GitHub Tutorial

Aswathi Balagopal V., Jeff Weber

IceCube Summer School, 2023

What is version control?

Tracks and manages changes that you make to something

- *Reversibility*: the ability to back up to a previous state if you discover that some modification you did was a mistake or a bad idea.
- *Concurrency*: the ability to have many people modifying the same collection of files knowing that conflicting modifications can be detected and resolved.
- *History*: the ability to attach historical data to your data, such as explanatory comments about the intention behind each change. Even for a programmer working solo, change histories are an important aid to memory; for a multi-person project, they are a vitally important form of communication among developers.

About git

- Created for linux kernel development
- Easy to use but powerful version control system
- Designed as distributed system. Manage your project on a server and work on local versions
- Keep track of different versions
- Split off different development branches and then combine them back together
- Makes collaborative work easy
- Widely used in software development

Github

- Web service to host remote git repositories
- Public and private repositories
- Forking and pull requests
- Bugtracking, feature requests and more
- Home to many projects
- There are other services like Bitbucket or GitLab

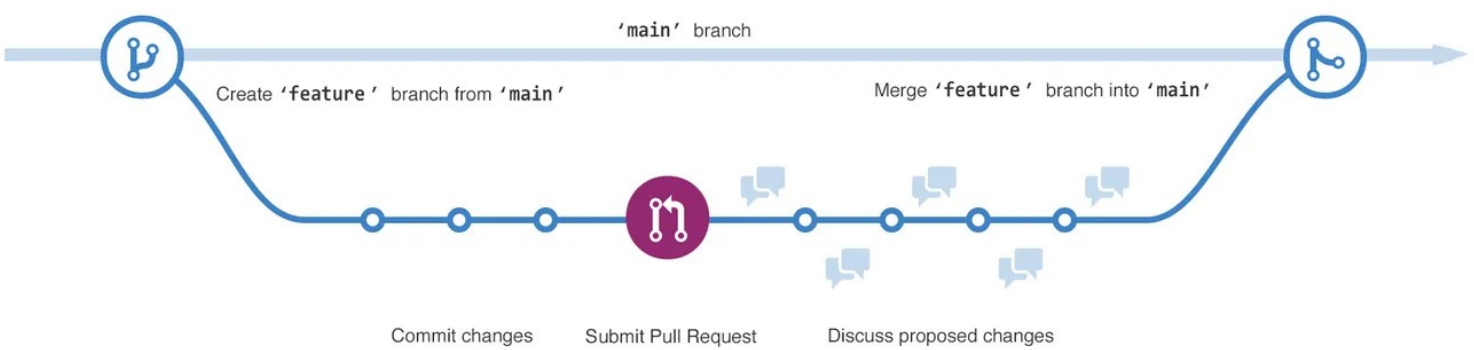
Some terms

Repositories

- Top-level directory of files and directories that is managed by a version control system
- Often stored on a webserver
- Developers contribute to it

Branches

- Repositories can contain parallel versions of themselves
- One main branch and several development branches
- Once developed, can merge to main



Some terms

Commit

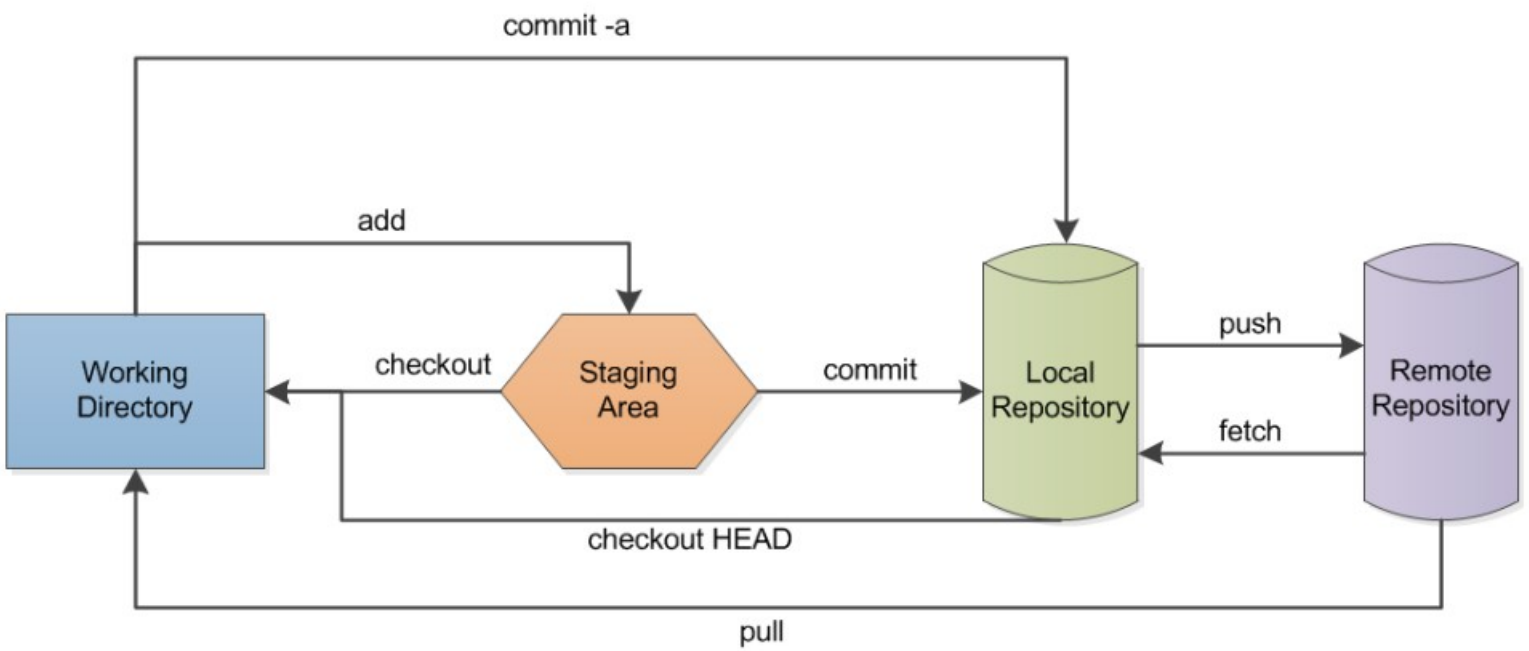
- Set of file modifications grouped under the same user-provided descriptive comment
- Provides a snapshot in time of the entire repository

Pull request

- Pull in your contribution (in your branch) and merge them into the main branch
- Better than a direct commit to the main branch to avoid mistakes

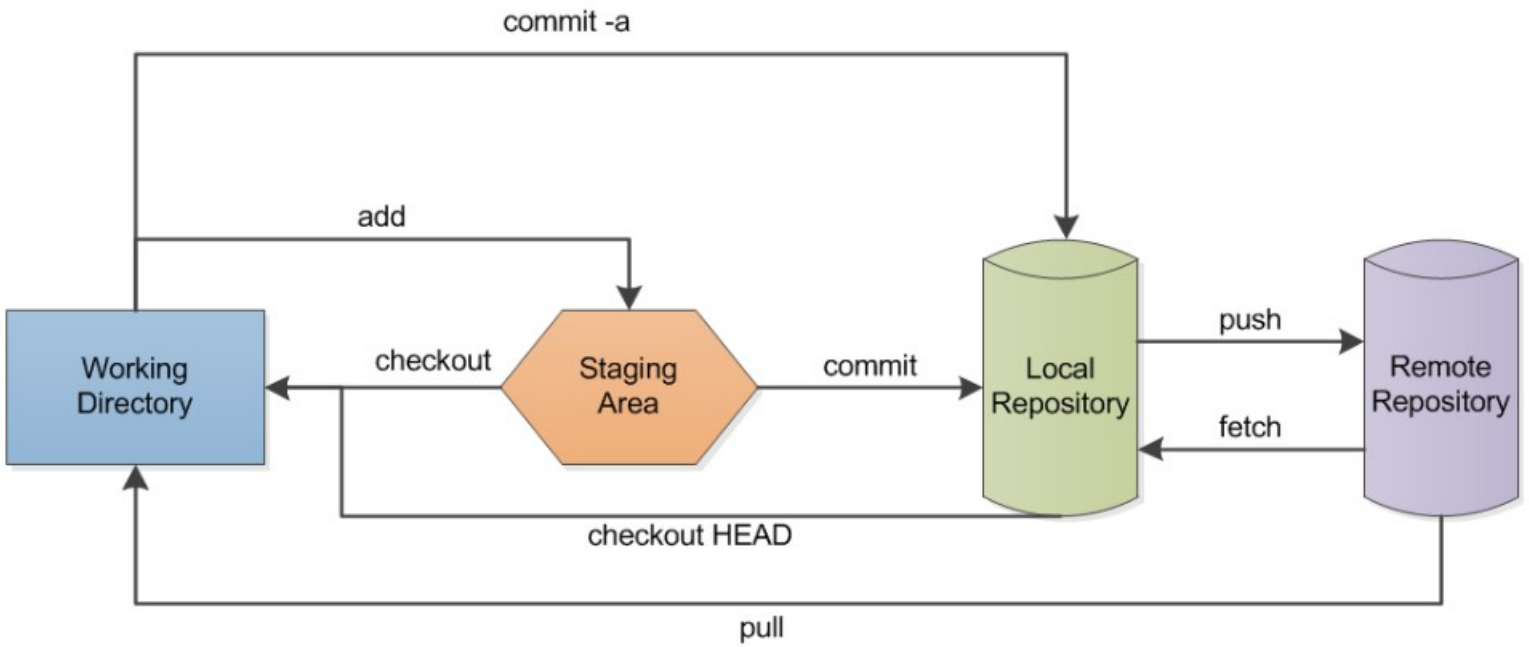
Cloning respositories

- To work with a repository you create a local copy of a remote repository
- Contains the project files and the git repository information
- Set the project files to a specific branch/version by checking it out



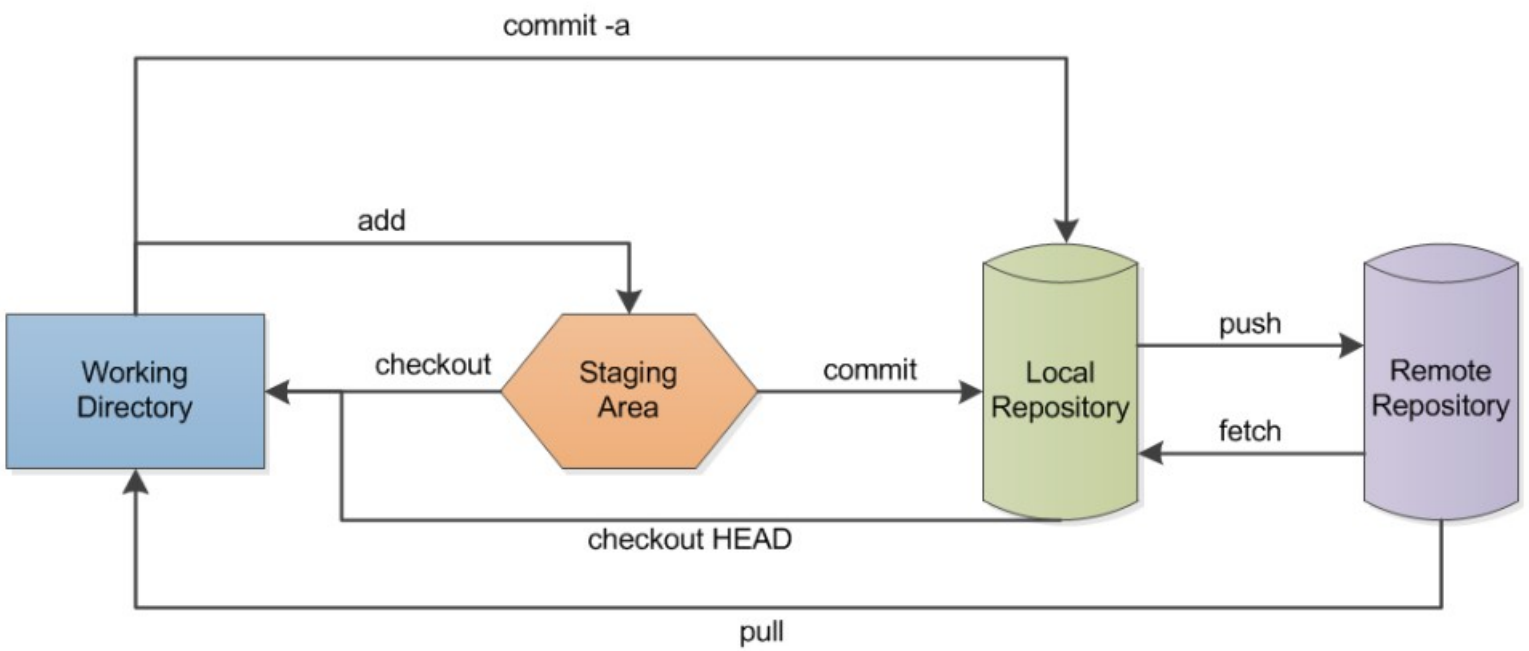
Committing changes

- If you made changes to your local files you can save them by
  - Adding them to the staging area
  - Committing them to your local repository
  - Writing a comment indicating the changes



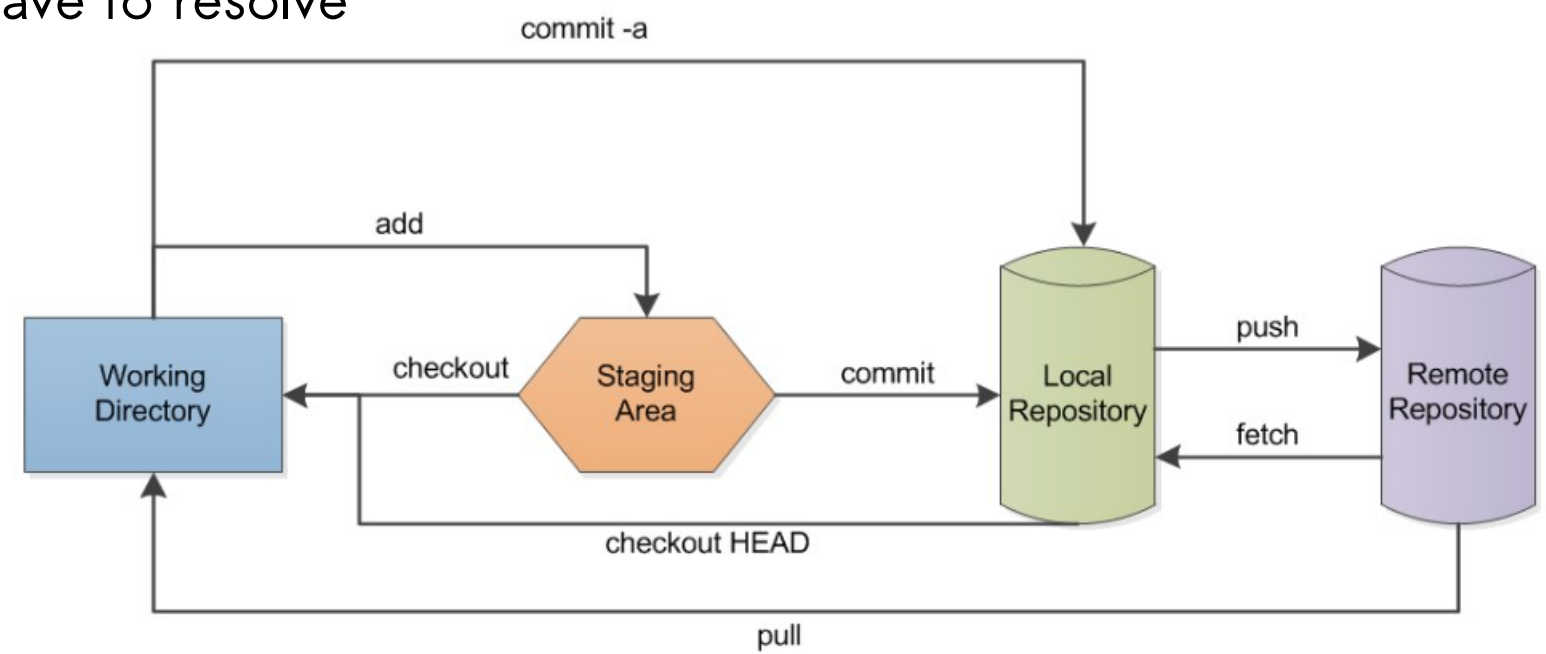
Pushing changes

- Upload your committed changes by pushing them to the upstream repository (if you have access)
- Most collaborative work use pull requests



Updating your clone

- Update your clone by fetching the latest changes from the upstream
- Update your branch by merging the fetched changes into your branch
- Or do both by invoking the pull command
- git tries to merge files automatically
- Sometimes this is not possible and you will get a conflict warning which you then have to resolve



Forks

- Fork is a new repository that shares code and visibility settings with the original “upstream” repository
- You want to contribute to a repository you do not own (e.g. some cool project)
- Create a remote copy (fork)
- Develop your fork as usual
- Send a pull request to the original repository to request merging of your changes

Setting up

- Create a Github account (https://github.com/)
  - These are free to get from github.com
  - Request you include your full name in your GitHub account profile (J. Smith is OK)
  - Include your current institution in your account profile
- Ask one of the MANY GitHub icecube ORG admins for an invitation
- Asking in #software or #icecube-it on Slack usually gets an invite created in a few minutes
- Take a look at the IceCube Github Guide

Setting up Locally

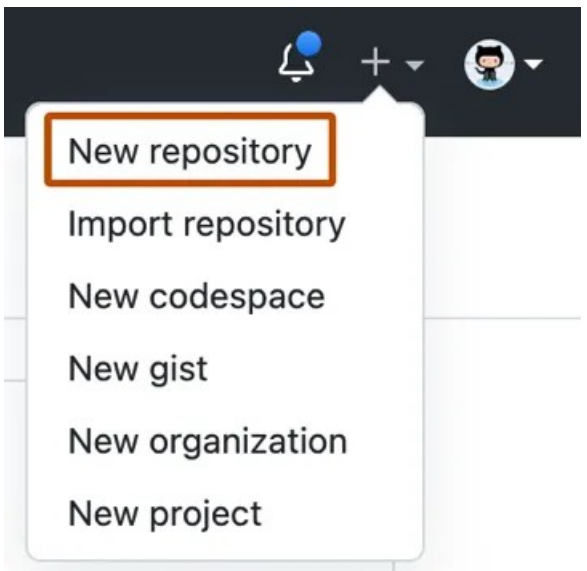
- You can install github on your computer
- Linux
  - sudo apt-get install git
- Mac
  - sudo port install git
- Also preinstalled on icecube cobalts.
- Set your name and email for your command line client
  - git config --global user.name "First Last"
  - git config --global user.email "user@icecube.wisc.edu"
- Make sure this account is associated with your GitHub account (can have many!)

Setting up for IceCube

- Use 2-Factor authentication with GitHub
  - Plenty of options are available. (SMS, Authenticator apps, tokens
  - GitHub will require this by end of 2023. Also requirement in IceCube
- Add and use your ssh keys
  - Nearly impossible to push commits with git on the command line otherwise
  - Follow instructions here: generating ssh keys

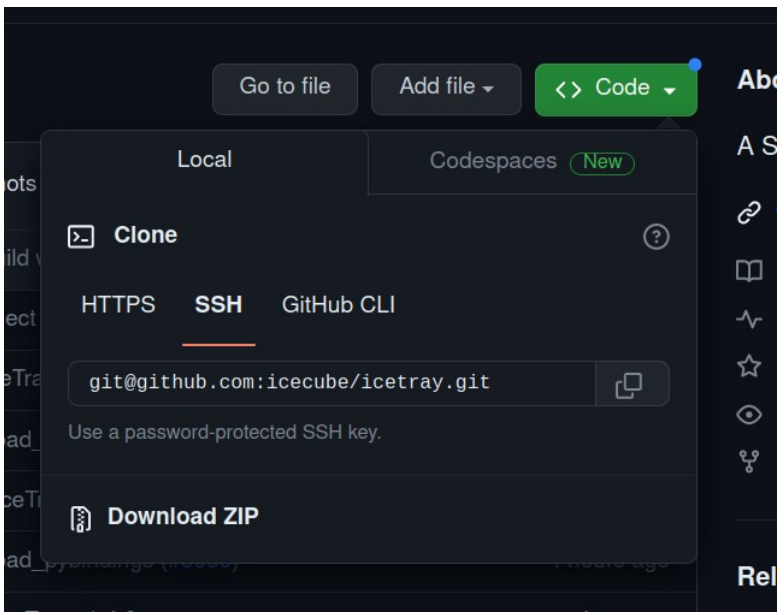
Creating a repository

- Login to github
- Click on + to create a “new repository”
- Give it a name, a description, choose “public/private”
- Also add a README file



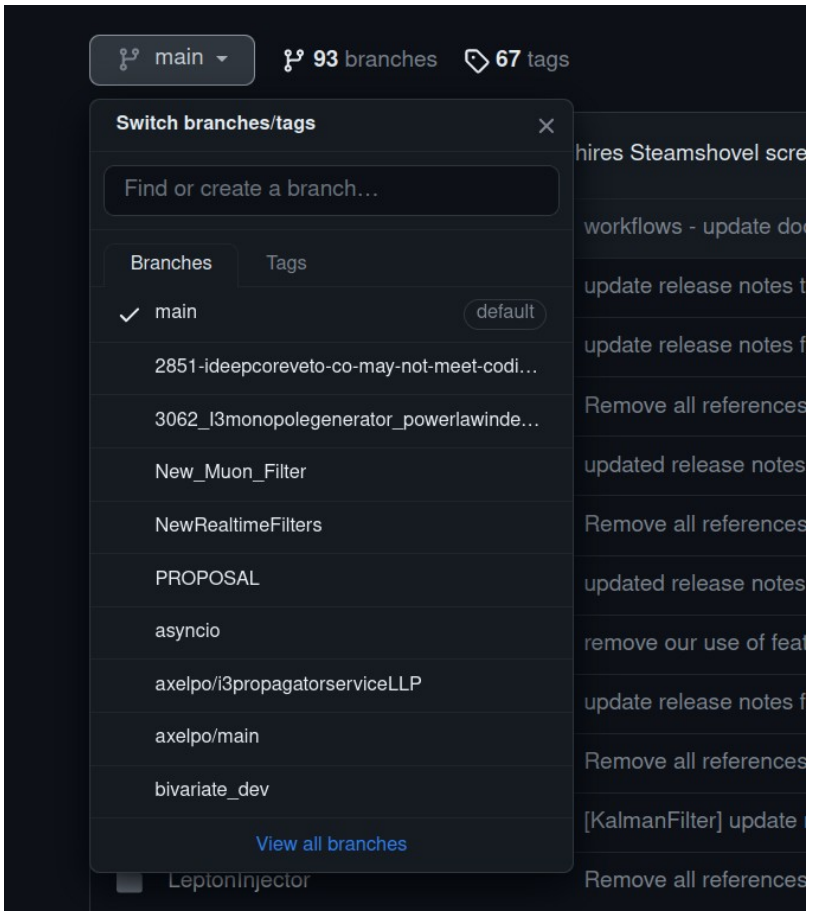
Cloning

- Go to the local directory where you want the clone
  - git clone <url>
- The url can be found from the repository page on github
- You can also clone a repository from another computer via ssh
- Try cloning icetray! https://github.com/icecube/icetray



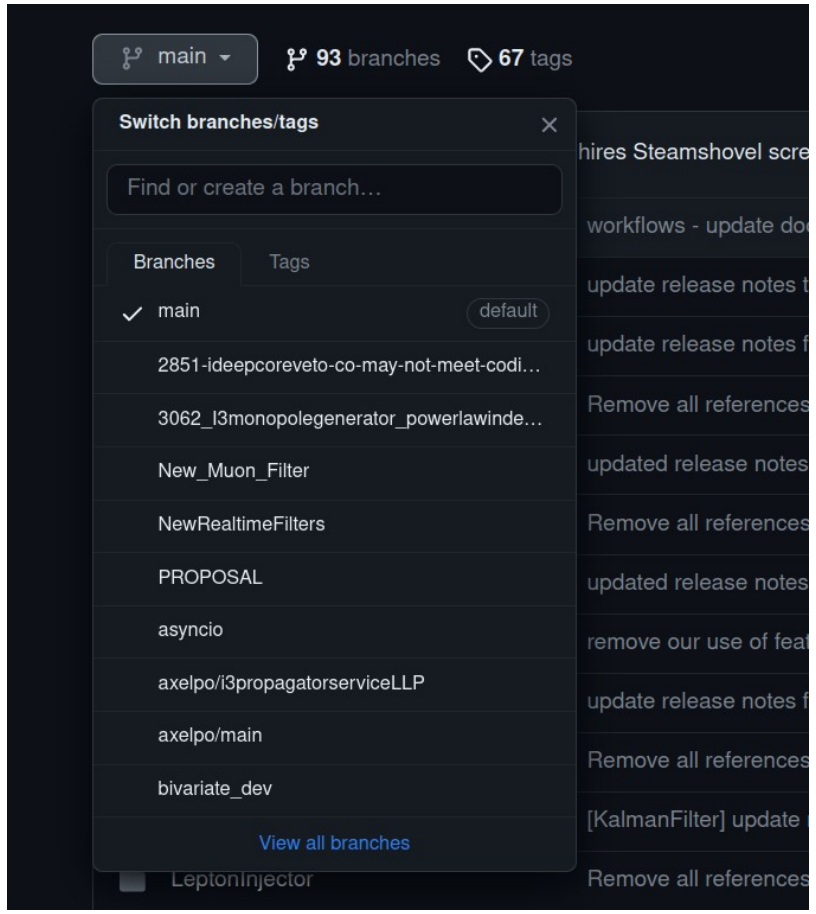
Working with branches

- A new repository starts with a master branch
- If you clone a repository you also start out in the master branch
- You can view branches in github interface or local with
  - git branch – a



Creating branches

- You can create new branches via github interface
- Using the dropdown menu switch to the branch from which you want to start
- To update your local repo do
  - git fetch



Creating branches

- You can also create new branches via the command line
  - git branch <branchname>
  - git checkout <branchname>
- FYI: these two commands can be combined with
  - git checkout -b <branchname>
- You can switch between branches in your staginf area with
  - git checkout <branchname>

Exploring branches

- You can look at the branches and how they are connected using the github interface

Updating repositories

- Sometimes the remote repo changes
- We can simulate this using the github interface
  - Switch to a branch
  - Edit the Readme file or create a new one
  - commit your changes
- To update your clone do
  - git fetch
- To update the local branch, change to the branch then do
  - git checkout <branchname>
  - git merge origin/<branchname>
- You can combine the fetch and merge commands by
  - git checkout <branchname>
  - git pull

Merging

- You can also merge different branches into each other
- We did this already on the last slide with the "origin/<branchname>" merge
  - git checkout <branchname1>
- And then
  - git merge <branchname2>

Commits

- Committing with the github interface is not the usual case
- Usually you work on your computer and want to commit the changes to the remote repository
- To do this first switch to a branch you want to work on
- Do all your developing
- Today you could edit the Readme file and create another new file
- To see the changes of local files with respect to the last commit you can do
  - git status
- It will list new and changed files

Commits

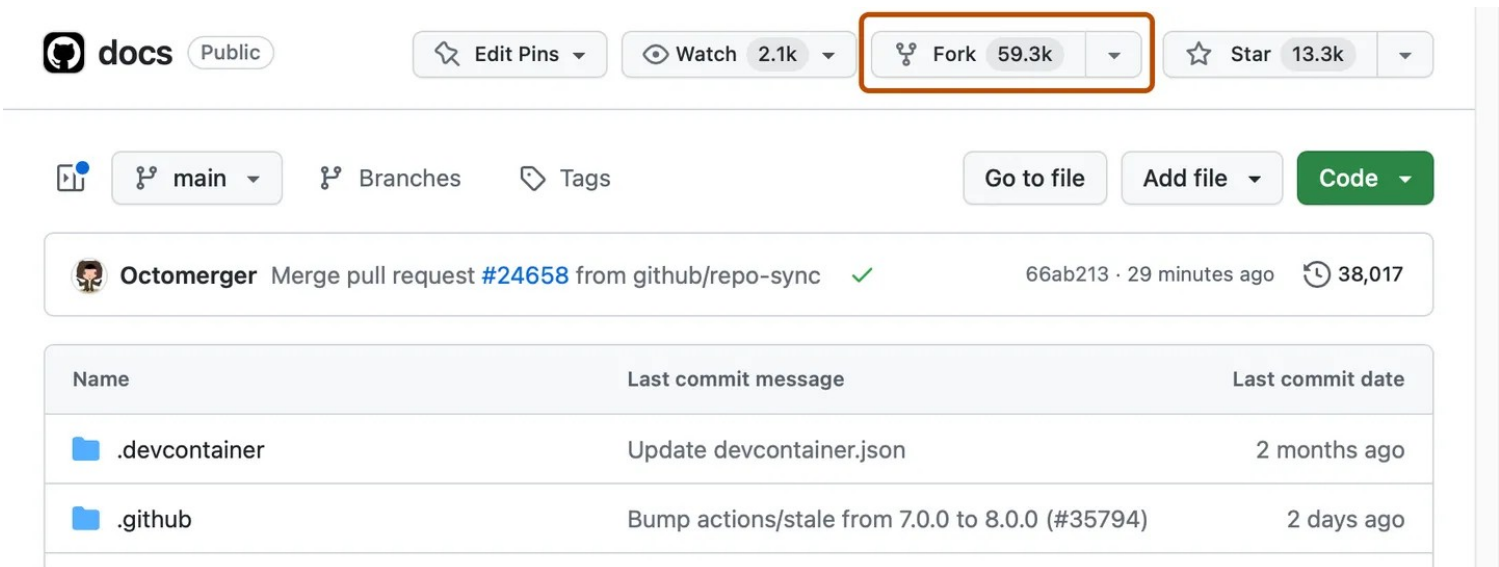
- Tell git which files to commit to the repository
  - git add <filename>
- You can also remove added files from this list
  - git reset HEAD <filename>
- To commit the changes to your local repository do
  - git commit -m “some comment”
- And to upload the files to the remote
  - git push
- Often the remote changes between your last pull and the push. So you can pull before pushing
- You can push to a brach on upstream too
  - git push --set-upstream origin <branchname>

Conflicts

- Sometimes git can not merge files because of conflicting changes
- We can simulate this
  - create a new branch from your current one, but do not switch branches
  - edit the Readme file in the current branch and commit the changes
  - change to the new branch
  - edit the Readme in the same line (with some other edit) and also commit the changes
  - Try to merge the first branch into the second
- You will get an error indicating conflicts in the file
- Edit the file to resolve the conflict (conflicting lines are indicated by «« and
  - »»)
- Add the resolved file to the staging area and commit the resolution

Forking

- If you want to fork a repository in github, you can use the fork button
- You would then clone your fork and work as usual



Syncing

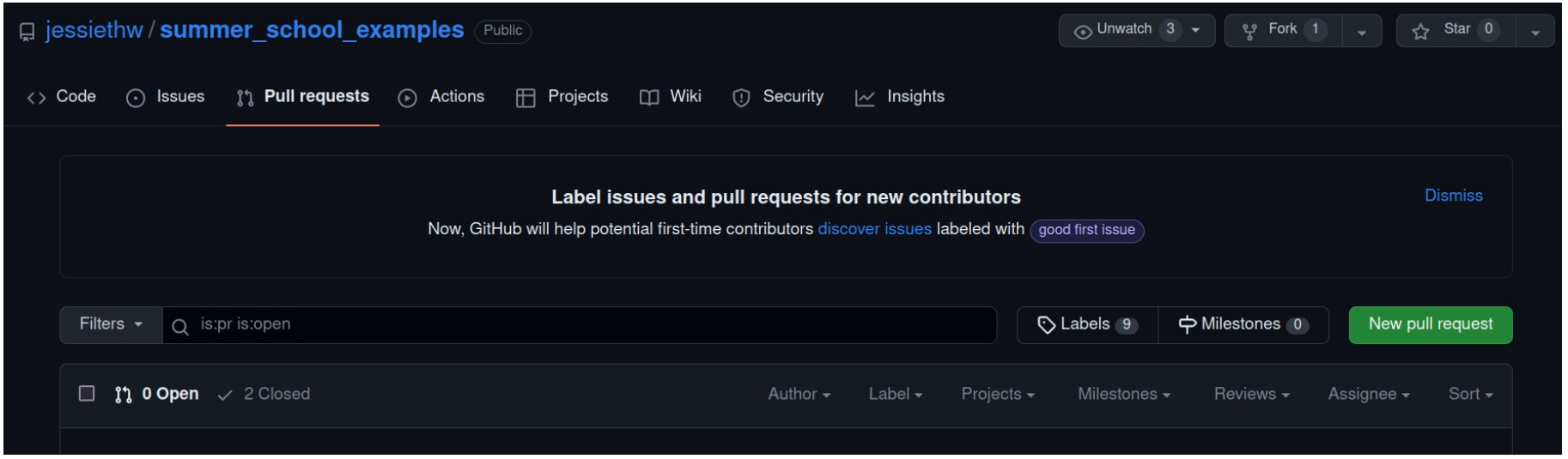
- To keep your fork up-to-date with respect to the original you have to set the original as upstream
  - git remote -v
- Add the original as additional upstream
  - git remote add upstream <originalurl>
  - git remote -v
- To fetch the original updates do
  - git fetch upstream
- Merge the original branch in your upstream branch
  - git merge upstream/<branchname>
- All changes are committed to your fork

Pull requests

- To merge your fork back with the original you have to send a pull request via the "New pull request" button of your fork. Here you should describe your changes
- The owner of the original will see this, can discuss the changes with you and ultimately accept your request.

Pull requests

- Checkout the summer school repository (https://github.com/jessiethw/summer_school_examples)
- Create a branch
- Add a file in the branch, make changes
- Commit changes in the branch to upstream
- Push to your branch
- Create a pull request to merge with the main



Summary

- Git is a powerful tool for collaborative software design.
- Many projects are hosted on github
- You should now be in a position to manage own repositories as well as contribute to other ones
- Many helpful resources on the internet
- IceCube Github guide