A team git process

- teams of 3-5 students, each team working on their own project
- each team has shared central git repo with two core branches: master (the “production” version) and dev (development)

  [based on a team git repo set up by our tech wizard,

  with the permission of the course instructor]

- instructor will evaluate based on what's in those two branches on the team's shared repo
- need a process to coordinate pulling/pushing of updates (don't want one member's push to wipe out changes made by others)

The initial setup

- as with our other labs using git:
  - instructor sets up a project repo on csci git server
  - team forks that to set up their shared repo on git server
  - team members clone, pull, push as discussed below
- strongly recommend each team:
  - pick one trusted team member (their git lead) to handle the initial setup and all subsequent access to the master branch
  - other team members only pull from/push to the dev branch, following process discussed in next few slides

Setup steps by team git lead

- each team will be given designated team identifier, e.g. *teama* below
- team's git lead forks, clones, sets up dev branch:
  - cd csci265
  - ssh csci fork csci265/project csci265/*teama*/project
  - git clone csci:csci265/teama/project
    - edit master readme, do git add/commit
  - git branch dev
  - git checkout dev
    - edit dev readme, do git add/commit
  - git checkout master
  - git push origin --all
- instructor can pull team's master and dev from shared repo to check setup is ok

Team member's work cycle

- each team member likely to be working on different components or features
- will be pushing their changes to dev, but also need to pull regularly from dev to get changes pushed by other team members
- each time we pull others' changes from dev there may be conflicts with our own local changes, will need to resolve those
- want to be careful we don't push (known) broken code to shared dev, since that breaks dev content for everyone else too
- potential scenarios where team members can wipe out each other's pushes if not coordinated (see next slide)

Conflicting pushes:

- Scenario: *two different team members pull same code from dev, merge it with their own (different) local versions, both make more changes and push*
- Result: *whoever pushes last “wins”, other's changes may be lost*
- Two steps to avoid this:
  - pull again just before you push: catches changes others just pushed
  - have a fixed location where you announce your intention to push (e.g. a team discord)
    - ALWAYS check and post here before pushing
    - if someone else is in the middle of a push: wait for them to finish, then pull their changes, then do your push

Sample team member use

- clone dev into a local repo to use for the feature or component you're going to be responsible for, pick a branch name for your feature
  - git clone csci:csci265/*teama*/project --branch dev
  - git branch *yourbranchname*
  - git checkout *yourbranchname*
- do your work, add/commit regularly until you're results are safe to share, pull the shared dev in case anyone has pushed since you last pulled
  - git pull origin dev
  - fix any issues then add/commit
- check your team announcements about pushes, and if no further pull is required push your update (next slide)

The push to shared dev ...

- go back to the dev branch
  - git checkout dev
- merge the changes from our branch into our dev (--no-ff helps in the commit record keeping)
  - git merge --no-ff *yourbranchname*
- push back to the shared dev
  - git push origin dev
- done!

Team git lead updating master

- done once the team agrees that the current version of dev is stable (compiles cleanly, passes current test set etc)
- git lead checks out and updates their local dev:
  - git checkout dev
  - git pull origin dev
- git lead checks out their local master, updates from their dev
  - git checkout master
  - git merge --no-ff dev
- git lead pushes updates to the shared repo
  - git push origin --all

Cautionary notes for the git lead

- strongly recommend git lead keeps a separate repo for the master/dev updates
  - this needs to be distinct from any repo(s) they use for their own contributions to the project
- git lead must always always always be sure they're working in the right repo!