Introduction to Git & Github

Notes introduction

Git repository = timeline: Everything is stored here, however if you go back and adjust something, the 'present' is also adjusted. It is locally stored and you can work together with other people.

Github = Backup of your timeline: Instead of going back in 'time' to adjust your document, you just retrieve a copy of your old document that you can adjust, without adjusting the newest version of it. (also online repository for code). Backup place for Git project. Cloud based location ~remote

- git init to initialize git -> only sees whats in folder and under!
- git is invisible in folder. Its the history of all the things you do in the folder
- Any point in timeline is a snapshot: When/how to take snapshot? Develop area -> staging area -> local repository area (commitment!)
- $ git commit
- Everytime you commit, git makes you write a message of why the snapshot is made! So make it a meaningfull snapshot/message
- $ git commit -m "meaningfull message"
- What is meaningfull? Why was it changed/ How this addresses the issue / Effects due the change / Limitations of the change (eg i changed part of code to make it work but now other part doesnt work / I studied the main thing but now i need to study something else to understand the main part)
- Be as descriptive as possible! More info is better than too little
- Git is tracking so just tell what y-ou mean with the changes you made, no need to state where you changed or what you put in
- IMPORTANT: don't double ´´´git git´´´
- IMPORTANT: dont double init init git
- You can save as much as you want but it won't create a backup in timeline unless you commit!

- Don't init git in a subfolder where the parent folder already init git
- Init in subfolder will create a timeline/repository inside a timeline/repository

Conceptual areas

- Develop area: working directory
- Staging area: Temporary space. Place where you dump stuff before putting it in local repository (git add). Also to organize logically your area. You can add 1 file, commit, then add another and commit and they will both be saved in the timeline as 2 seperate timespot.
- Local repository: Place where your snapshots/timeline is saved
- Remote repository (in our case it is github): Backupspace on a server
    - Create repository
    - Name: Easy if you use the same name as your repository (the name of folder that git is tracking)
    - Description
    - README.txt: Detailed description of your project and tool usage (if it is the case) (should always have one!). Describes my project, code, main goals, usage, etc. Can also include links and directions for related data for example. Github will automatically recognize the file README.txt and show it on frontpage.
    - .gitignore: A textfile with no extension that should always be in non-caps, and where i list all the files/lists that should be ignored and not tracked by git nor shared on github: eg: data file

      Make a .gitignore and in it list all files that should be ignored filename must be '.gitignore'. Github will ignore the file then, and it won't show that file. Do it before committing the file, otherwise an older version still might be somewhere in the history.

      Can also ignore whole folder!

Git status allows me to check what files are:

- To be staged: You have committed, it before, you have made new changes and git recognise the new changes are not yet added nor committed
    - Why is staging inmportant? Too many commits will make it hard to find a specific one. Too little commits will have too many changes in one commit. Use staging area as

organisation point in order to classify what should be committed together.

- To be committeed: You have committed this file before, you have made new changes and git recognises you have added but not yet committed
- untracked files: Is a completely new file/folder, you have made changes and git recognises that you have not yet added nor committed.
    - This way you can check and organise in combination with staging area. Everything in staging area will be comitted together.

Travelling in the timeline (local repository)

- git log to see history of all your commits: who, when, msg, commit ID
- git show/diff to see difference between two commits
    - Show just shows what happend in ID1 and then ID2
    - Diff shows the difference between ID1 and ID2
- How do we connect to a remote repository
    - If you have acces and can edit the repository -> ssh key
    - if you just need copy -> html key
- Backup created only for commited changes!
- How to make files cross the bridge?
    - git push
    - first time will always give error message:

      fatal: The current branch master has no upstream branch. To push the current branch and set the remote as upstream, use

      ```
      git push --set-upstream GitGithub_training master
      ```

      To have this happen automatically for branches without a tracking upstream, see 'push.autoSetupRemote' in 'git help config'.
    - We need to tell the remote one who is the master. Copy paste the command and run it
    - Can also remove files from github via git rm , commit and push (DELETES THE WHOLE FILE)

Github repository

- Commits to see the history of commits.
- Github 'bridge' works both ways.
- Need to see with pushing and pulling as it goes both ways so if you delete something on github, you cant push anything else from local to remote untill you have pulled the changes so it is synced. And once you pull it will adjust on your local. So if its deleted, it will delete on both and theres no way to get it back.
- If you deleted your git folder on your computer, you can clone the Github latest pushed copy via `git clone`
- Collaborations: Go to the repository, settings, access and then collaborators. Add people via their mail connected to their github. So if you collab, always make sure you are synced. Pull first to see changes others made before editing again, because git won't allow you to push if you're not synchronized. Carefull because you can't push at the same time same file. Person that created repository, is owner (can add other collaborator, change name, delete it, ...). Collaborator can edit but doesn't own it.

Experimenting Risk Free (branching)

- Like working on parallel timeline
- git branch to create a branch. Need to know in which timeline you're working after because it is seperate commit! (maybe also pushing will work on one timeline.) Can also merge them back later!

  commits: 1-2-3-4-5 If you branch from 3 and make new one: 3-4*-5*... In this timeline, 3 will be the first commit in your new timeline.

- - Maybe to test new things while keeping master timeline safe
  - When collaborating in a project
  - Looking for different solution on the same issue
  - Release series (V1.1, V1.2, ...)
- Branch
  - Alternative history of commits
  - Has a specific name (chooseable)

- Independent timeline
- Connected to a moment in time
- IMPORTANT: **Close all files your editing (add and commit/push), and after switching reload.**
- When branching: The Develop area becomes a mirror of the timeline.

Parallel timeline: how to experiment risk free

1. Create new timeline: branch and give it a new
2. checkout timeline you want ot work on.