repository = project/code

Git & GitHub

Git: version control system
GitHub: Repository hosting service

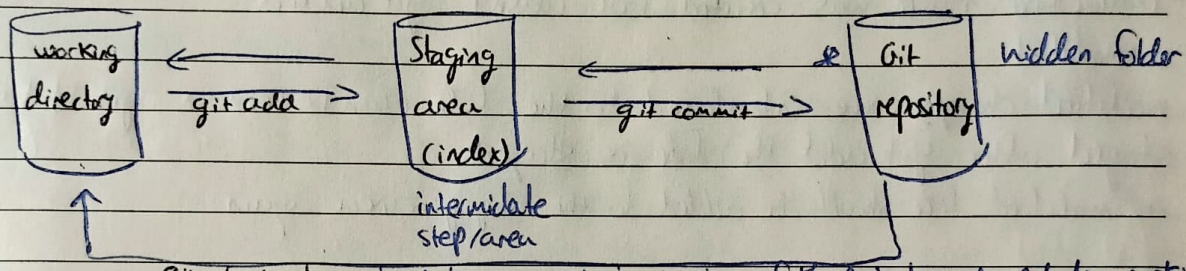
Getting started

cmd -> git --version if you don't have result, install git
git repository: Entire folder that is managed by Git
git objects types: 1) Blob 2) Tree 3) commit 4) Annotated Tag
Blob: files
Tree: Folders
Commit: pointer to specific trees, snapshot

* snapshot: snapshot of the project at a given time
Every object has a specific hash code (40 Hexadecimal), a unique identifier

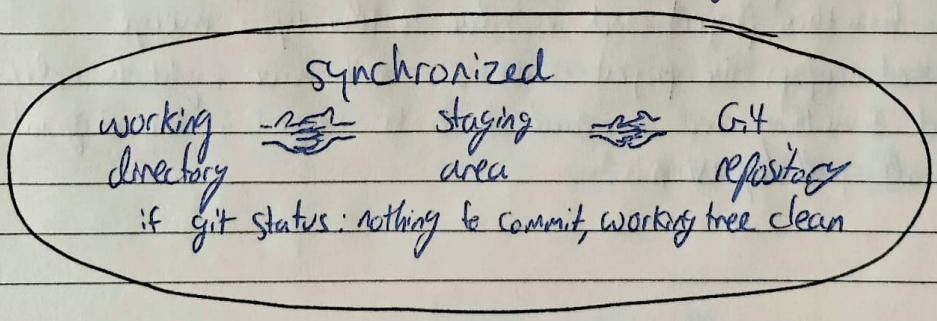
Commiting Changes in the Git Repository

In Git there are 3 areas where files/folders could be located



~~checkout~~

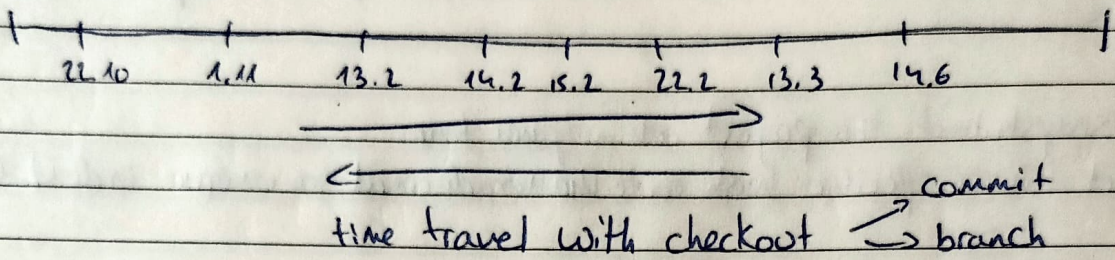
untracked file: file that is not tracked/being watched by git, I can change this file but those changes won't have trace
untracked file can't be commited, it need to be added before (with git add)
commit: When I do a commit, git save the author name and email in the ~~author~~ commit object but not only. Every commit hold the data on's
* author name + * email * description * Parent commit pointer
Parent: the commit that was made before (1 or many)



checkout branch: where I am, on which branche: A Head

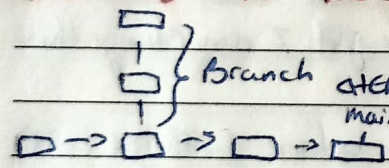
- Head : current location, I can move between branches and what git will do in the background, is to point the HEAD to another branch
Detached HEAD: a state that indicates that HEAD points to a specific commit instead of specific branch.

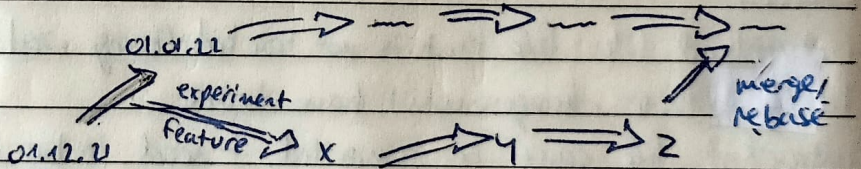
Where Am I? I can be on a branch or on a commit. if I am on a branch, git status will reply On branch x, else it will reply (HEAD Detached at x)

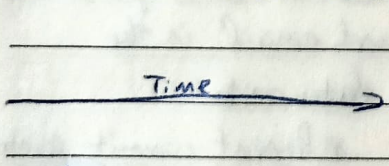


File Tracking Status
- untracked: a file was created but wasn't git added
- unmodified:
- modified: changes were introduced to the file (can be seen in "git status")
- staged: after git add the file is staged
a modified file should be added to the staging area again

Branches and mereging







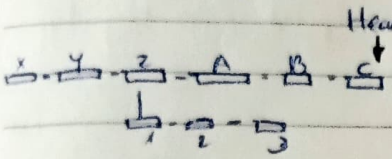
Branch: A copy of the original work, Keep only the changes

merge: please, will you consider to integrate my changes into master/main?
If I am from this project and I have authority - merge
If I forked (copy) the project cause I don't have right to edit it and I changed something and I want it to be merged in the original project this is call pull request

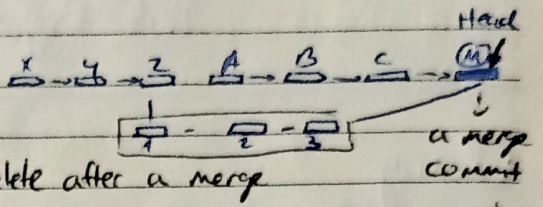
Merging process

1) create new branch from the main branch
2) make changes, commit
3) checkout main branch (or the receiving branch)
4) merge feature brunt to the receiving branch with git merge <feature branch>

Before merge



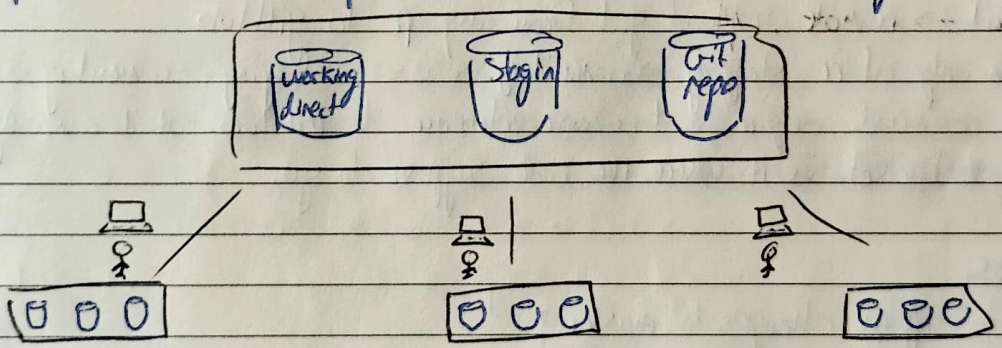
After merge



After a merge I can delete the branch. When merging, a new commit is created with 2 parents (in this ex: C & 3)
fast-forward merge: only when there are no other commits made to the original branch after the creation on the branch x. Git won't create merge commit. it will move the Pointer of the receiving branch to the last commit in the feature branch

Github: Remote Repository

look the same, have the same structure like local repository, instead of saving it on your computer, it saves it on the cloud.
connecting local repo to remote repo: you create new remote repo in the local repo. All remotes have specific names. The default is origion



When I connect the local repo to a remote repo
fetch: update, download of updates from remote to local repository but not working dir.
push: upload, push changes from local to remote
pull: download updates from remote to local repo and working directory: It's 2 command in 1: git fetch + git merge

* It's not possible to push to remote in detached HEAD state

Git Objects and files

are found inside all the folders in .git/objects
hash code : foldername + object name inside the folder
HEAD file : what branch the HEAD is pointing to
ref folder : contains all the branches

source: Packt - Git and GitHub Crash course
Prof: Bogdan Stashchuk

Cloning

1) SSH key: let you bypass entering your username and email address every time you want to make changes when working with remote repo
2) HTTPS: will require authentification
3) Download as zip

the SSH Key is generated with cmd command and saved locally where I indicate. I can present the string of the SSHK with the command cat and then in GitHub: Setting -> SSH and GPG keys -> new SSH keys

clone: download from GitHub and put it on my pc remote -> local

Pushing

push: local -> remote. Upload got from my pc to github
I can push only after staging -> commiting so it's staging -> commiting -> pushing
in the git command, origin point or make reference to GitHub but it can only be gitLab or bitbucket. Is it where the code supposed to go.

Branches

why not to upplay changes to master?
If I work in a team, and I modify master, Git doesn't know who to trust. should it trust my modification? should it trust the master that was written by all of the team? that's why It's good practicace to work with branches

Merge branch to main/master

Before merging, ALWAYS make sure that you are up to date with the target branch with git pull origin master
1) merge locally
2) push changes

check if you up to date with git log:
checkout the correct branch (master/main) and then git log. check the 7 first letters of the commit hash. Go to github, loose yourself and the master branch and see if the first 7 letters (hash code) are the same

Pulling & Fetching

Daoload of all the code from github, stores it in memory, ~~righ~~ write all the commits to the local repository => Pulling
what if you don't want to download and change your repo? => Fetching
what if you just want to see what have been done, the changes?
git fetch: download but don't apply changes
git pull = git fetch origin master + git merge origin/master

Go back in time

If someone made a changed in a code an I want to go back in time before the change, I can go to github repo, find the commit I want to go back to, copy the hash string and git checkout hash. Problem: I will be in HEAD detached mode and I can't push in detached mode. Solution: Assign this commit to a new branch, and this will exit me from the Detached mode -> git checkout -b <branchname> and if I want to push it to remote -> git push origin <branchname> (the same name)

Ignore files

mis en place: create a file with the name .gitignore. Open this fib with an editor and write the list of files you want to ignore. ex:
*.ico: for all the files that ~~end in~~ have the type ico
I need to add and commit it to master, and if I want it to be a guiding rule to the projet, t need to push it also

Create Alias

Edit the gitconfig file, Add [alias] and key = command without git. ex
[alias]
lg = log --topo-order --all --graph etc ...
and save the fide. When I will use git lg, it will replace lg with log --topo...
you can do it also by a command git config alias

Forking

cloning/copying entire project to your remote repository. The diffrence with clone is that clone copy it to your local repository your account/profile

why to fork?
If I fork, it believes under my repositories, and I can do everything. I can push.
If I just clone A certain open source project and I want to push, I don't have access to do it (imagine what would happen if I did, I can delete all the files and push the changes)

Git issues

a place to start a conversion regarding the code: a problem, suggestion, a bug, type etc...

you can assign it to someone, add tables, milestones

Pull Request

you have some code that you want to incoperate in my code or I have some code that I want to incoperate in your code
what we can do with pull request:
- create a merge commit: All commits will be added
- squash and merge: The 1 commit will be added
- rebase and merge: The 1 commit will be rebased
what is 1 commit? All the commits are squashed into one commit
+ After accepting a pull request, if you don't need the branch, you can delete it.

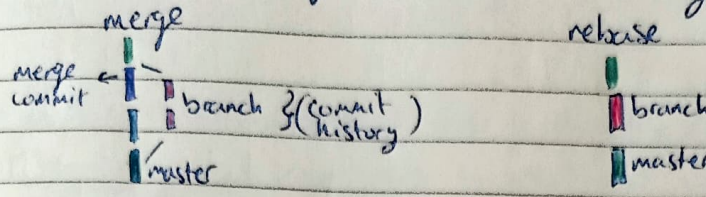
UnCommit

You did a commit and you want to take it back, you missing a file, you did it on the wrong branch etc. You can undo it by hard/soft reset
- hard reset: delete the commit, delete your work also and take you back in time to how the code was before all the changes
- soft reset: delete the commit but keep the changes, does not delete files

Force Push: not recomanded if working with a team
A dangerous actions that sometimes required. What if you did a commit and you regret?
if it's local and you didn't pushed it - use reset for uncommit
if you pushed it to github? I can use force push, but use it with caution. Why? because it will delete files and can end up in deleting code of your team mates
to do this:
1) git reset --hard <hash>
2) git push origin <branch> -f #[-f = force]

Rebasing

Rebasing is like a merge but without creating a commit checkout



with merge it's always easy to go back to the moment of the merge because we have a merge commit that made this fusion. With rebasing we don't have this "flag" that will tell us at which moment the fusion was made.
~~rebasing~~ with rebasing we don't have all the commit history with merge - yes.

Merge/Rebase conflicts

when you try to merge/rebase code into other branches/repos and you and someone else has made changes on the same line, you will get conflicts!
Git doesn't know. Do you prefer version A of the code? version B? maybe you want to edit and make hybrid?

```
+<<<<<<< HEAD
+ angular@cli: 16.4.13 . —> locally
++ = = = = = the code is seperated by ===
+ angular@cli: 14.2.16 —> remote
++ >>>>>> dce4a7413 ...
```

this is where the conflict is what do you want to keep

How to resolve? Edit the file and keep the version you want to keep, stage it and commit and it will create merge commit
If we try to rebase, we edit the file, we chose what we want to keep, we stage but then we don't need to commit, we can run git rebase directly with git rebase --continue