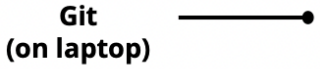
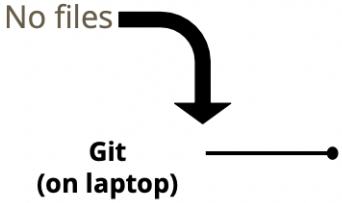
CAS CS 506
Lec 01

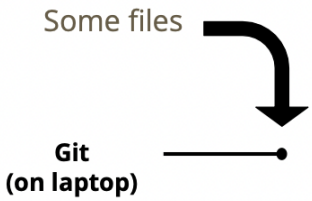
Git

1. Git
    a. A Tool to help us manage the timeline(s) of a project (also called repository)
    b. Formally called a Version Control System or Source Control Management
2. Fundamental Workflow
    a. As we change the project over time
    b. Create save points (called commits) that track the timeline of the projects's evolution
    c. Create save points, timeline of the code/files

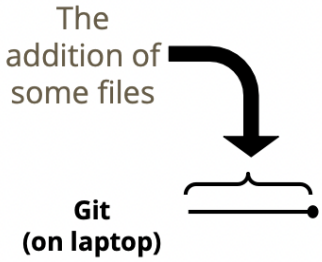
d. 

→ add checkpoint

e. 

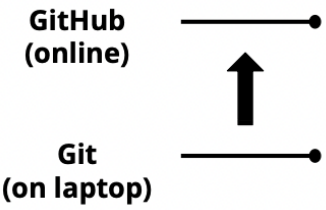
f. 

→ adding some files

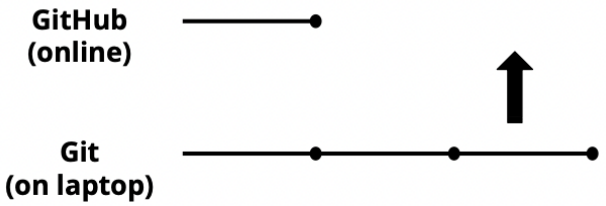
g. 

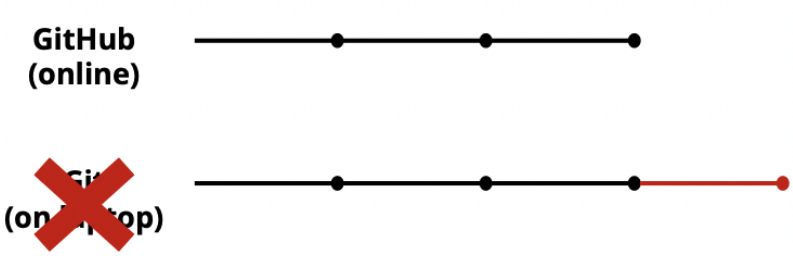
h. 

→ can add new files and build timeline

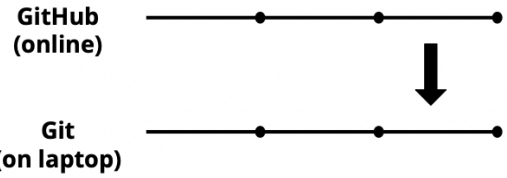
3. Demo
    a. Git Commands
        i. 1. git init → create a git repository to track the project
        ii. 2. git status → check the status of the git repository that checks the commit points
        iii. git add “Filename”
        git commit -m “Message”
        iv. git config –global user.email “9chrisyang22@gmail.com”
        git config –global user.name “Jeong Yong Yang”
        v. git log → history of commits that gives an unique id for each commit and message
        vi. git diff → see the difference between what was previously committed and what we’re looking forward to add and commit
        vii. touch secret.txt → create a “secret.txt” file
        viii. rm secret.txt → delete “secret.txt” file
        git add secret.txt
        git commit -m “delete file” → if we do git.log, there is still history of the repository secret, which should not be the case
        ix. Therefore, we need to change the history
        x. git log → to check the id
        git checkout “unique id” → come back to that point in time (switch to that moment), fundamentally change the project and go back in time
        git checkout master → come back to the current point
        xi. git rebase -i “unique id” → opens bin
        Delete the commits we do not want
        Do dd to delete the lines → control c to get out of the mode, do :wq to quit mode
4. GitHub vs Git
    a. Git → (terminal) a version control system
    b. GitHub → (browser) a website to backup and host the timeline(s) of your project
5. Fundamental Workflow
    a. Push the updates to GitHub (from your laptop) to back up your work
    b. 

→push this project onto Github (uploading the timeline)

c. 

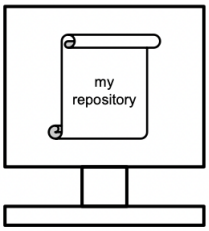
d. 

→ even if laptop crashes, GitHub still has the timeline up until the time we saved

e. 

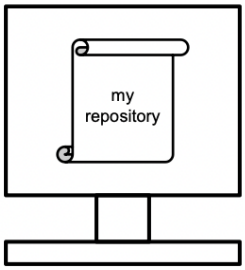
→ can download the whole history

6. Initialize a repository
   a. git init



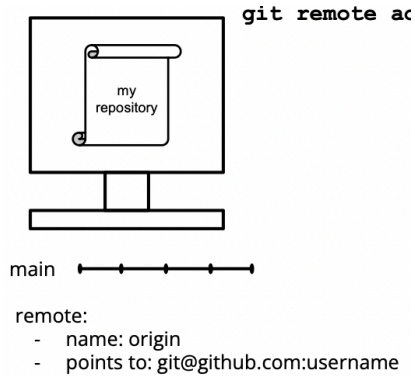
   b. main

7. Add and Commit Changes
   a. git add <files>
      git commit -m “some message”

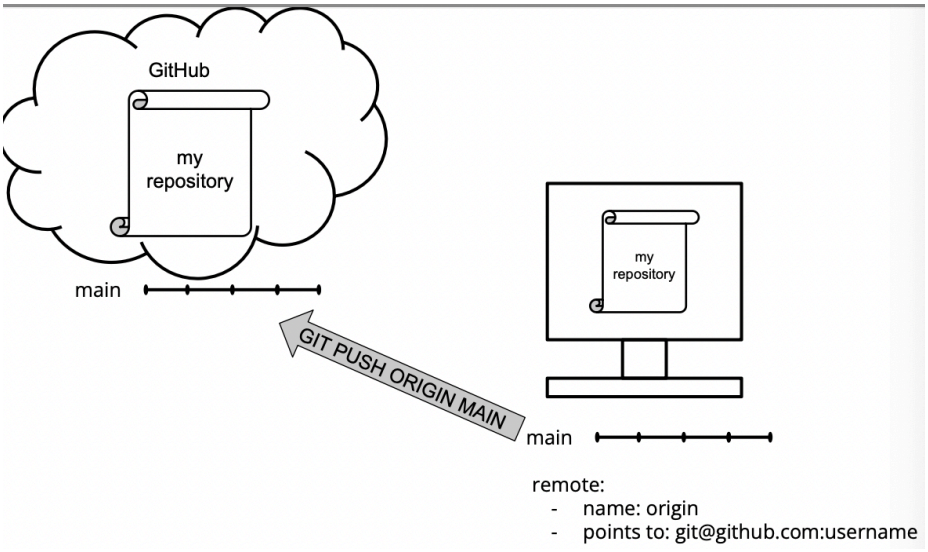


   b. main 

8. Add a Remote that Points to GitHub
    a. git remote add origin <link>

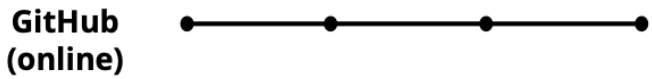


    b. → create a link between local laptop and gitHub URL

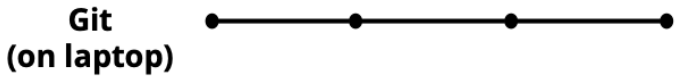


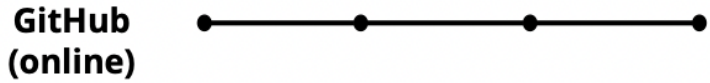
    c. → push the history of the timeline by using git push
9. Demo
    a. Use SSH
    b. ssh-keygen → create ssh
    c. cd .ssh
       cat id_rsa.pub → copy the long id and add SSH key to GitHub
    d. git remote add origin “URL”
    e. git status
       git push origin master → files are uploaded to GitHub repository
10. Motivation
    a. For each project (repository) I own, I want to write code where:
        i. Iterating on (+ keeping track of) different versions of the code is easy
        ii. Work is backed up to and hosted on the cloud
        iii. Collaboration is productive

11. Iterating on Different Versions
    a. The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase
    b. It may be easiest to add this feature at a specific commit

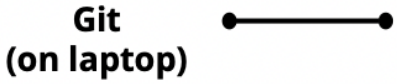


c.

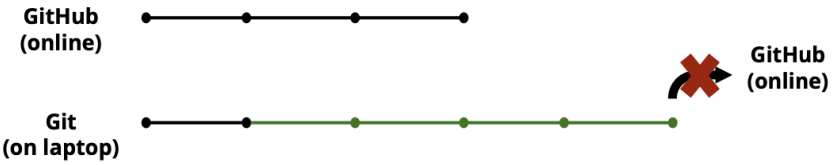




d.

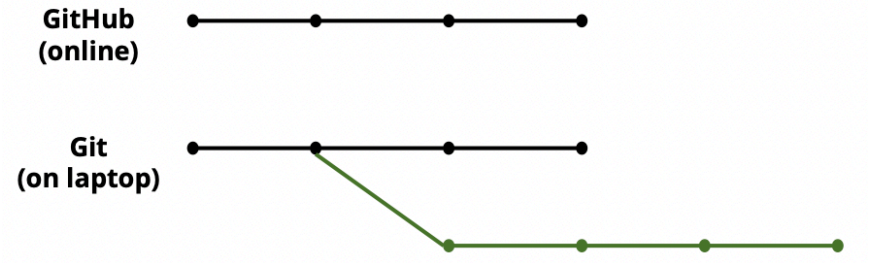


    e. What happens now?

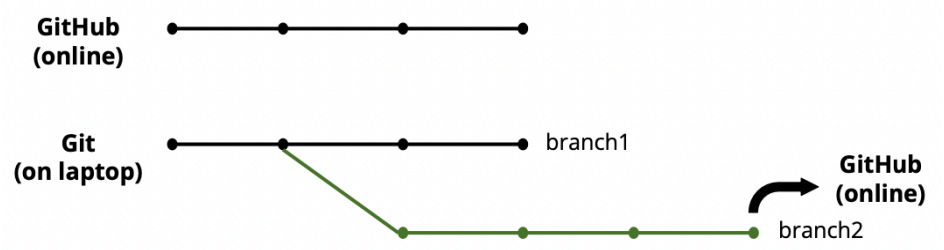


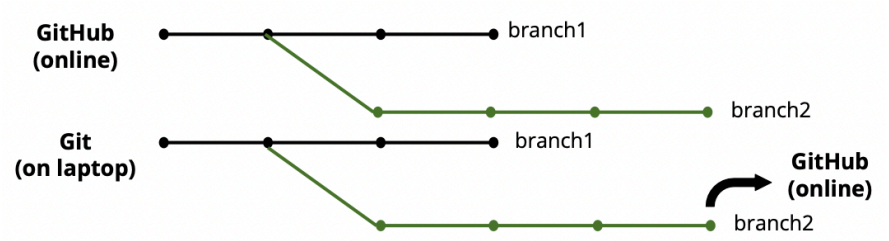
→ when there is conflict, Git disagrees and prevents saving
dd

    f. Looks like we need:
        i. A way to preserve both versions of history
        ii. A way to overwrite history if we choose (this is dangerous as we will lose that history)
    g. Try again
    h. Branch off of that particular commit to create a new timeline

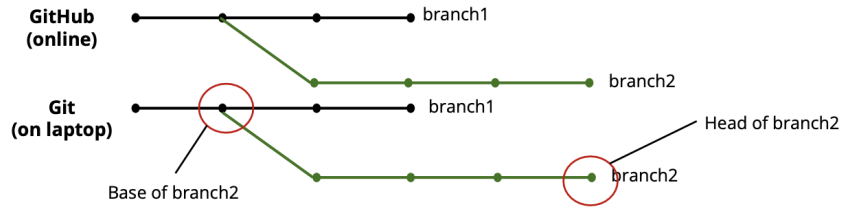


i. We can push commits per branch



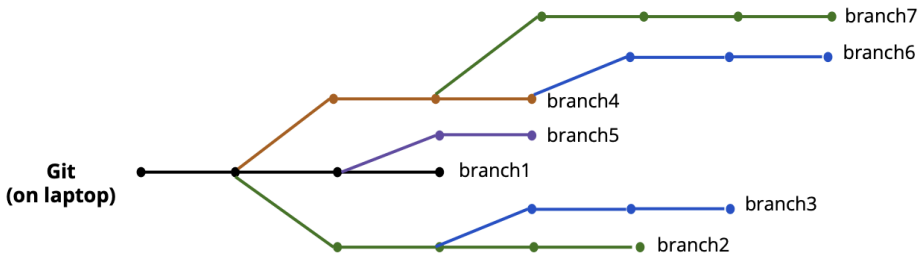


j. → multiple branches and timelines



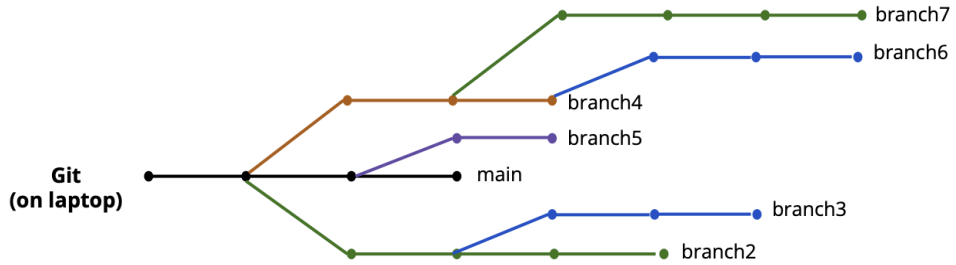
k.

l. We can create lots of branches

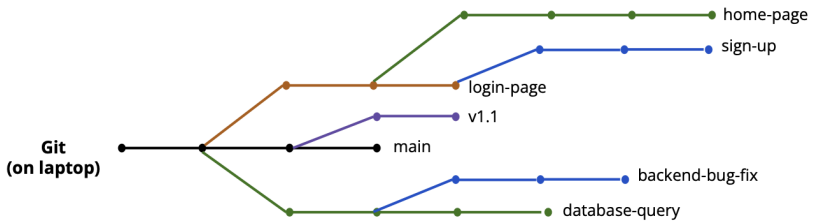


m. But one branch needs to chosen as the primary, stable branch

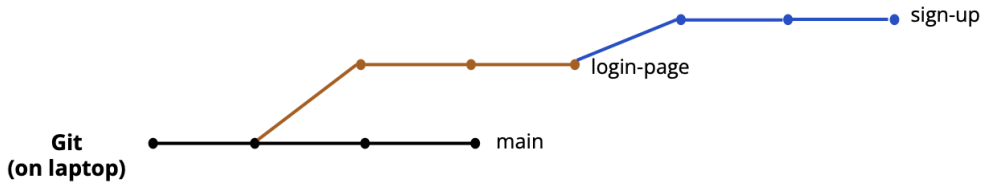
n. This branch is typically called the “main” branch



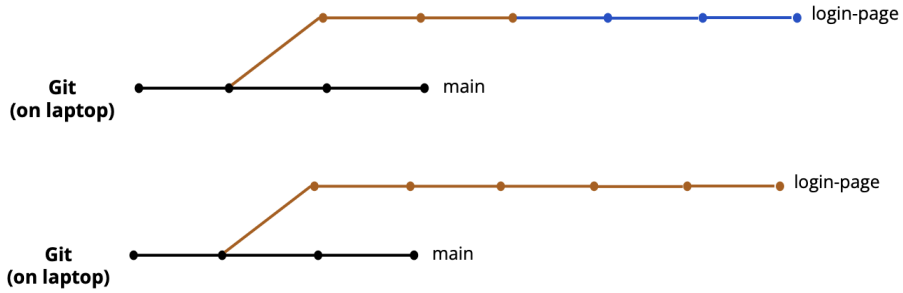
o. Other branches are usually named after either the feature that is being developed on or the major or minor version of the software / product



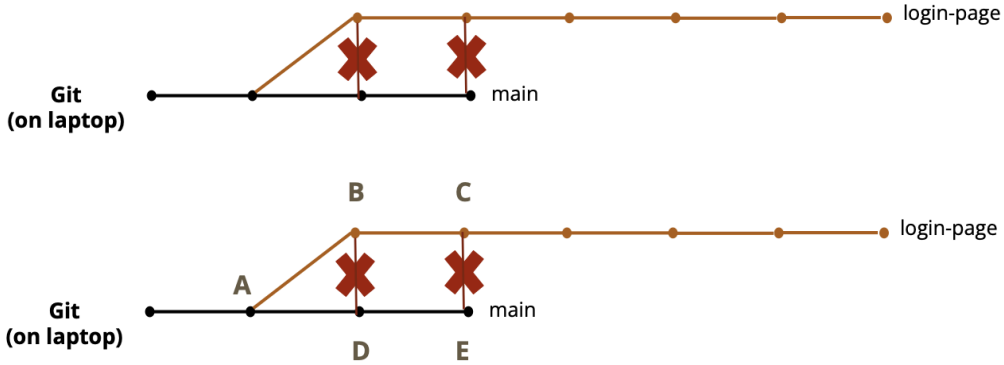
p. At some point we will want to clean up certain branches merging them with the master / main branch or with each other



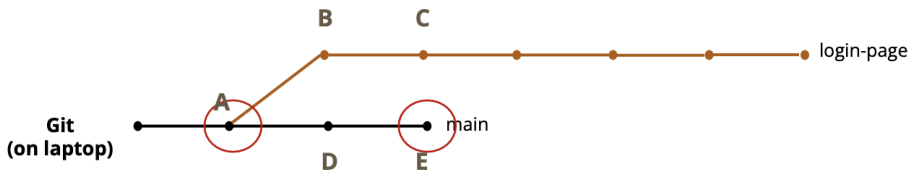
q. Merging is trivial if the base of one branch is the head of the other - the changes are “simply” attended

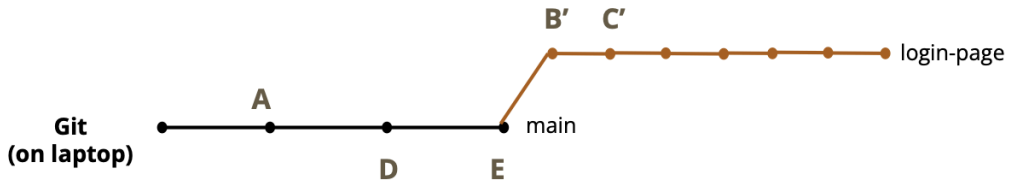


r. When this is not the case, commits can conflict with each other

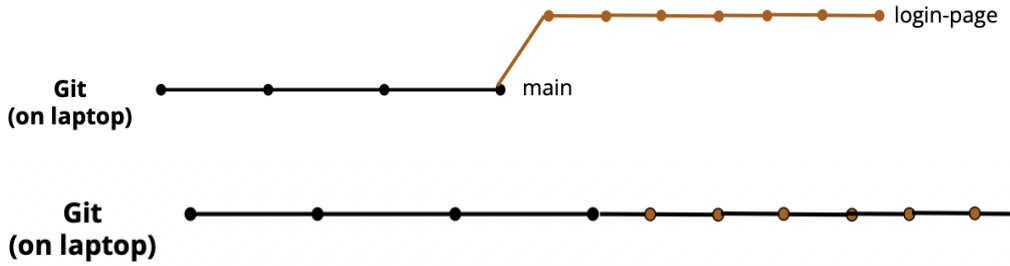


s. We need to change the base of the login-page branch (rebase) to be at the head of the master branch

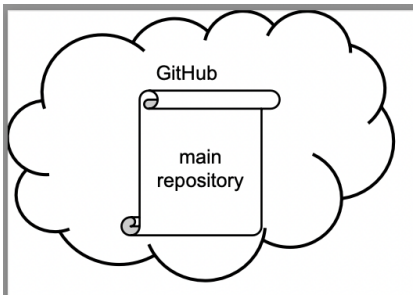


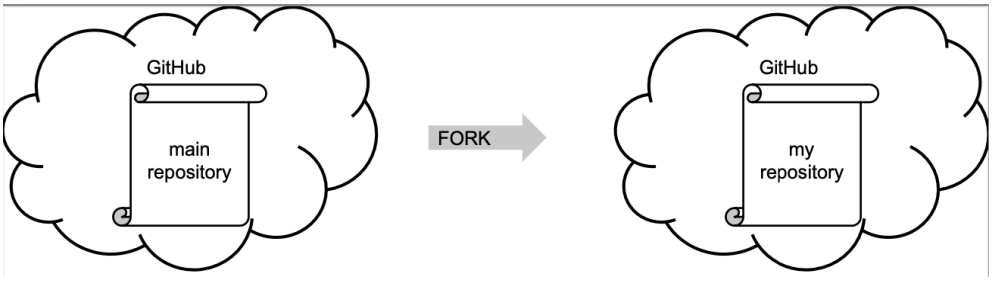


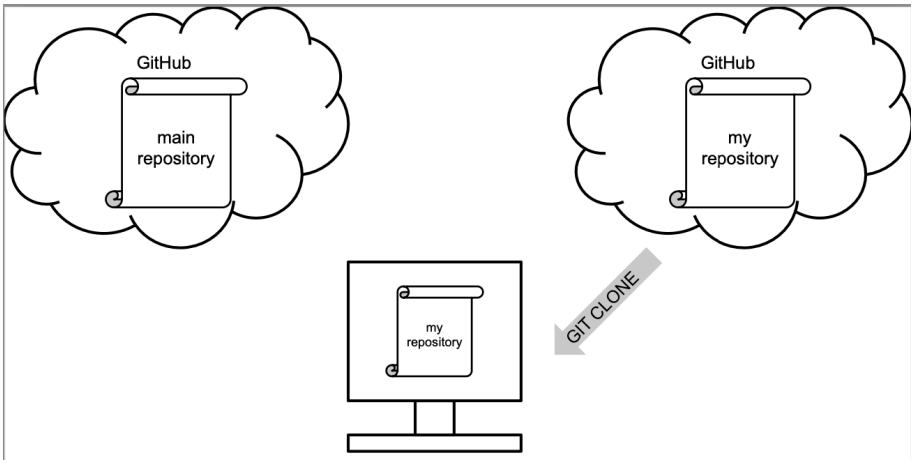
t. This is not a simple operation and will often require manual intervention to resolve the conflicts



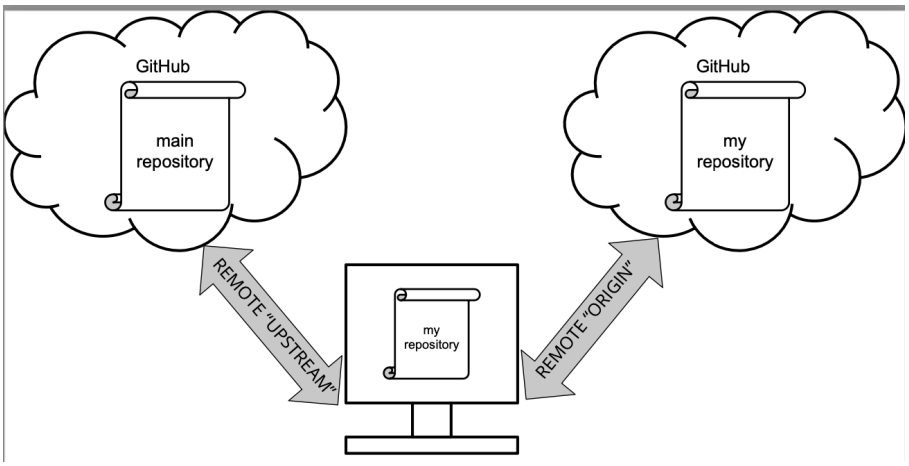
12. Collaboration
    a. Other repos can be thought of as other branches
    b. In order to contribute code, collaborators must:
        i. Make a copy (fork) of the main repository
        ii. Make all the changes they want to this copy
        iii. Request that part of their copy be merged into the main repository via a Pull Request (PR)

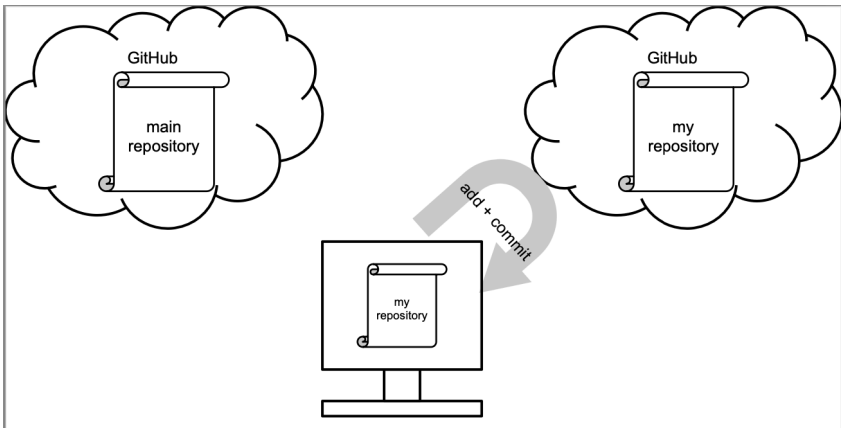
c. 

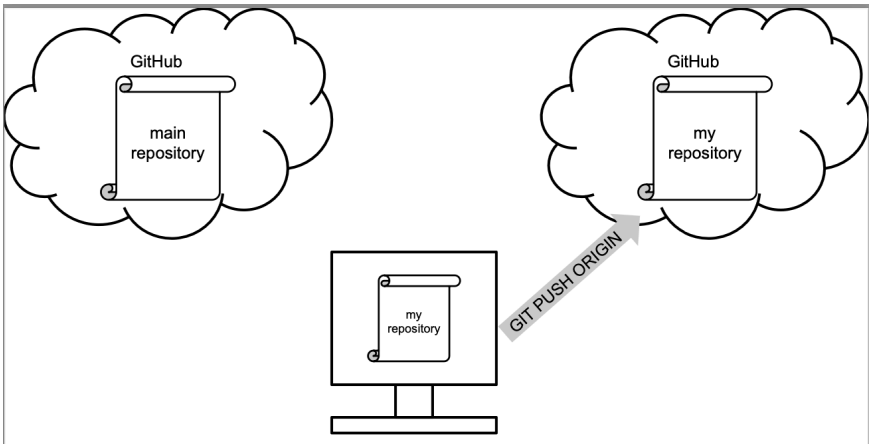
d. 

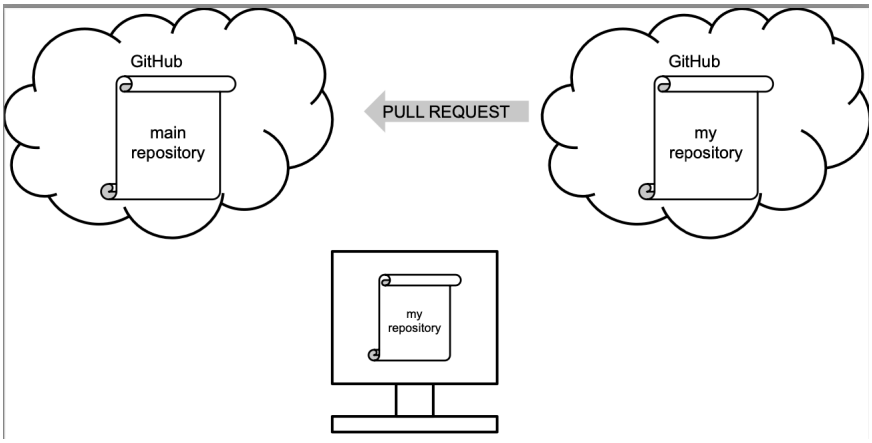
e. 

→ clone the repository locally so we can make the changes we want to make

f. 

g. 

h. 
GitHub
main repository
GitHub
my repository
my repository
GIT PUSH ORIGIN

i. 
GitHub
main repository
PULL REQUEST
GitHub
my repository
my repository

j.