Karlsruhe Institute of Technology

GRK 2450

Introduction to Version Control with Git

Lukas Petersen | 26. May 2023

Questions for today

- What are git and GitHub?
- How to do commits?
- How to create branches?
- How to merge branches?
- How to contribute to other repositories?

Questions for today

- What are git and GitHub?
- How to do commits?
- How to create branches?
- How to merge branches?
- How to contribute to other repositories?
- How to reattach a child's detached head?

git and GitHub

git

- Local version control software
- Saves and tracks changes made on files inside a repository
- Controlled by command line commands like
  - git init
  - git status
  - git add
  - git commit
  - git log

GitHub

- Hosting platform for git repositories
- Makes it easy to access and share current and previous versions of your code
- Enables collaboration on larger projects
- Possible endpoint for git remote commands

When to use git

Collaboration

"Dude, I tried to use this script of yours. Why doesn't it work?"
"Oh, that's the old version.The new version is at \data\..

Legacy Comments

```
#def old_function(a,b,c):
# return a*b*c
def new_function(a,b,c):
return a*c*b
```

Cross Server Work

"I spent all day working with this script before realizing that I didn't bring over the changes I made last week on my own machine. "

Getting started with git

Things you have to do once and then never again(on that machine):

- Verify installation
  ```
  git --version
  ```
- Configure your name and email
  ```
  git config --global user.name "FIRST_NAME LAST_NAME"
  git config --global user.email "MY_NAME@example.com"
  ```
- Optionally, change the name of the default branch(Github's default branch is called main)
  ```
  git config --global init.defaultBranch main
  ```
- When pushing to a remote repository for the first time, you might have to authorize VSCode to access GitHub. Without a code editor setting up an SSH-Key prevents repetetive logins

Doing local commits

Commits consist of three steps: **Staging**, **Commiting**, **Pushing**

Do this once per project

- Create a folder on your computer and change into it:
  ```
  mkdir test_git
  cd test_git
  ```
- Initialize the new folder as a git repository:
  ```
  git init
  ```

→ a hidden folder .git is created

Do this once per commit

- Create any change in the folder, e.g.:
  ```
  touch app.py
  ```
- **Stage** your change:
  ```
  git add app.py
  ```
- **Commit** your change
  ```
  git commit -m "Descriptive message"
  ```

→ Your change is now saved locally

Doing local commits

Commits consist of three steps: **Staging**, **Commiting**, **Pushing**

Do this once per commit

- Create any change in the folder, e.g.:
  `touch app.py`
- **Stage** your change:
  `git add app.py`
- **Commit** your change
  `git commit -m "Descriptive message"`

→ Your change is now saved locally

Let's walk through what happened

- Check the status of your repository:
  `git status`

→ Working Tree and HEAD are in sync

- Create another change and check the status:
  `touch index.html`
  `git status`

→ Working Tree has changes, that are uncommitted

- **Stage** the change and check the status:
  `git add app.py`
  `git status`

→ Working Tree has changes, that are committed

- After commit Working Tree is clean again

Doing remote commits

Next to the status of the Working Tree and HEAD, you also have to be aware of the status of the remote

- Create a new repository on your GitHub-Account (Public or Private)
- Add the new repository as a remote to your local folder named origin (by convention)
  `git remote add origin https://github.com/YOUR_NAME/YOUR_REPO.git`
- Push your local changes to the remote AND set a upstream location as a default for future pushes
  `git push -u origin main`

→ Your changes are now live on GitHub

- Future pushes on this branch can be done with `git push`
- Make sure to pull changes from the remote, if you are working with someone else:
  `git pull`

Undoing local commits

If you want to go back to a previous commit or undo fatal mistakes

- Check the logs of your commits:
  ```
  git log
  ```
- Identify the commit to return to, copy the corresponding SHA-code(Secure Hash Algorithm)
- Reset your progress to the previous commit:
  ```
  git reset 7733eeb9adef3978d088c6b7d9db56237066fad4
  ```
- This will return you to the state JUST before you committed, changes are still staged, but you are able to unstage the changes, that you don't want to commit
  ```
  git restore --staged fatal.mistake
  ```

Undoing local commits

If you want to go back to a previous commit or undo fatal mistakes

- If all the all the changes in one commit are unnecessary, you can reset HARD to return to the state before staging:
  `git reset --hard 7733eeb9adef3978d088c6b7d9db56237066fad4`
- To go back to a commit relative to your HEAD the following syntax is also possible
  `git reset HEAD~2` # to go back 2 commits
- Go back a previous commit without removing more recent commits using:
  `git checkout 7733eeb9adef3978d088c6b7d9db56237066fad4`

→ This will enter a "detached head" state, where the current Working Tree is not associated with any existing branch. You could start new branches from here

Commits with VSCode

VSCode makes our lives easier, as it has a graphical user interface for most git commands

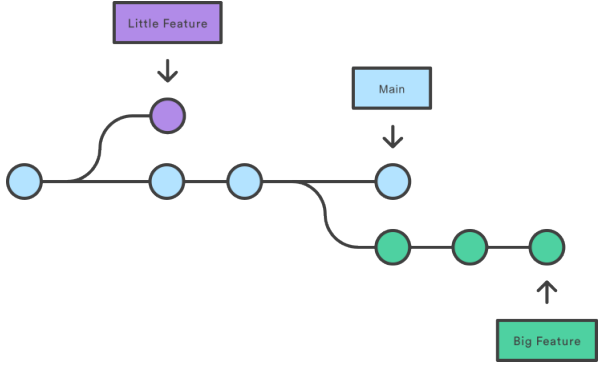
- File –> Open Folder –> Choose any folder, that has been git initialized
- Open the third option on the left sidebar called "Source Control"
- Make a file change. Staging, Committing and Pushing is completely intuitive from this here

! From this point on it's easy to forget, what's actually happening under the hood

- Install the GitLens Extension in VSCode for easy comparison over all previous changes

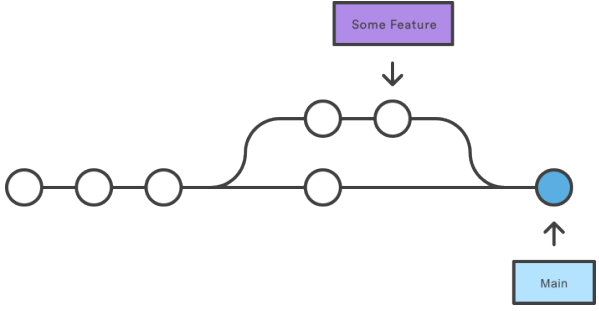
Branching

- Branches are an experimental playground to develop new features or refactor code
- The HEADs of each branch are unaware of other branches, although they might share a common history



Branching

- Branches are an experimental playground to develop new features or refactor code
- The HEADs of each branch are unaware of other branches, although they might share a common history
- Branches can be merged, commits from both branches will be combined into one resulting branch



Branching

- Create a new branch called feature:
  `git checkout -b feature` or `git switch -c feature`
- Get a list of all local branches and of all local and remote branches:
  `git branch`
  `git branch --all`
- Create a change, stage, commit and push it. If you still use git command, you will have to add an upstream origin to track the changes when pushing
  `git push --set-upstream origin feature`

Merging without Conflicts

Merges into main should be done on GitHub to let other people review changes before merging. For merges from main to your own branches(experimental playgrounds), you can merge locally. But do whatever you like, I am a student, not a GitCop.

- Look at your repository on GitHub, you will have two available branches
- Create a pull request for your feature branch
- Complete the pull request, delete the feature branch when done
- Checkout your local main branch. The changes won't be present by now, we still need to:
  `git pull`
  without added upstream: `git pull origin main`
- git pull will merge changes from the remote into your HEAD. Alternatively, use git fetch to only stage the changes beforehand and only commit what you want

Merging with Conflicts

When multiple changes happen on the same file, you will have to decide, which changes to keep. This is called resolving a conflict. Let's create and solve a conflict.

- Create a new branch slowfeature
  ```
  git checkout -b slowfeature
  ```
- Make a change in one of the existing files and commit it
- Checkout main and do another change in the same file and commit it
  ```
  git checkout main
  ```
- Checkout back to slowfeature, and try to merge main into slowfeature again
- Resolve the conflict by deciding on on of the options both or none
- Stage and commit the resolved change
- Boast about your conflict management skills in your resume

Other Features on GitHub

- Displays the README.md to inform other about the purpose of the project/give instructions how to use it
- Ability to change files, create/delete branches on the remote
- Remark issues in projects
- Create pull requests
- Configuration of security and requirements for commits
- General statistics, including a graph representation of the branches

git clone/git fork

To use other code from GitHub, you will have to clone it and use it locally. Copy the url and use the command `git clone THE_URL_GOES_HERE`. You now have a local version of the remote, but you might lack the necessary permissions to actually push changes.

If you actually want to keep developing on someone else's code, you might want to fork it instead. In the top right corner of the repository on GitHub there is an option to fork the repository. This will copy the repository and create your own repository from it, that you can clone and use as you like.

.gitignore

The .gitignore is a special file inside a git-repository. Any file name inside it will not be tracked by git commands. It allows * as placeholder, with e.g. *.csv any csv files in the will not be affected by commands like `git add .`, which would normally stage all changes in the folder. They also won't show up as changed when comparing between different commits. Temporary files or constantly changing files like logs are probably something, that you would add to your .gitignore

Overleaf + git/GitHub

- Altough you might not need git for coding, there is a more common use for version control: Writing
- Overleaf supports an option to link your Overleaf folder to a GitHub repository
- Never be scared about making major changes again, that you later might regret
- Regulary push your current state and use your git knowledge to go back, if requiered

Summary

Commits

```
git add .
git commit -m "..."
git push
```

Branching & Merging

```
git checkout -b new_branch
...
git checkout main
git merge new_branch
git branch -d new_branch
```

Further Information

- Original documentation(kinda hard to digest): `https://git-scm.com/doc`
- Nice set of tutorials: `https://www.atlassian.com/git/tutorials`
- Good explanation: `https://www.youtube.com/watch?v=RGOj5yH7evk`
- Git for dummies: `https://www.youtube.com/watch?v=mJ-qvsxPHpY`

Playground

Please let me know your GitHub-Account and experiment around on your own for a while.

I'll add you as collaborators to this test-repository. Then try to create branches, merge them, mess stuff up, and clean up other's mistakes.

Try to find problems, where you can't do what you want, so that we can go through them together again.