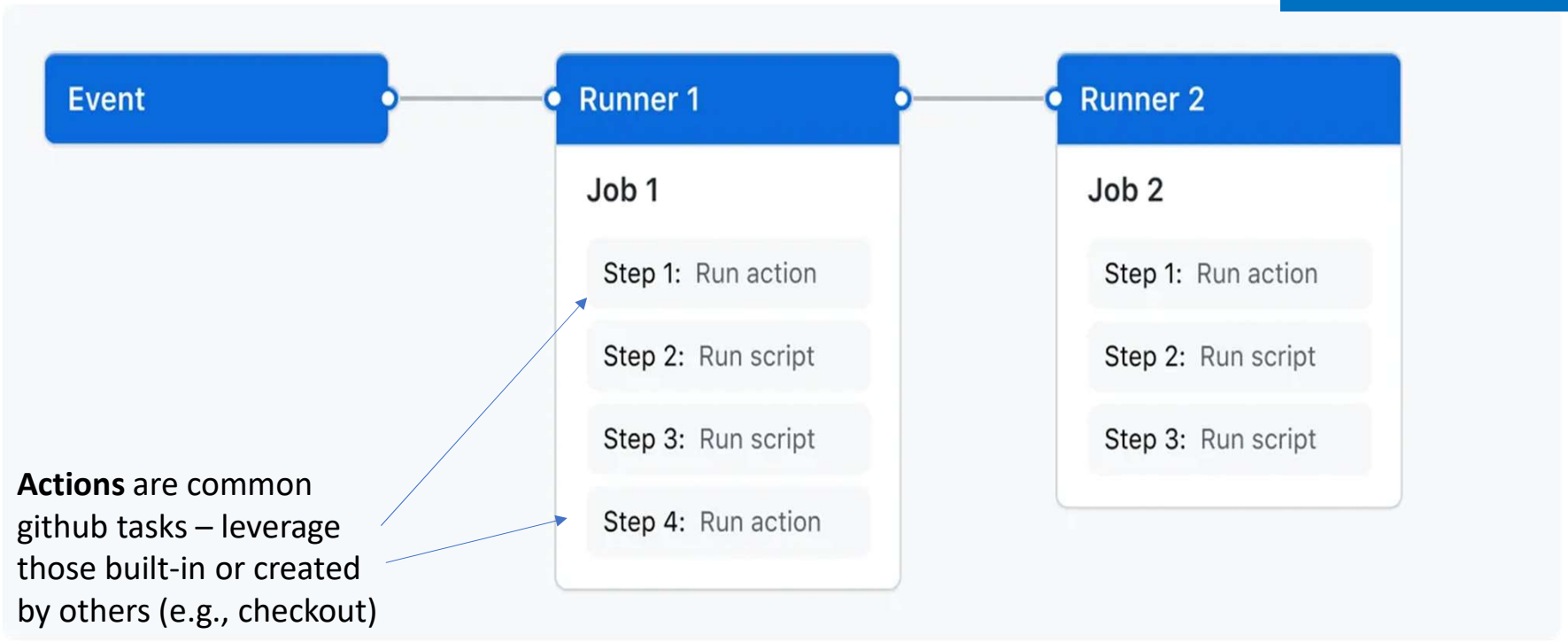
- A workflow contains **jobs** that run in a defined order
  - A job is like a shell-script and can have multiple steps
  - Jobs run in their own vm/container called a **runner**
  - Example jobs
    - Run static analysis
    - Build, test
    - Deploy to test, deploy to prod

Using GitHub CI terminology but concepts span other CI systems

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

CI basics (w/ GitHub CI)

What SW architecture is this using?



Actions are common github tasks – leverage those built-in or created by others (e.g., checkout)

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

Let's try writing our own simple workflow

Follow along at:

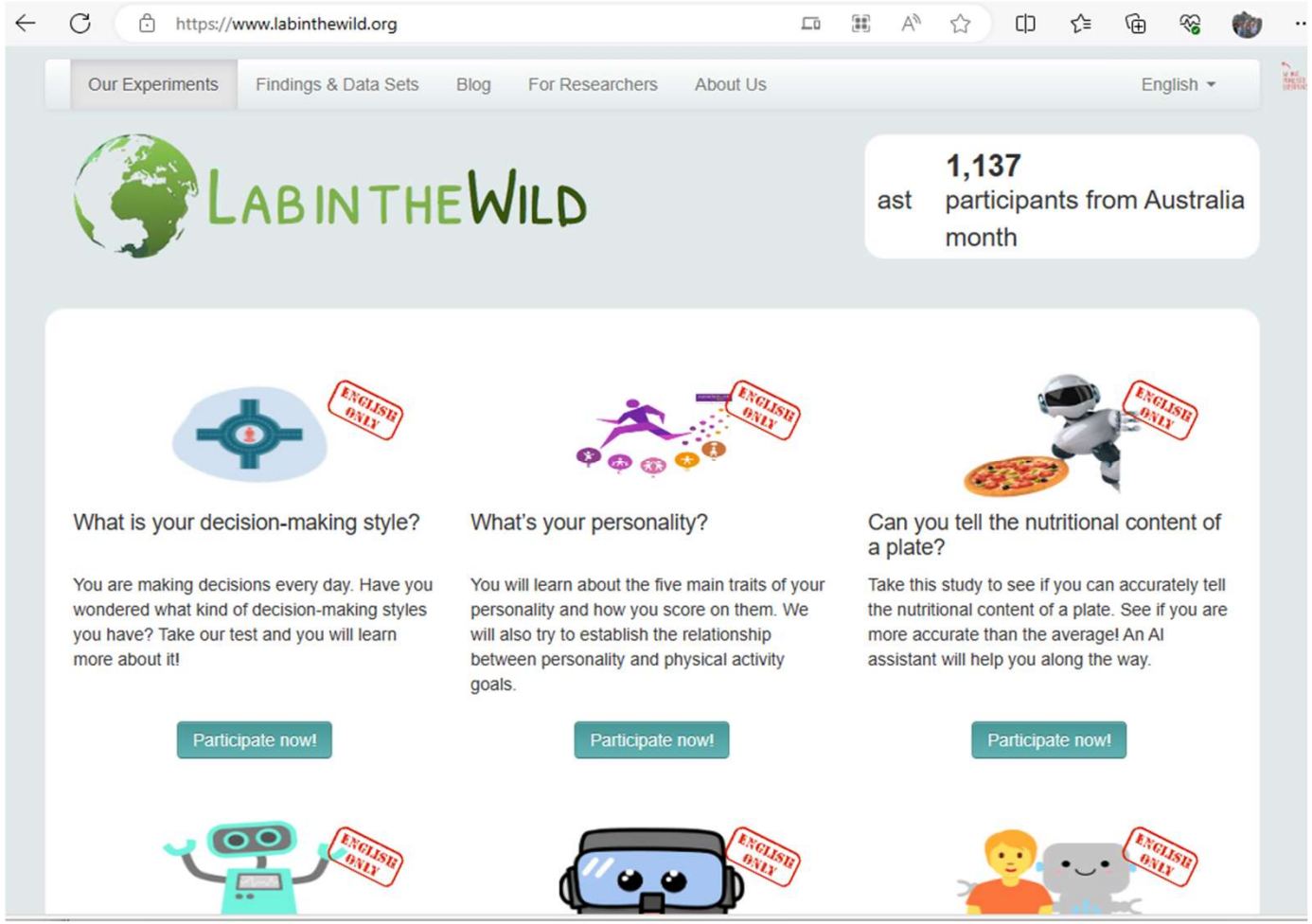
https://github.com/alv880/UW-CSE403-Au23-Projects

Nice light starter tutorial – Automation Step by Step:
https://www.youtube.com/watch?app=desktop&v=ylEy4eLdhFs

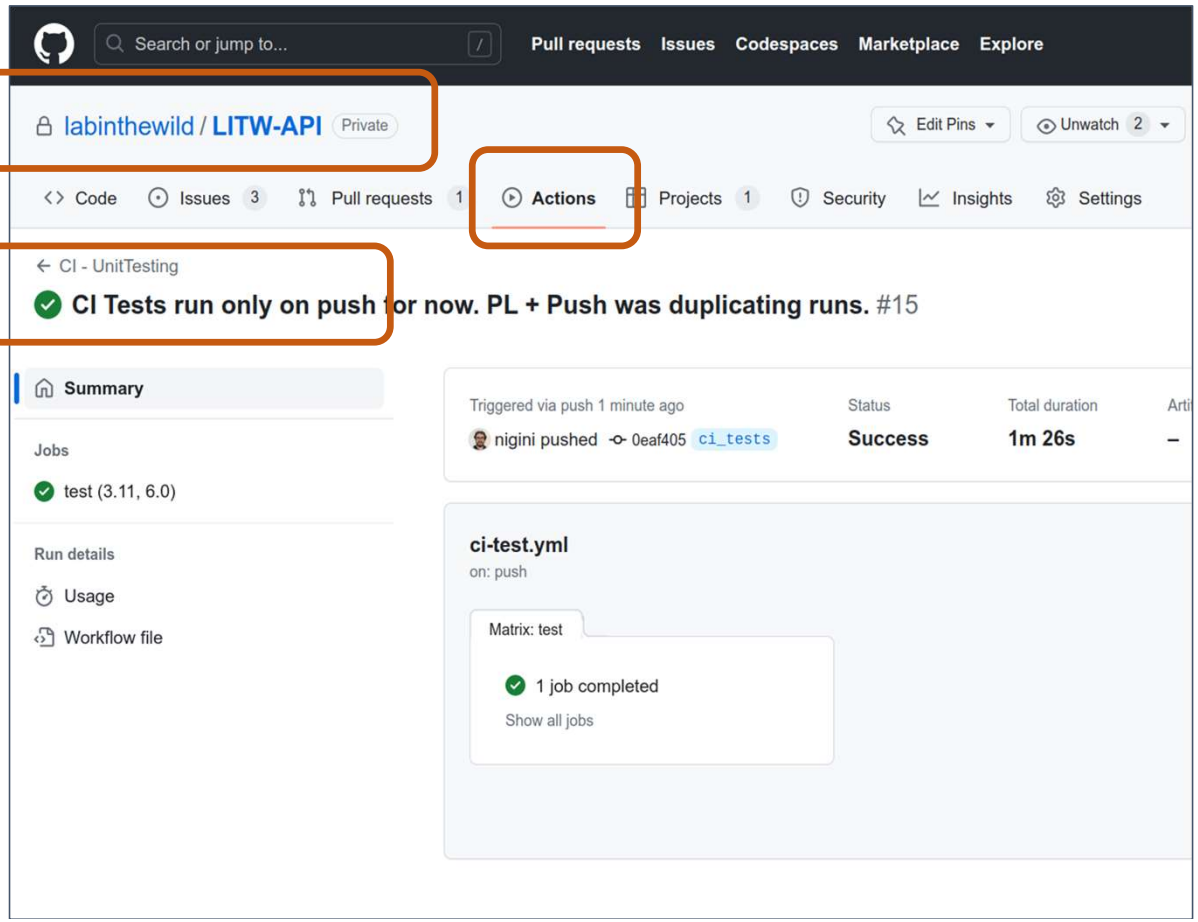
Example: CI at work at UW

Lab In The Wild is a research project drawing survey input from diverse community

– Nigini Oliveira researcher and 403 prof too provided this example



Example: CI with Github actions



Example: CI with Github actions

Unit tests are triggered on every push of new code

```yaml
name: CI - UnitTesting
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    strategy: <2 keys>

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v3
      with: <1 key>
    - name: Set up MongoDB ${{ matrix.mongodb-version }}
      uses: supercharge/mongodb-github-action@1.8.0
      with: <1 key>
    - name: Install dependencies
      run: python3 -…tall hatch
    - name: Pre-fly setup
      run: cp $GITHU…GITHUB_ENV
    - name: Test with hatch
      run: |
        hatch run test:test
```

Workflow name

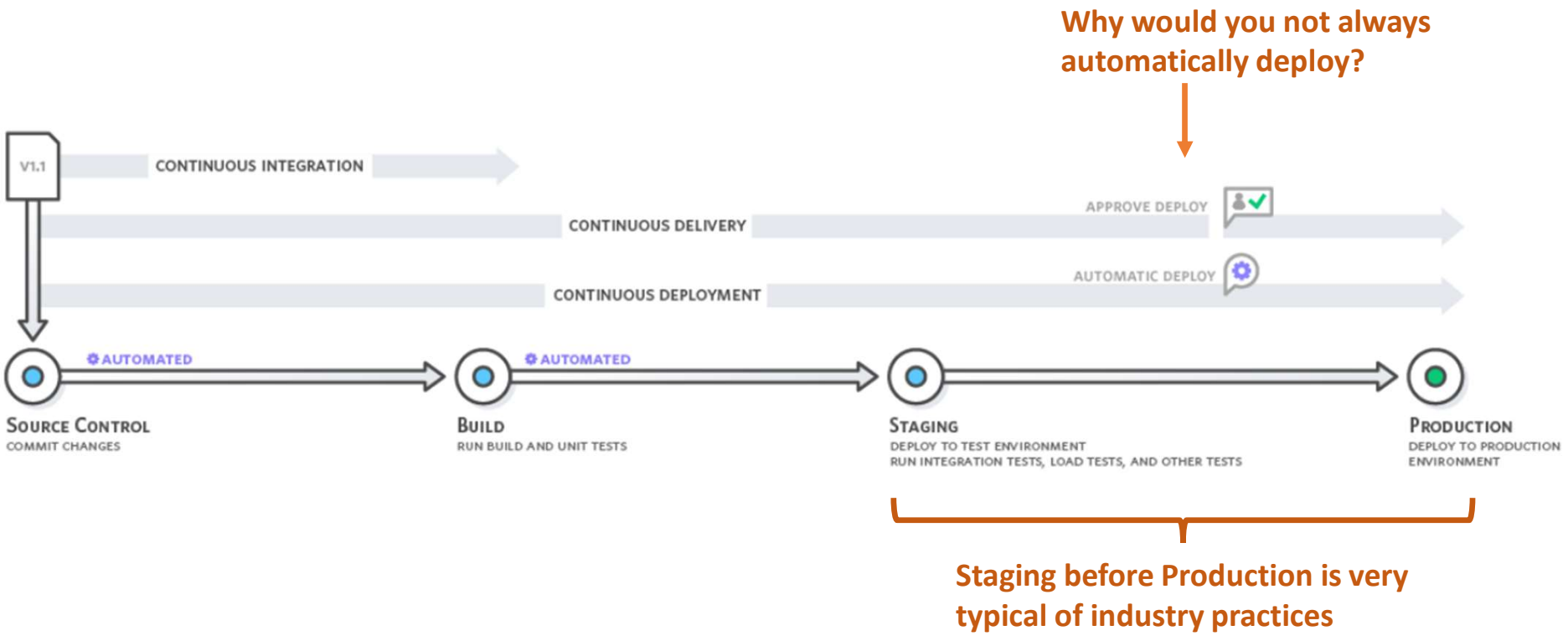
Trigger

Linux OS environment

Code reuse with established “actions”

One command to run test suite

Continuous delivery/deployment basics

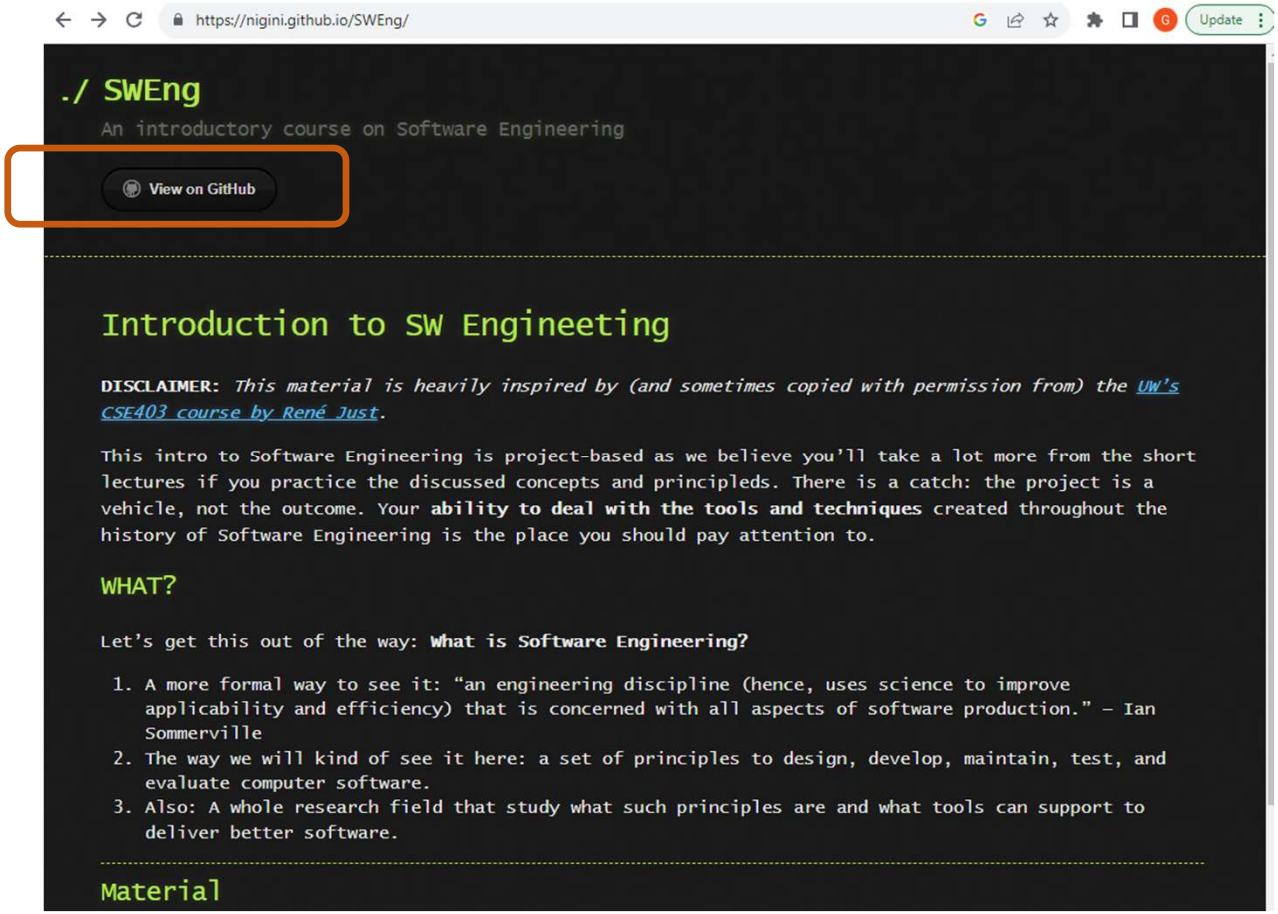


What is Continuous Delivery? – **Amazon Web Services**

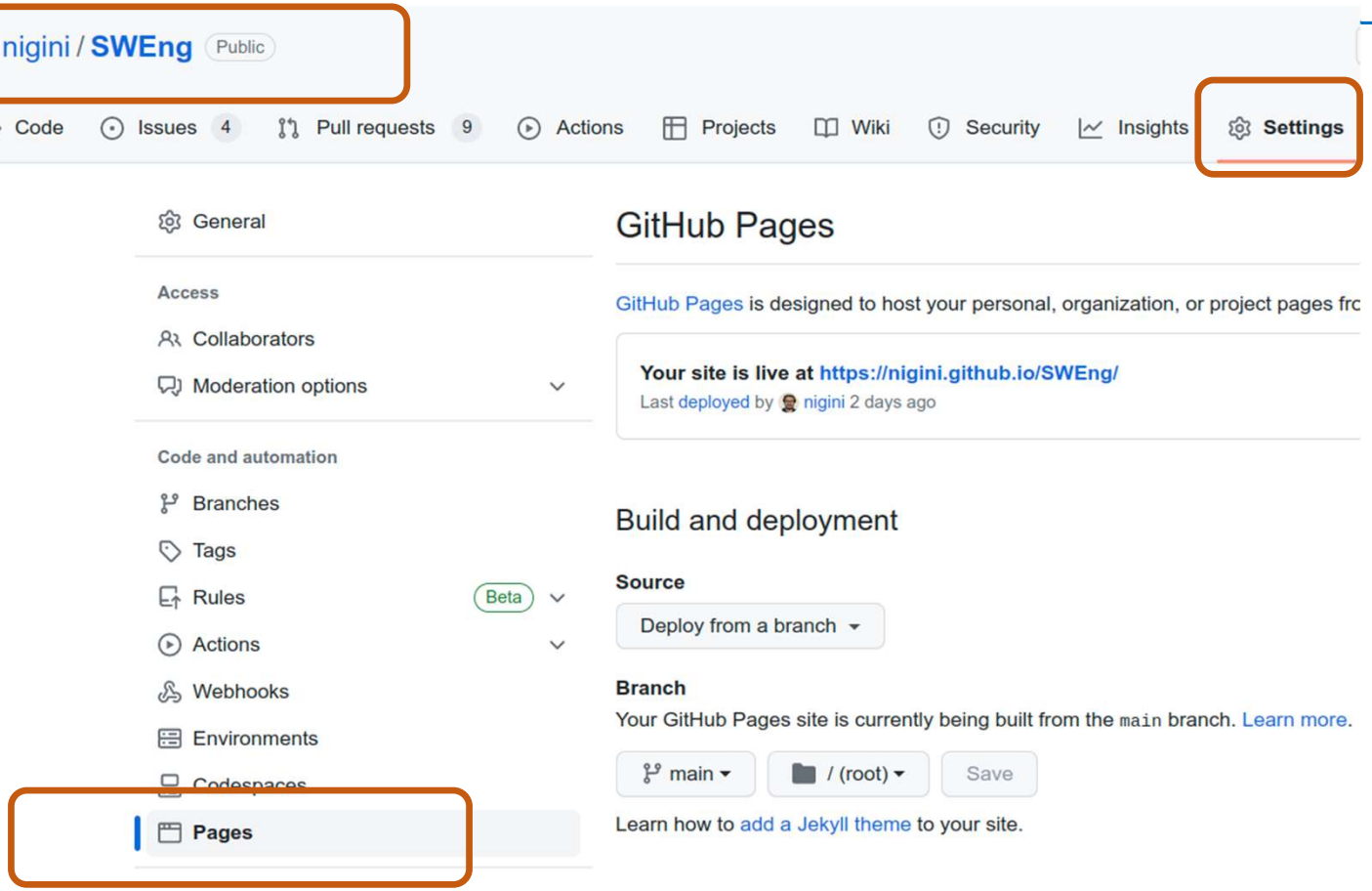
Example: CD with GitHub Pages

Spring ‘23 class hosted their 403 class website on GitHub pages

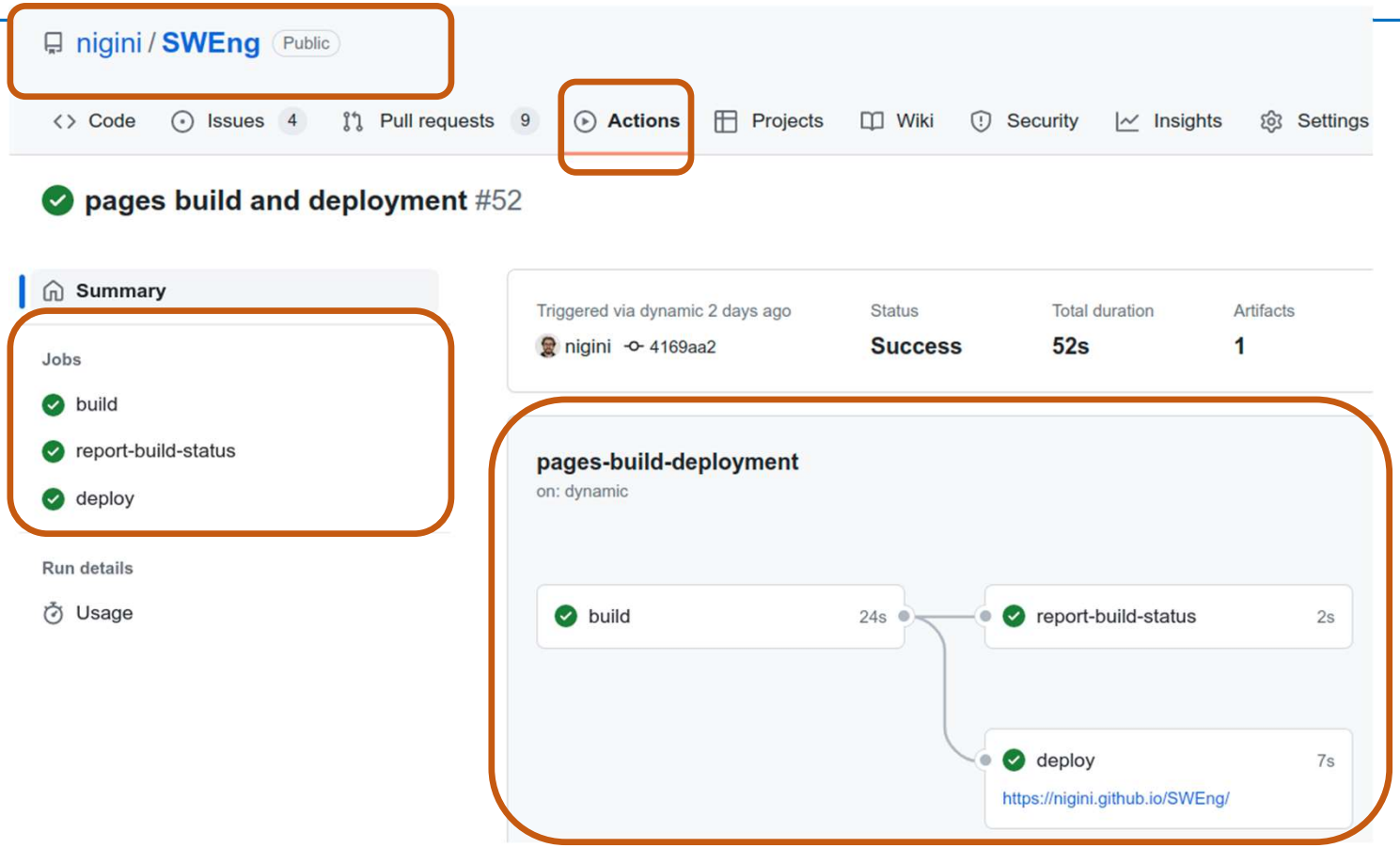
Used CD so that updates triggered publishing the website update



Example: CD configuration

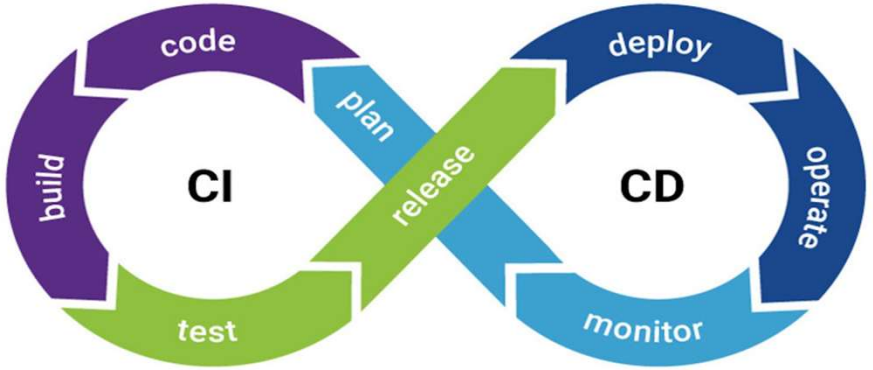


Example: CD configuration



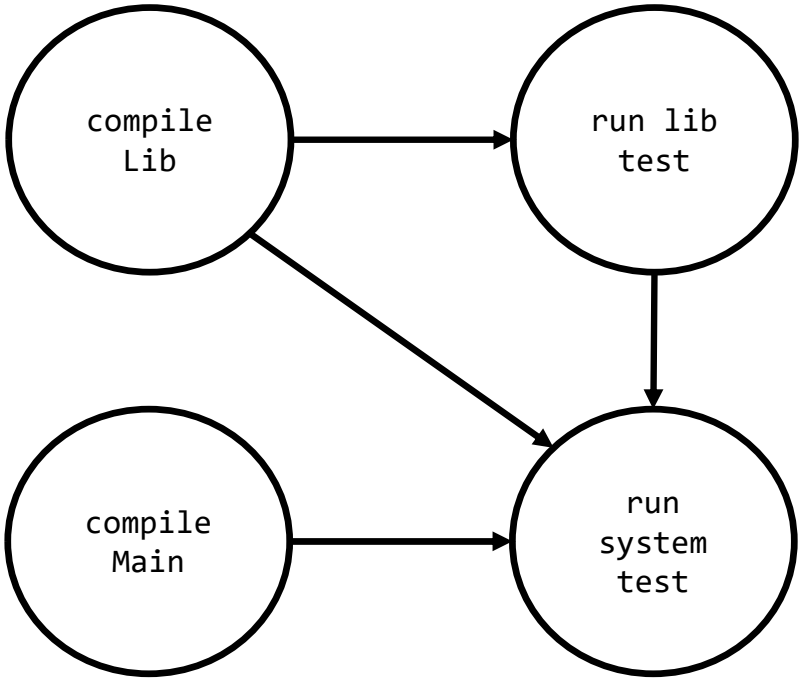
Build, CI - Remember these best practices

- Automate, automate, automate everything!
- Always use a build tool (one-step build)
- Use a CI tool to build and test your code on every commit
- Don't depend on anything that's not in the build file
- Don't break the build!



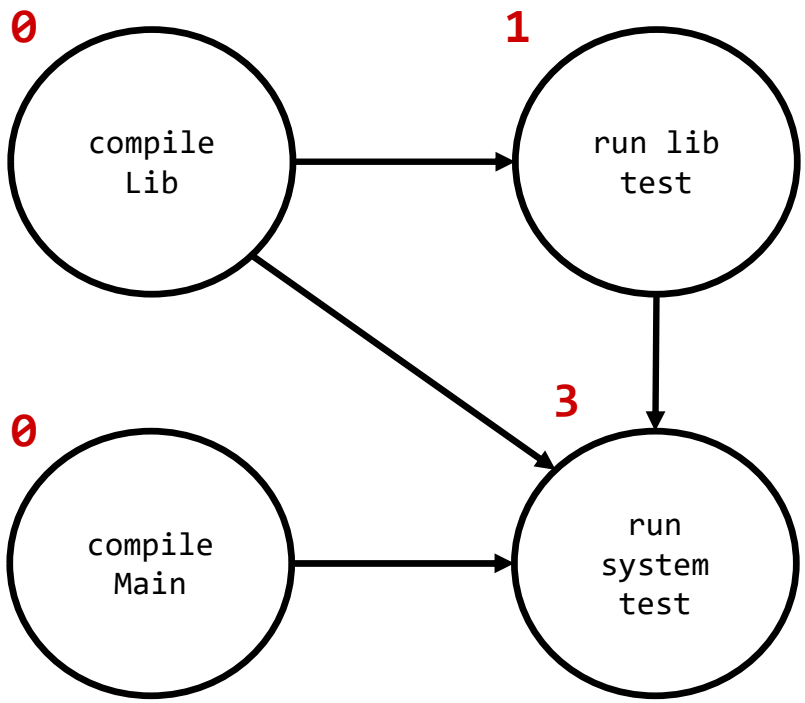
Appendix - Topological sort example

Build systems: topological sort

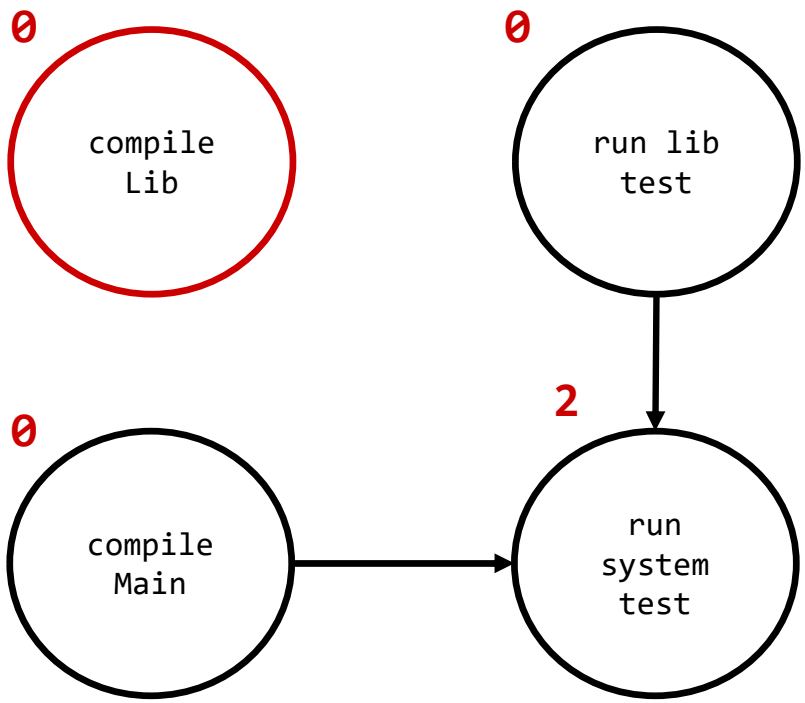


What's the indegree of each node?

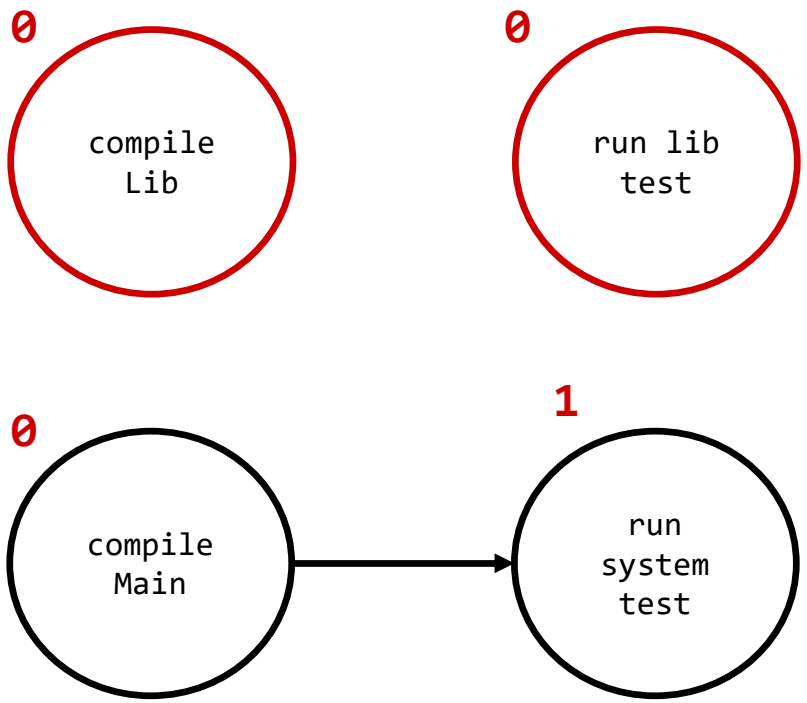
Build systems: topological sort



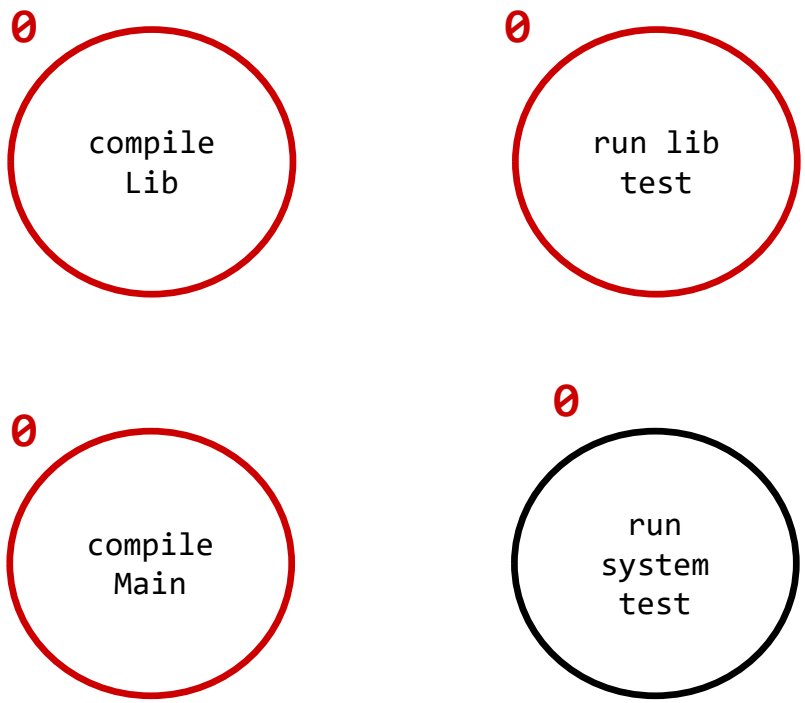
Build systems: topological sort



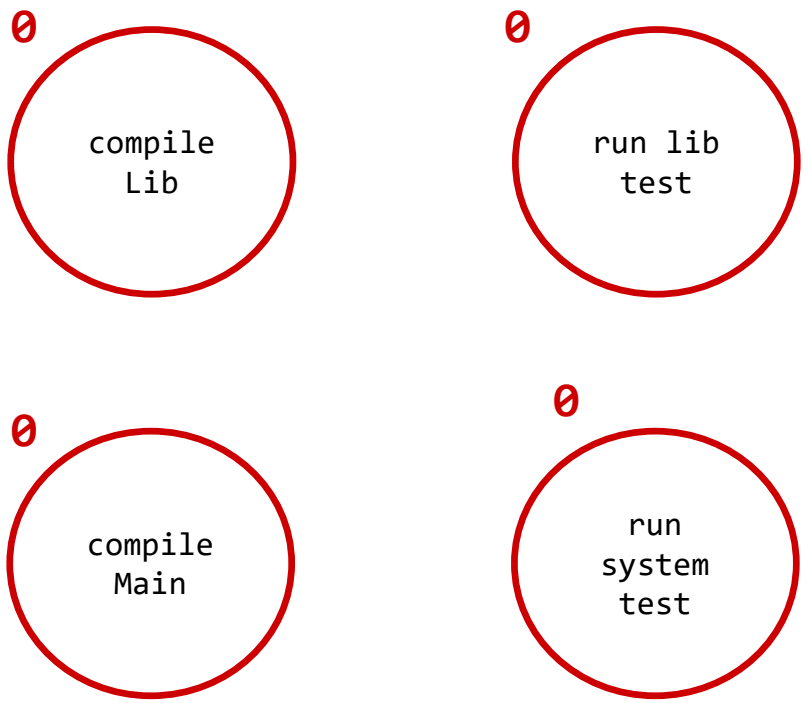
Build systems: topological sort



Build systems: topological sort



Build systems: topological sort



Build systems: topological sort

Valid sorts:

1. compile Lib, run lib test, compile Main, run system test

2. compile Main, compile Lib, run lib test, run system test

3. compile Lib, compile Main, run lib test, run system test

Which is preferable?