Lecture 5: October 18

Interviews, Git, CI/CD

Agenda

- Project Website Requirements
- Project Status Check-in
- Software Engineering Interviews
- Git, PR Reviews
- CI/CD

Team Website Requirements

Team Website

Goal: Act as a central, public location to show off your senior design project

Examples: https://gw-cs-sd-2022.github.io/tutorials/project-teams.html

For Sunday:

- Deploy a basic template for your team website
- Include an about us section with photos & info on team members

As the year progresses, you'll update the website with more information

Setup Instructions: https://pages.github.com/ , https://docs.github.com/articles/configuring-a-publishing-source-for-github-pages/

Project Status Check-in

October Deliverables

- ~~Updated Gantt Chart~~
- ~~Writing: Executive Summary~~
- Writing: Technical Summary (due 10/29)
- ~~Presentation 1: Elevator Pitch~~
- Presentation 2: Technical Design (due)

Current Focus

- Begin technical investigations (services, apis, programming language)
- Flesh out project functionality & requirements
- **Coding should start (scaffolding, ci/cd, prototyping)**

Software Engineering Interviews

What interviewers are testing for

- **Analytical Skills**
  - Could you solve the problem? Did you solve it optimally? For system design, did you structure the problem well & think through tradeoffs?
- **Coding skills**
  - Could you translate your algorithm to code? Was it well-organized? Did you use good style?
- **Technical knowledge**
  - Do you have a strong foundation in the relevant technologies?
- **Experience**
  - Have you made good technical decisions in the past? Have you built interesting projects?
- **Culture fit / Communication skills**
  - Do your values fit with the company and team? Can you communicate your thoughts clearly?

Engineering Interview Process (one possibility)

1. Recruiter Screen
2. Hiring Manager Screen (optional)
3. Coding screen/take home project
4. System Design
5. Team Fit / Behavioral
6. Subject-specific interview / coding problems

Why companies use this process

- **Problem Solving skills & clear communication is valuable**
  - You’ll spend a good amount of time talking through problems with other engineers. Coding challenges are a (sometimes poor) proxy for this
- **Data structures & algorithms are useful**
  - Knowledge of fundamentals is a good proxy for how versed in CS you are
  - They do come up in work, and when they do its important to know the basics

Why companies use this process

- **False negatives are ok, false positives are not**
    - Better for companies to reject good candidates than accept poor fits.

Being good at interviews is a practiced skill, and is only tangentially related to being a good engineer

Recruiter Screen

- **Purpose**: identify if you meet the minimum requirements on paper
- ~15 minutes long
- Interviewer likely knows very little about the technical requirements for the position
- Your resume plays the largest role in this stage
  - Should be easily skimmable
  - Skills & technologies should be clearly listed

Hiring Manager Screen

- **Purpose**: pitch you on the company, confirm you meet minimum technical requirements
- ~30 minutes long
- Interviewer will be technical, acts as a filter before you begin the rest of the interview process
- This interview is more common in smaller companies & startups

Coding Screen / Take Home Project

- **Purpose**:
  - problem solving ability
  - coding ability
  - algorithms fundamentals
- 45 mins - 1 hr long
- Interviewer will be technical

Coding Screen: Tips

- Use python – it saves you a ton of boilerplate
- Communication:
    - Talk out your thought process, the worst thing is a silent candidate
    - Ask clarifying questions to understand the problem
    - If you don’t know the syntax, ask your interviewer
    - If you have a brute force solution, explain it before moving to a more optimal approach
- Coding:
    - Walk through examples & define edge cases before writing code
    - Start with pseudocode (or just use python)
    - Start with a brute force solution
    - Walk through your code with examples
    - Practice algorithms & data structures
- Resources
    - Cracking the Coding Interview
    - www.pramp.com
    - https://www.structy.net/

System Design

- **Purpose**: Can you break down a large ambiguous problem into manageable pieces
- 45 mins - 1 hr long
- More typical for non-junior candidates
- Evaluates your ability to be a tech lead, not a programmer

Tips

- Ask lots of questions, the problem will be ambiguous
- Take good notes so the interviewer has something to review
- Talk through tradeoffs, there is rarely a “correct” answer

Team fit / behavioral

- **Purpose**: Do you work well with others, do your values align with the company & team
- 45 mins - 1 hr long
- Interviewer is usually someone you will work closely with

Tips

- Be prepared to discuss prior experiences from your resume
- Have a few anecdotes on hand about challenges you faced and how you addressed them

Subject-specific

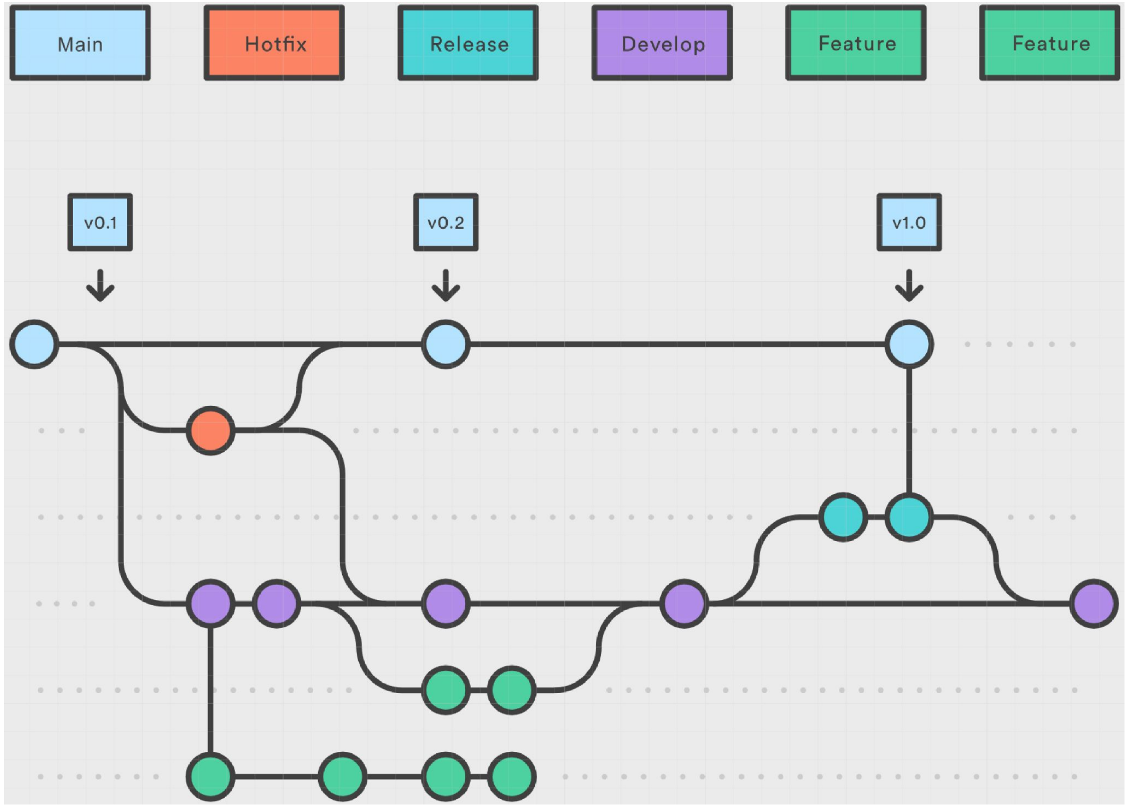
- **Purpose**: Do you have the specialized skills needed for the role
- 45 mins - 1 hr long
- This will depend a lot on the position (ml, security, data science)
- Interviewer is usually a senior engineer
- May ask questions about projects on your resume

Overall Interview Fundamentals

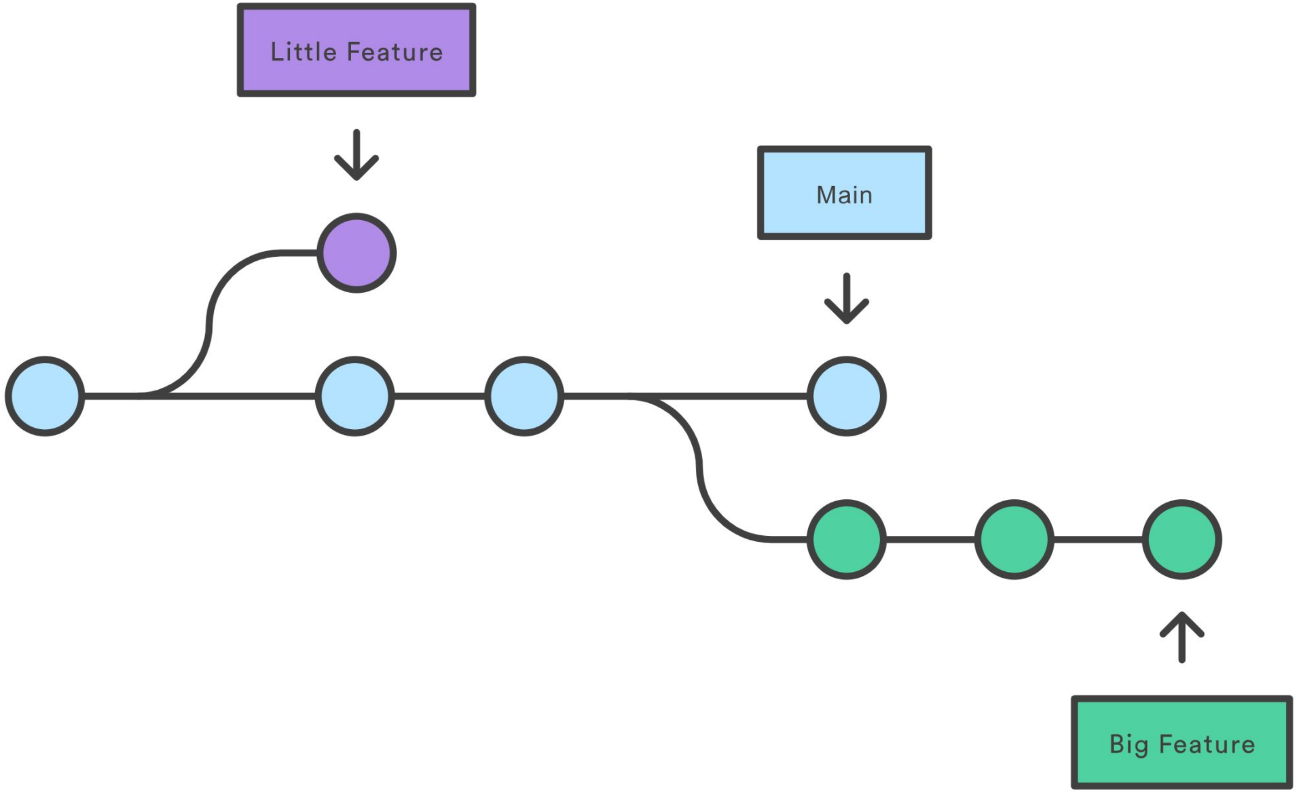
- If you join a zoom call and the interviewer isn’t there, give it 5 minutes before dropping/reaching out to the interviewer
- If you need to reschedule, do so as early as possible
- Dress appropriately
- Turn your camera on
- Ask thoughtful questions
- Send thank you notes
- Once employed, try to interview as soon as you can!

Git

Git Workflow Diagram



Git Workflow Diagram for Senior Design



Developing a feature

```
git checkout main && git pull
git checkout -b js-my-feature
git push -u origin js-my-feature
```

(code changes)

```
git add .
git commit -m “made changes”
git push
```

```
git checkout main && git pull
git checkout js-my-feature
git merge main
```
(may need to resolve merge conflicts)

```
git push
```

(open PR)

Git Resources

- ChatGPT
- https://dangitgit.com/en
- https://www.atlassian.com/git/tutorials/using-branches
- https://code.visualstudio.com/docs/sourcecontrol/overview#_3way-merge-editor
-

PR Reviews

Purpose of Code Reviews

- Ensure that team members are aware of changes to the codebase
- Allow others to verify the correct things are being tested
- Facilitate discussions over implementation design

The overall code health should be improving over time, and developers should make progress on their tasks

Reviewers should favor approving PRs once its in a state where it improves code health, even if the PR isn’t perfect

Authoring a Pull Request

- A single PR should represent a single piece of functionality
- Multiple PRs with small changes is better than one PR with lots of changes
- The description should include **what** changed and **why** the change is necessary
- Add pr comments to code changes to help reviewers navigate the diff
- [Optional] link PR to trello/jira ticket
- If the PR is large or complicated, meet with the reviewers to discuss

Reviewing a Pull Request

Goal: Ensure the changes are positive, even if they aren’t perfect

- **Mountain**: feedback that blocks all related work and requires immediate action
- **Boulder:** feedback that blocks the work from being approved, but doesn’t require immediate action
- **Pebble:** feedback that does not block the PR, but requires future action
- **Sand**: feedback that is not blocking, but should be considered if multiple team members concur.
- **Dust/nit:** feedback that is more a suggestion and not required

https://www.netlify.com/blog/2020/03/05/feedback-ladders-how-we-encode-code-reviews-at-netlify/

Code Reviews for Senior Design

- Team members should not push directly to main
- Team members should try to review each other’s code
- Mentors can provide code feedback if requested, but should not be reviewing all code changes
- PRs do not need to be blocked by approvals

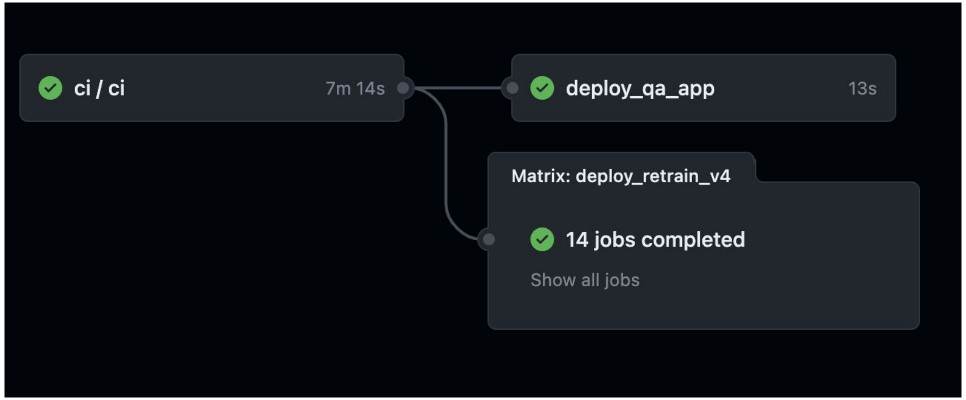
CI/CD

Continuous Integration & Deployment

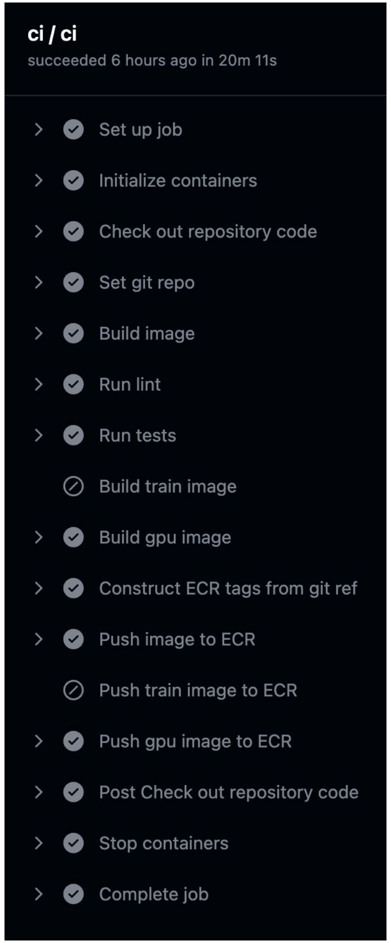
- Continuous Integration is a practice that involves frequently and automatically integrating code changes into a shared repository. The core idea is to detect and address integration issues early in the development process.
  - Unit tests, integration tests, linting. Blocks merging bad code. Frees up developers from manually testing

- Continuous Deployment is an extension to CI that automates the deployment process. It means every code change that passes CI tests is automatically deployed without manual intervention.
  - Builds artifacts, deploys to staging and/or prod environments

Example CI/CD Pipeline



- Run the CI step on every push
    - Gate merges on CI step
- Run the deploy step on every push to main
    - Gate deploy step on CI step



ci / ci
succeeded 6 hours ago in 20m 11s

- ✓ Set up job
- ✓ Initialize containers
- ✓ Check out repository code
- ✓ Set git repo
- ✓ Build image
- ✓ Run lint
- ✓ Run tests
- ⊘ Build train image
- ✓ Build gpu image
- ✓ Construct ECR tags from git ref
- ✓ Push image to ECR
- ⊘ Push train image to ECR
- ✓ Push gpu image to ECR
- ✓ Post Check out repository code
- ✓ Stop containers
- ✓ Complete job

CI/CD Tools

- Circle CI, Travis, Jenkins, Argo, Codefresh, Spinnaker
- Github Actions
    - Free!
    - Easy to configure as part of your github repo

Example Github Action Pipeline

CI/CD for Senior Design

- This is not required, but highly recommended
- Use github actions for CI/CD execution
- Recommended CI steps (on every push):
  - Lint code
  - Run tests
  - Build artifacts
- Recommended CD steps (on merges to main or manual trigger):
  - Build artifacts
  - Deploy changes

Reminders

Ongoing Work

- Create & update October trello boards (these are graded!)
- Post weekly standup updates to team slack channels (these are graded!)

Deliverables & Due Dates

- Team Website due 10/22