

Lecture 07 - From Git to GitHub

.gitignore

- ▶ good for ignoring file system files or data files (privacy and storage space, especially on GitHub)
- ▶ if you previously tracked some files and want to add them to your .gitignore file, you'll have to make a commit with those files absent from the directory and then add those files to the .gitignore file. How would you accomplish this?
- ▶ `git status --ignored` returns files that are currently ignored, but doesn't show hidden files

git checkout

- ▶ useful for reverting files to a previous version: `git checkout <commit hash> <file name(s)>`
- ▶ If you forget to specify a file when you do this, Git will tell you that “You are in ‘detached HEAD’ state.” In this state, you shouldn’t make any changes. You can fix this by reattaching your head using `git checkout master`.
- ▶ One key point, remember to use the hash for the commit before the change we are trying to get rid of, not the hash from the commit that implemented the change we want to get rid of...

branching and merging

```
git status
```

What branch are we on?

branching and merging

```
git status
```

What branch are we on?

Think of a branch as an experimental version of the repo that we may or may not decide to keep.

branching and merging

To create a branch:

```
git branch branch_name
```

```
git checkout branch_name
```

OR

```
git checkout -b branch_name
```

branching and merging

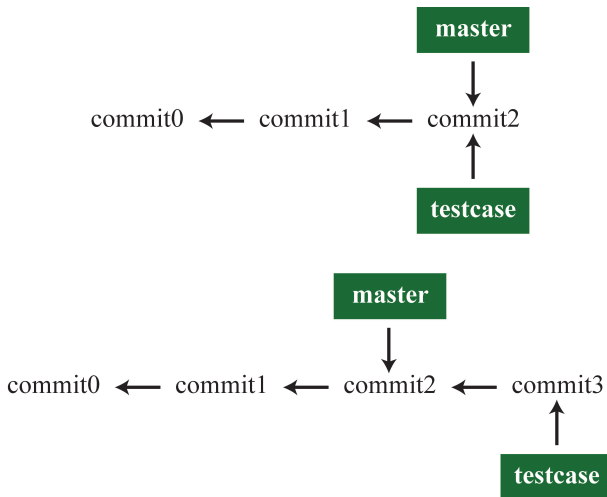
To permanently keep the changes from a branch we merge them back into the master branch:

```
git checkout master
```

```
git merge <branch_name>
```

branching and merging

We can visualize a series of commits and branches as a tree diagram.

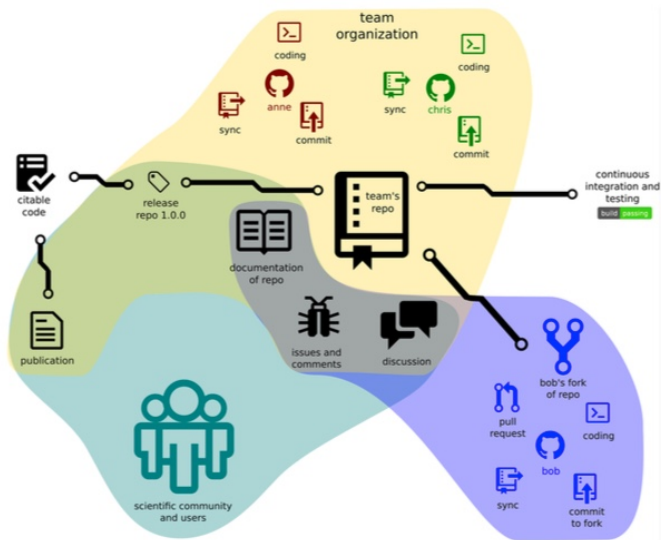


branching and merging

Draw a Git tree diagram at each timepoint of the following scenario.

1. A repo was initiated and four commits were made to it.
2. A branch named "issue001" was created and you make one commit on this branch.
3. Your advisor calls you and says they need you to fix a bug in the code you are developing. So you return to the master branch, create a branch called "quickFix", fix the error and commit a change.
4. You double check the fix works, and then merge the quickFix branch with master.
5. You go back to issue001 and continue working, during which time you make 3 commits. The new code seems to work well and you merge issue001 into master.

What is the connection between Git and GitHub?




GitHub repos



Stuart Jones

joneslabND

[Add a bio](#)

 University of Notre Dame

 sjones20@nd.edu

 <http://www.nd.edu/~sjones20>

Organizations



Overview

Repositories 15

Stars 0

Followers 2

Following 0

Popular repositories

Customize your pinned repositories

ICB_Fall2017

 Shell ★ 1

peaksND

R tool for processing of amplified fragment length polymorphism data generated by ABI sequencers

glmtools

Forked from USGS-R/glmtools

Tools for interacting with the General Lake Model (GLM) in R

 R

methaneEcosystemModel

 R

KF-code

Forked from ctsolomon/KF-code

Kalman Filter analysis of lake C models

LakeMetabolizer

Forked from GLEON/LakeMetabolizer

Collection of Lake Metabolism Functions

Creating a GitHub repo

See SWC - Version Control with Git: Remotes in GitHub

- ▶ Always need to create the repo with GitHub web interface
- ▶ Once created one can do more or less work in Git (local)
 - ▶ create locally and push
 - ▶ create on GitHub and clone

Creating a GitHub repo

The key goal is to have the same content in a local Git repo and in the GitHub repo

- ▶ create locally and push:

```
git remote add origin <url for GitHub repo>
```

```
git push -u origin master
```

- ▶ create on GitHub and clone:

```
git clone <url for GitHub repo>
```

****simple with own GitHub repo, but collaborator needs permission**

push vs. pull; origin

- ▶ talking back and forth with a GitHub repository
- ▶ origin is a nickname for remote repository linked to the local repo

`git push origin master` - sends local content to the associated GitHub repo

`git pull origin master` - brings GitHub content to the local Git repo

- ▶ how does Git know what GitHub repo is associated (i.e. what origin is)?

conflicts

- ▶ can happen when alone, but very likely when collaborating on same repo
- ▶ occurs when parallel changes have been made to the same lines of code
- ▶ shows sophistication of Git and GitHub
- ▶ these conflicts must be manually resolved before successfully pushing code, pulling code, or merging branches

issues

- ▶ built in structure for development and fixing bugs
- ▶ on GitHub repo page

forking

- ▶ creating your own version of someone else's GitHub repo
- ▶ doesn't require permission
- ▶ initial forking occurs on GitHub web interface
- ▶ need to link your local Git repository to original GitHub repo:

after forking and pulling the repo:

```
git remote add upstream url_for_repo_forked_from
```

```
git fetch upstream
```

pull requests

- ▶ when you have made changes to your GitHub repo that you think the original owner would like
- ▶ must be approved by original repo owner/user

function review

git checkout

git merge

git pull

git push

git clone

git remote

git fetch