Introduction to Version Control with Git
Part 2

Ben Winjum
OARC
November 20, 2024

Git + collaboration (using GitHub as example)

- Git already includes the machinery to move work between two repositories
- In practice, a central repository is usually used as a master copy
  - GitHub, BitBucket, GitLab…
- These hosting services also offer tools to facilitate collaboration
  - Web-based – anyone with an internet connection can view the repo
  - Wikis, task management, bug tracking, feature requests

Git + collaboration (using GitHub as example)

- Let’s make a new repository on GitHub

Git + collaboration (using GitHub as example)

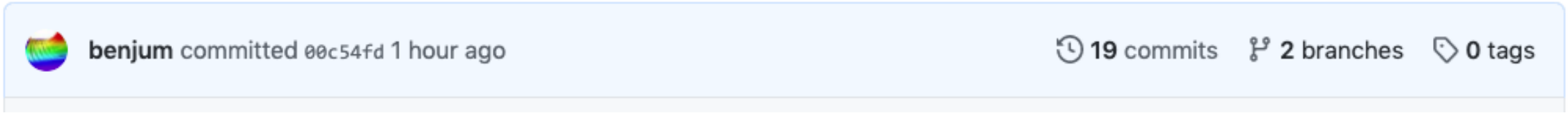
- Let’s make a new repository on GitHub
- After making it, we need to link our local repository with the remote repository
  - We’ll use HTTPS – you are encouraged to investigate configuring the SSH option later
  - `git remote add origin https://github.com/username/reponame.git`

Git + collaboration (using GitHub as example)

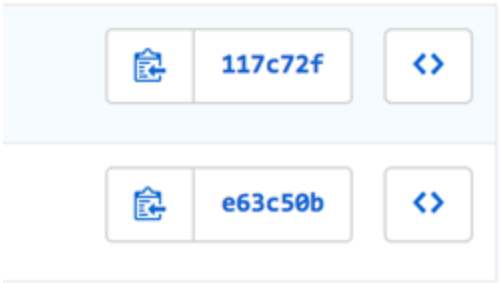
- Exchanging changes between repositories
  - `git push`
  - `git push origin main`
  - `git push –u origin main`
  - `git pull`
  - `git pull origin main`

Review

- Look at your new repository on GitHub and find the bar that looks similar to this:



- Click on commits, and then find three buttons per line that look like:



- What do these buttons do when you click them? And how can you do something similar in the terminal?

Collaborative workflow

- The basic collaborative workflow:
  - update your local repo with `git pull origin main`
  - make your changes and stage them with `git add`
  - commit your changes with `git commit -m`
  - upload the changes to GitHub with `git push origin main`
- It is better to make many commits with smaller changes rather than one massive commit with lots of changes
  - Small commits are easier to read and review

GitHub exercise

- Since we're all remote and it's tricky to coordinate efforts…. Let's pretend we are each two people

- Clone your own GitHub repository into a new directory

- Part1-of-you: Make a couple changes to the local repo, and push changes back to GitHub
- Part2-of-you: Pull the changes that Part1-of-you made into your local copy of the repository

- Switch roles and repeat

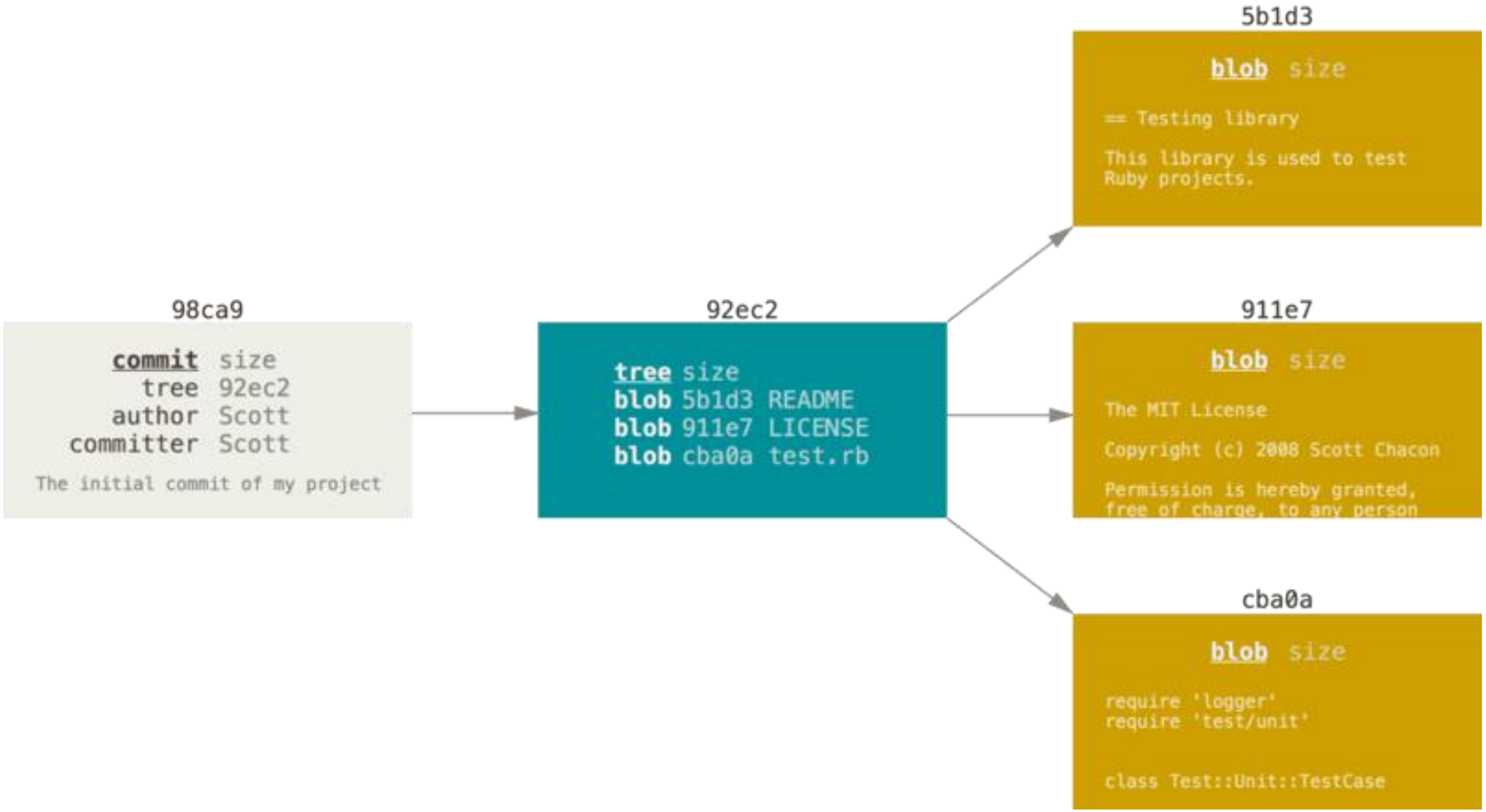
Dealing with conflicts

- You will inevitably encounter scenarios in which you make local commits and try to push them to a remote repository, only to discover that a collaborator has updated the repository too
- Git will recognize potential conflicts and refuse to accept a push request
    - Solution: pull changes from the remote repo, merge them locally, and push again

Branching

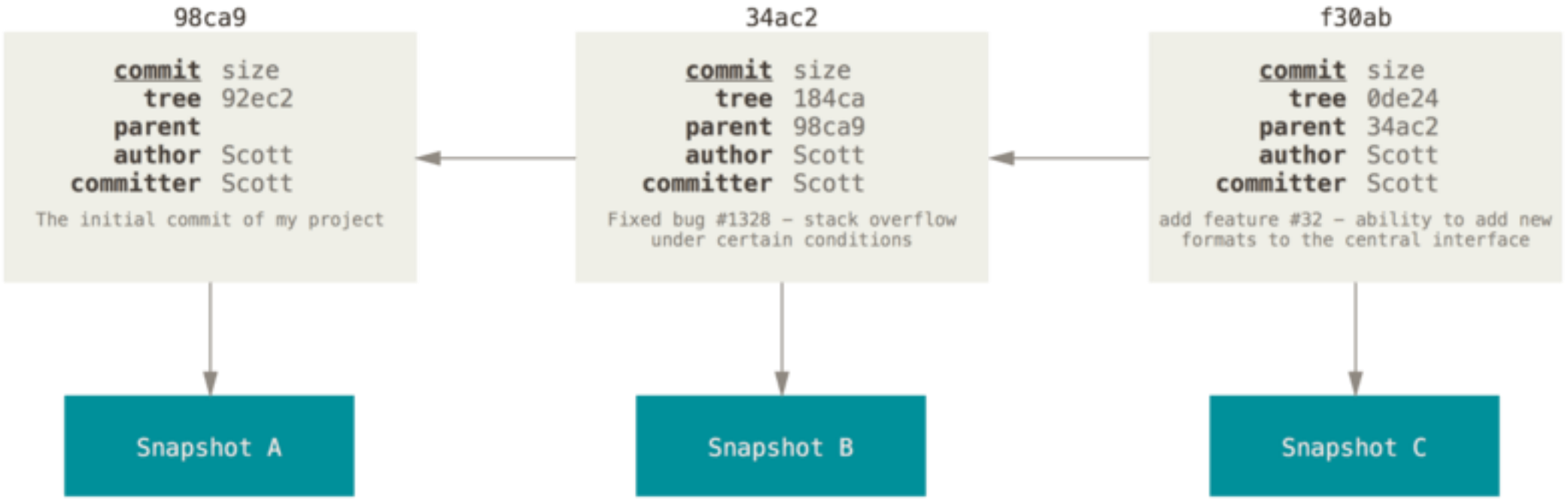
- Branching is one of the key features of git
- Git has a very light-weight method for developers to work simultaneously and separately on a project (in separate branches) and then merge their work together when they are ready
- To understand this, it helps visualize some of the conceptual structure behind git

Files and the commit history



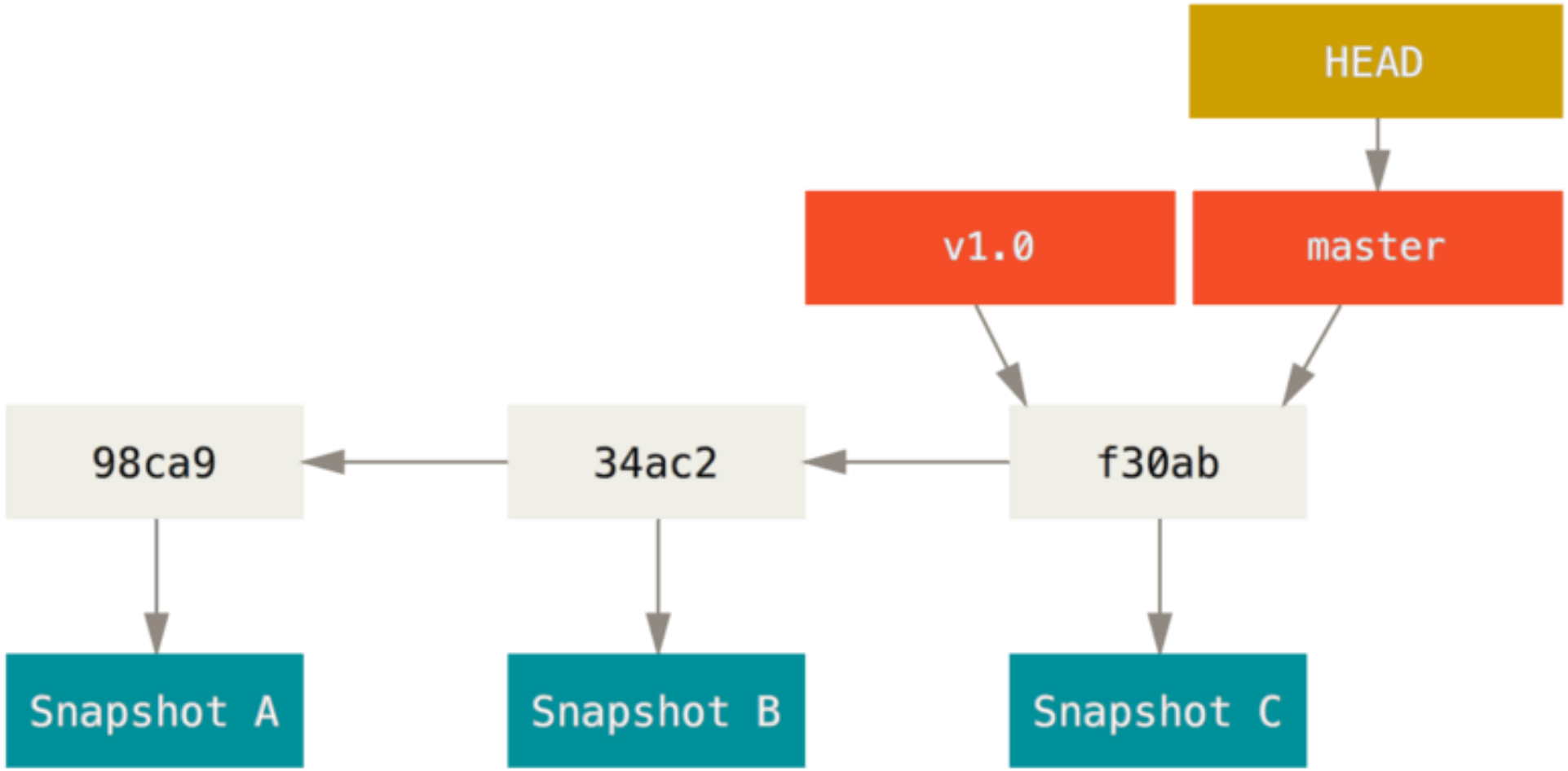
Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

Files and the commit history



Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

Pointers to commits



Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

Branching commands

- Create a new branch:
  - `git branch <branchname>`
- Switch to a branch:
  - `git checkout <branchname>`
- Create a new branch and switch to it at the same time:
  - `git branch -b <branchname>`
- Delete a branch:
  - `git branch -d <branchname>`

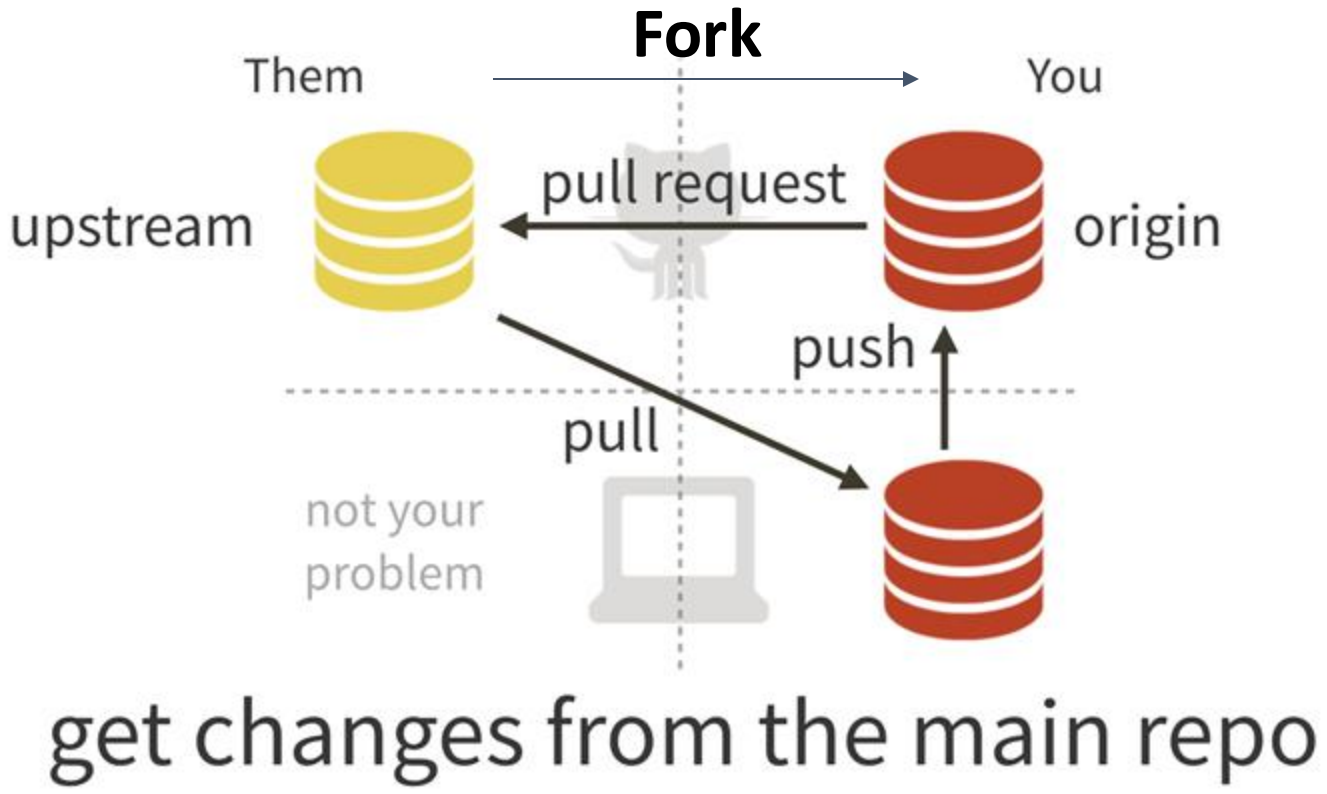
The tool for visualizing git actions can be found at:

http://git-school.github.io/visualizing-git/

Branching example

- Create a new file for elementary math examples (with an error)
  - Commit to main
- Add features to the math file
  - Create a branch for this development path
- Correct the original error
  - Create a hotfix branch, fix the error, merge into main, delete hotfix branch
- Finish editing
  - Try merging math into main --> pull request illustration

Forks, Pushes, and Pull Requests



https://happygitwithr.com/upstream-changes.html

Some final thoughts: If you run into lots of conflicts

- Conflict resolution costs time and effort and can introduce errors if conflicts are not resolved correctly. If you find yourself resolving a lot of conflicts, consider these technical approaches:
  - Pull from upstream more frequently, especially before starting new work
  - Use topic branches to segregate work, merging to main when complete
  - Make smaller more atomic commits
  - Where logically appropriate, break large files into smaller ones so that it is less likely that two authors will alter the same file simultaneously
- Conflicts can also be minimized with project management strategies:
  - Clarify who is responsible for what areas with your collaborators
  - Discuss what order tasks should be carried out in with your collaborators so that tasks expected to change the same lines won't be worked on simultaneously
  - If the conflicts are stylistic churn (e.g. tabs vs. spaces), establish a project convention that is governing and use code style tools to enforce, if necessary

Any Future Questions:
Please email me!

bwinjum@oarc.ucla.edu