Intro to Git

FIRST, AN INTRO TO YOU ALL

Answer a few questions on the slido

Today's presentation

- will highlight patterns and basics of using git
- will give you tools to keep learning and using git
- will not teach you all of git

I AM NOT YOUR PROF

so please ask all your questions

even if they feel stupid

they're not stupid

you're not stupid

git is just hard

Why use Git?

Used by 96.7% of professional developers[1]

» version control

» collaboration

[1] Stack Overflow 2022 Developer Survey

Why use version control?

Track entire story of project

- no more "logo.jpg", "logo-again.jpg", "logo-for-real-this-time.jpg"
- no more huge blocks of old code stored in comments
- can check out any past version of project

WHAT IS GIT?

- distributed version control system
- software tool that runs on your computer
- unfriendly
  - hard to learn
  - lots of jargon
  - no built-in ui

KEY GIT TERMS

Every project has a git repo which tracks the history

- just a folder in the project
- copy of repo exists on each developer's machine

Key Git Terms

History stored as series of commits

» snapshot of project at a specific moment

» "permanent save"

» includes message, author, date/time, & changes

PLEASE NOTE:

GIT ≠ GITHUB

Git is an open source software tool that stores data in repos

GitHub is a website where repos are stored and shared

A git repo is uploaded to GitHub like a document is uploaded to Google Drive

What is GitHub?

- cloud-based git repo hosting service
- holds at least 200 million repos
  - open source projects
- owned by Microsoft since 2018
- friendlier than git but not an equivalent

PRACTICE:
USING GITHUB

- make an account
- make a repo
- make a commit

Collaboration using Git

Projects are not a linear progress

- working on multiple changes at once
  - multiple goals under construction
  - experiment with different ideas
- multiple people working at once
  - want to work freely without worrying about others

What is a branch?

- version of a repository which diverges from the main copy
- each branch is associated with a chain of commits
- branches form "tree"
  - trunk is called `main` branch (good copy of repo)
  - other branches merge back into trunk when ready

Demo:
Branching in Git

Branches: Under the Hood

- commits have unique hash values
- each commit points to previous commit (linked list)
- branch is pointer to a commit
- one commit can have multiple others pointing to it

Merging Branches

- Git will automate as much as possible
- when merge conflict happens, need to resolve manually
- merging into main is often formalized as a pull request
  - "I want to merge this into main"

Practice: Using Branches

- make a branch
- make a commit
- make a pull request

LOCAL GIT

You don't want to edit projects in GitHub, you want to use IDEs (Visual Studio Code, PyCharm, etc)

Every person working on a project has a local copy of the repo, in addition to the shared remote copy (stored on GitHub, GitLabs, etc)

Periodically sync local copy with remote copy ("origin") for backup & collaboration

Local Git Commands

- init to create repo on local machine
- clone to get remote repo on local machine
- fetch to update knowledge of remote repo
- pull to get new changes from remote repo
- push to send new changes to remote repo

LOCAL DEVELOPMENT

- project is edited in workspace (IDE)
  - check out branch to update workspace
  - HEAD pointer marks checked out branch
- when ready to commit, stage changes
  - next commit contains changes in staging area
  - stash changes that aren't ready to commit

Local Git Tools

- command line git
  - easy to install, hard to use
- built-in to IDEs
  - Visual Studio Code, PyCharm, IntelliJ, etc
- third-party application
  - Git Desktop, Tower, etc

DEMO:
LOCAL GIT