# Github

## Git started

Check git version

- git —version

Change name or email

```
git config --global user.name "Name"
git config --global user.email "hello@email.com"
```

To create a new repository

- git init

This creates new hidden files, which can be made visible via files.exclude in settings. Removing these files removes git from the project

Check status to see the branch you are in and which files have not been committed yet

- git status

If there is files with secret keys or build info which shouldn't be public, put them in a '.gitignore' file so they can't be committed

To select files for the staging process (the current picture of your dir):

- git add . (to add all files in the dir)
- git add <filename> (to add single file)

To undo staging use

- git reset . (again for all files)

Commit staged files to a repository (save the picture

```
git commit -m "initial commit"
```

Tip: Add files and commit in a single command

```
git commit -a -m "additional commit"
```

or

```
git commit -am "additional commit"
```

Use git log to see the history of commits:

```
git log
```

# Remote

Connecting a remote GitHub repo to your local repo

Git remote which remote repos are linked to your local project

git remote add origin <url-to-your-github-repo> add the remote github repo to your local project

→ git remote now provides origin

→ git remote -v  also provides the repo link

→ git remote show origin provides more info like branch

Git push: takes the local code and syncs it with the remote repo

git push origin master  pushes the code to remote origin, branch master

add -u flag to set origin as upstream remote in your git config so git pull can be used without any arguments in the future (use when the remote repo is the final source of truth)

On GitHub you can change code as well quickly, now remote repo is 1 commit ahead on local repo

2 ways to combine the different master branches

- git fetch will download the latest changes, but local code is not updated yet then, this is done with the merge command
- git merge origin/master (takes 2 branches and merges them together) Here the origin/master (remote) is the branch to be merged onto our local code

Simpler way:

- git pull combines fetch & merge no need to add origin master as this was already specified with the -u flag earlier
- pull won't work if uncommitted changes in current local dir, either commit local changes first or use stash (later)
- conflicts will be handled later

When you want to copy a remote repo to your local machine and optionally change the name of the directory (also keeps reference to the remote repo s.t. you can pull from it)

git clone <repo-url> <local-directory>

## Collaboration

Branching: team members can branch of to work on their own features and add them to the main codebase later

git branch lists all branches. Current branch you are in showed in green and has a * in front (you can check with git status)

git branch <name> creates a new brand. After creating a new branch, you not automatically working in this branch.

git branch -d <name> deletes a branch

git branch -M <name> changes the name of the current branch to <name>

git checkout <name> switches the user from the current (often master) branch to the <name> branch. Then all commits will be separate from master branch.

git checkout -b <name> creates a branch and automatically switches user to that branch

git checkout - takes you into your previous branch

Merge conflicts

Merge conflicts happen when two commits affect the same line of code at the same time.

1. Feature branch modifies line 5 and commits.
2. Master branch modifies line 5 and commits.
3. Master branch tries to merge feature branch.

Here's how a merge conflict looks from the command line:

```
git branch feature
# make some changes
git commit -am "awesome branch stuff"

git checkout master
# make some changes to same code
git commit -am "master branch stuff"

git merge feature
# CONFLICT!
```

Use git diff to compare the changes in the feature branch and master branch.

```
git diff
```

The easiest way to fix the merge conflict is use the editor to choose between the incoming changes (feature) or the existing changes (master). Then create a new *merge* commit with the changes you want to keep.

```
# choose preferred code on master branch
git commit -am "resolved merge conflict"
```

If you're not sure, you can abort:

```
git merge --abort
```

Fork on github will copy (someone else their) repo to your own account. This way you can make changes to it without affecting the official repo. But you can also pull new changes from the original while you modify it as well. Or suggest a pull of your code by the author (lets say new feature) (contribution).

## Advanced

When you are working with a wrong file

git reset removes all the files from the staging area

or

when a commit is made with the wrong file, you check the commit id using git log and then use

git reset <commit_id> to bring you back to the stage before that commit, then you can modify or change things as needed. It keeps your changes in the working dir

git reset <commit_id> —hard does rollback and discards all changes (you lose the file), be careful with that,

NEVER HARD REMOVE A COMMIT IF ITS ALREADY PUSHED TO REMOTE REPO WHEN WORKING IN TEAMS

For this, use git revert

The previous commit is not lost and still in the commit history but overwritten. A new commit is added and overwrites it.

git revert <commit-ID> -m "reverting last commit”

In case you are not happy with your commit message, update the message of a commit with:

git commit --amend -m "better message”

In case you forgot to stage a file before committing, you can

git add <your-file>
git commit --amend --no-edit
no edit to commit message

This prevents git reset or git revert

Stash, saves your work without committing (useful for experimental stuff you want to temporarily save)

git stash removes content from a file

git stash pop puts the most recent stashed item back where it was previously

git stash save <name> to give a stash a name

git stash list provides a list of all stashes

git stash apply <index> pops the stash with the given index

In a team, often use rebase because commits and merging becomes to messy in logs

Git rebase is a way to integrate changes from one branch into another by moving or combining a sequence of commits to a new base commit. It rewrites the history

git checkout feature
git rebase master

git rebase master --interactive to combine (squash) multiple commits together

You do this often in teams just before you pull or merge back into the master branch