Git Workshop - Let's git our shit together!

Installation und Accounterstellung

- siehe `README` des Repositories
- `www.github.com/friep/git-our-shit-together`

Partner up!

- Mona Lovalace Octocat

- Grace Hopper Octocat

Warum Git?

- Masterarbeit.docx

Figure 1: Help

- Masterarbeit_v1.docx
- Masterarbeit_FINAL.docx
- Masterarbeit_FINAL_TimsKommentare.docx
- Masterarbeit_FINAL_FINAL.docx

Version Control to the Rescue!

- Beispiel: diese Präsentation.

Daten runterladen - Fork und Clone

Fork und Clone

@boxbg-blue text-black rounded

@box[bg-blue text-black rounded](Fork#“A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.” (Source)

Hands On 1 - Fork und Clone

Mona

@ol

- https://github.com/friep/git-our-shit-together/: Fork (oben rechts)
- `https://github.com/{USERNAME}/git-our-shit-together` öffnet sich
- unter Settings->Collaborators Grace hinzufügen

@olend

Hands On 1 - Fork und Clone

Mona & Grace

@ol

- Gitkraken Clone Repo -> Clone with URL
- Kopierten Link unter URL eintragen

@olend

Oh!

Oh! - Nicht-Gitkraken

```
Cloning into 'git-our-shit-together'...
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

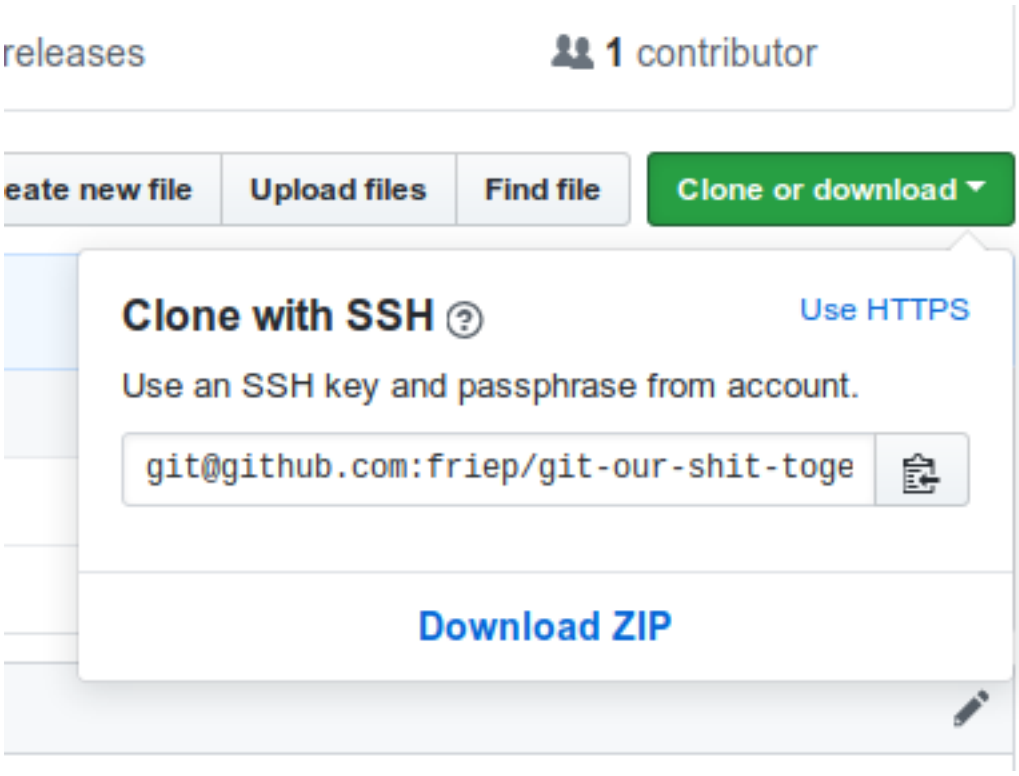

Figure 2: Git clone

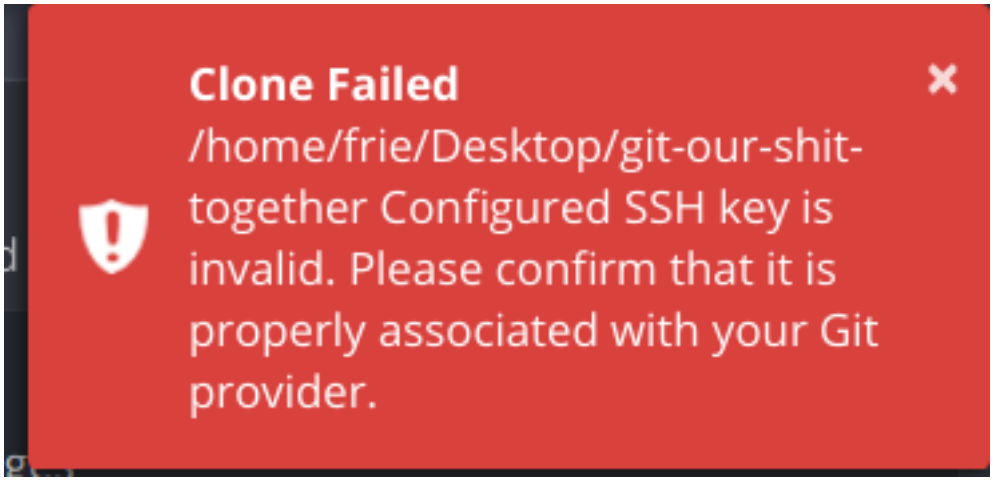

Figure 3: Git clone error

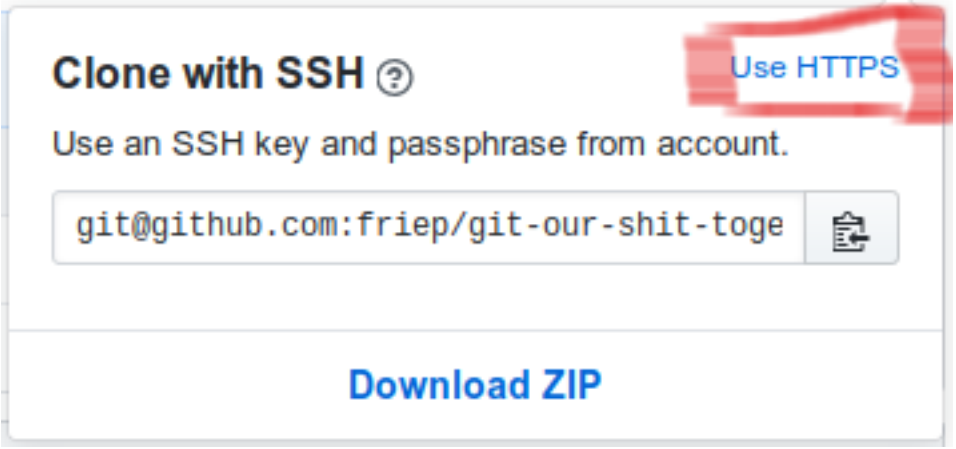

Figure 4: Git clone

SSH

@ul

- Download:
  - prinzipell jede*r @fa [lock-open] über `https` –> @fa [github]: @fa [check]
  - `ssh`: vorherige Einrichtung ntowendig –> @fa [github]: @fa [question]
- Upload: nur authentifizierte Personen @fa [lock]
- –> @fa [github]: @fa [question]

@ulend

Authentification - Passwort

- bei jedem Push GitHub Passwort eingeben
- beachte: clone `https://...`

Authentification - SSH

- public key, private key cryptography (siehe z.B. Youtube)
- nur einmal einrichten ->
- clone `ssh://...`

Hands On 1.1: Gitkraken mit Github verbinden

@ol

- Gitkraken Profil (rechts oben)
- Preferences->Authentification->GitHub
- connect to GitHub
- Generate SSH key and add to GitHub

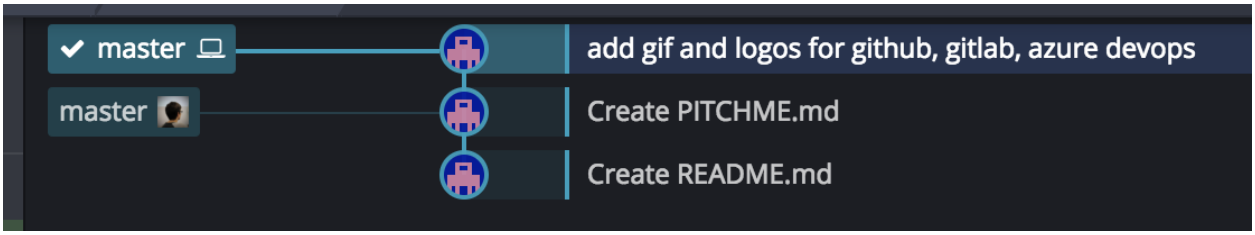

Figure 5: Git commits

@olend

Daten speichern - Add und Commit

Commit history

Go back in time!

Hands On 2 - Go back in time

@ol

- `reset master to this commit`
- spiele mit: `hard`, `mixed`, `soft`
- `fast forward master to origin/master` (oberster commit)

@olend

Commit

- Commit hält **Veränderungen** gegenüber dem vorherigen Commit fest
  - Änderungen von Dateien
  - Neuerstellung von Dateien
  - Löschung von Dateien
  - Umbenennung von Dateien
- ein Commit kann mehrere Änderungen beinhalten

Adding und Staging Area

(Source: https://git-scm.com/about/staging-area)

Figure 6: Time Machine

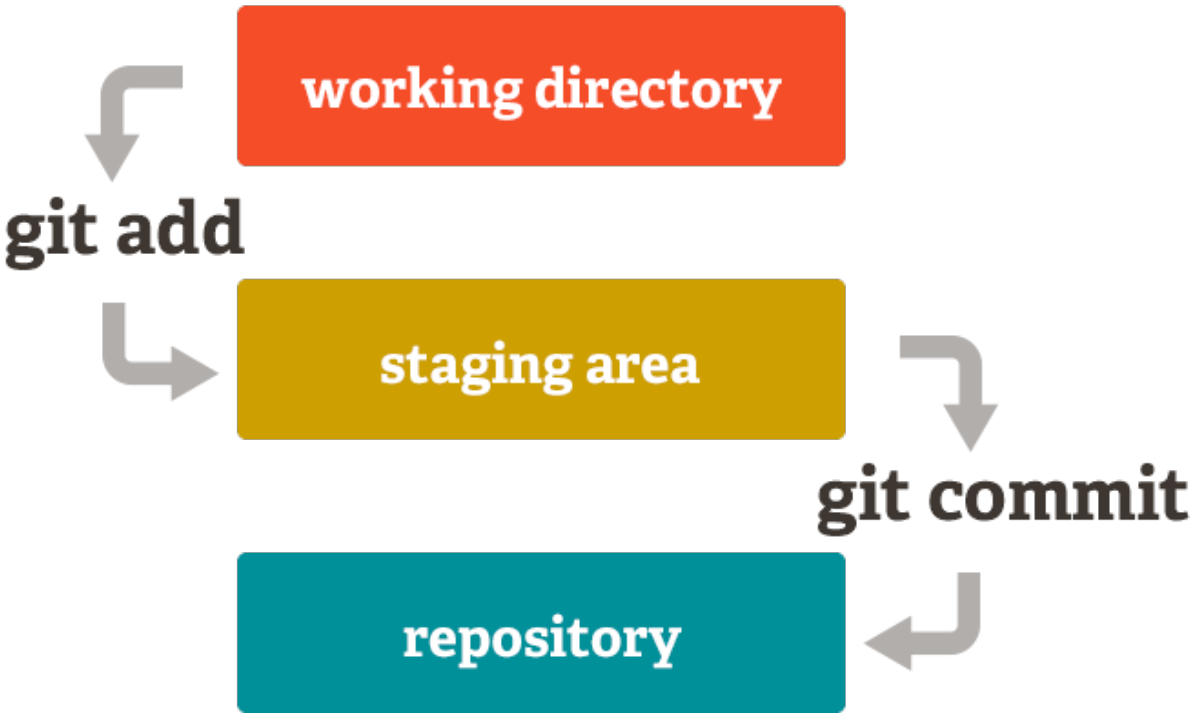

Figure 7: Staging Area

Hands On 3 - einen Commit machen

Grace + Mona

@ol

- Change stuff!
- **GIT ADD** von den "Unstaged Files" Dateien **GIT ADD**en, die man in Git "speichern" möchte.
- (halbwegs) aussagekräftige Commit Message schreiben
- **GIT COMMIT**

@olend

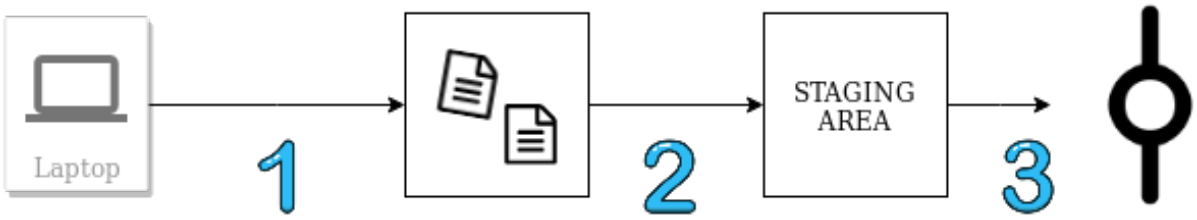

Figure 8: Lokal

Git quizzed!

Git quizzed!

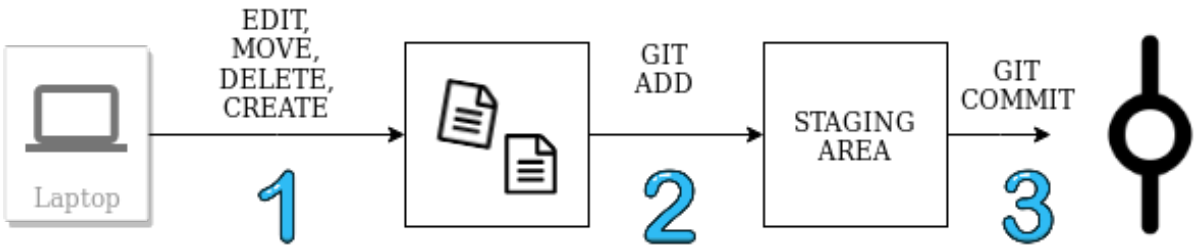

Figure 9: Lokal

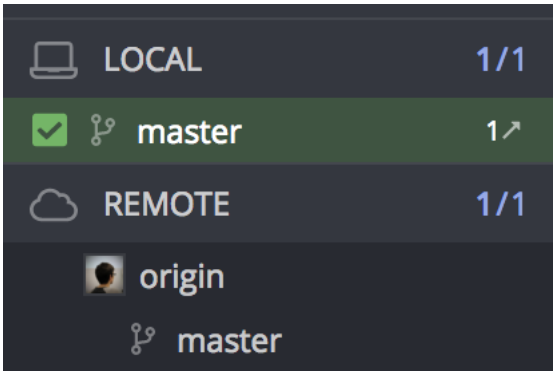

Figure 10: Gitkraken Lokal Remote

Daten syncen - Push und Pull

Git Hosting

- Die Cloud! z.B.

@fa [gitlab] @fa [github]

Git Lokal und Git Remote

. . . what?

Lokal: dein PC **Remote**: in der Cloud (GitHub, GitLab, . . .)

Sync: Git Pull und Git Push

- Git Pull: neue Commits von GitHub downloaden
- Git Push: lokal erstellte Commits nach GitHub hochladen

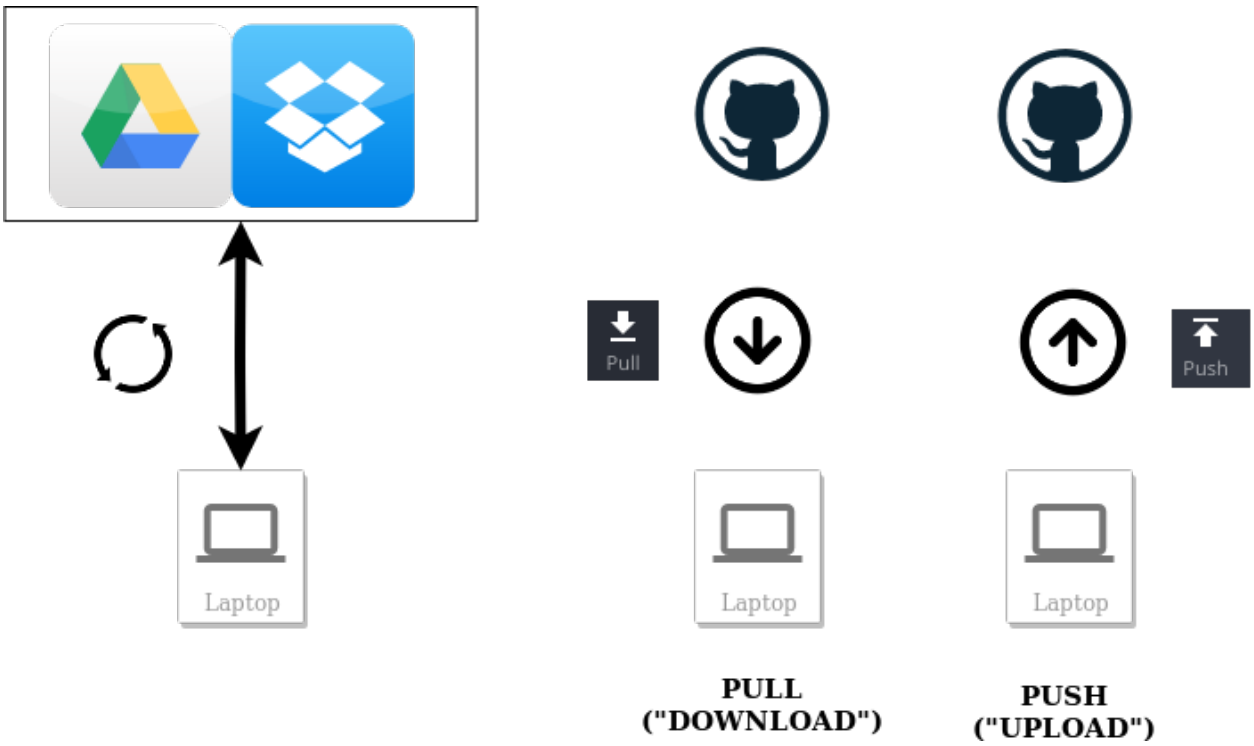

Figure 11: Push Pull

Sync: Git Pull und Git Push

Hands On 4 - Pull und Push

@ol

- Grace: Push
- Mona: Pull
- Mona: Push
- Grace: Pull

@olend

Git quizzed!

Git quizzed!

Figure 12: Push the button

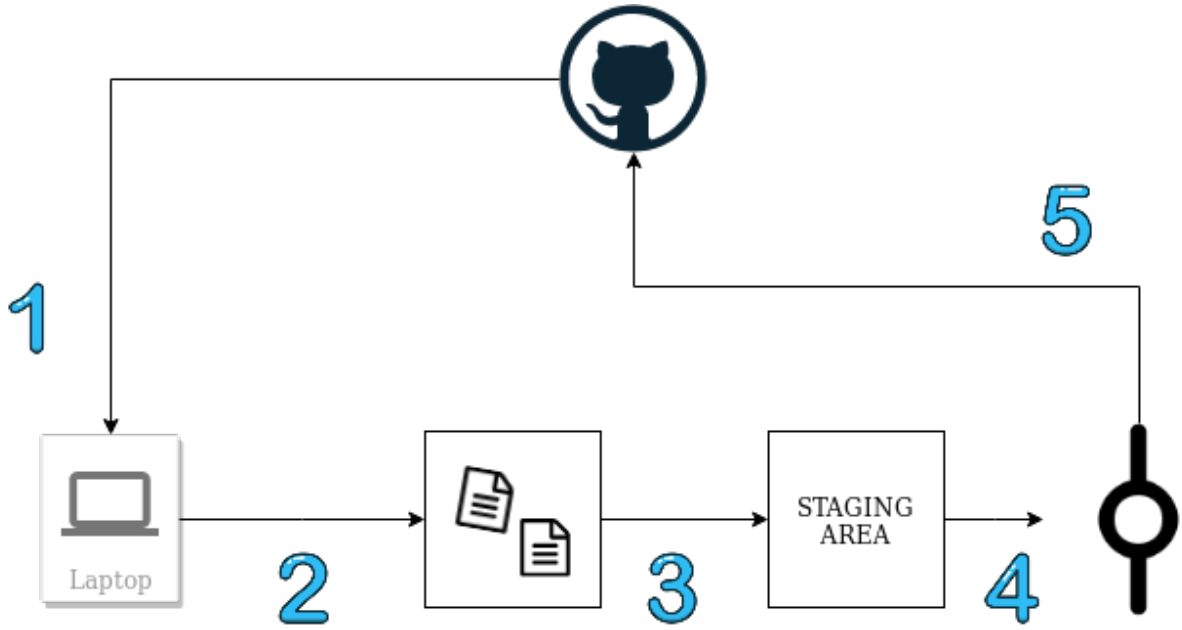

Figure 13: Push Pull

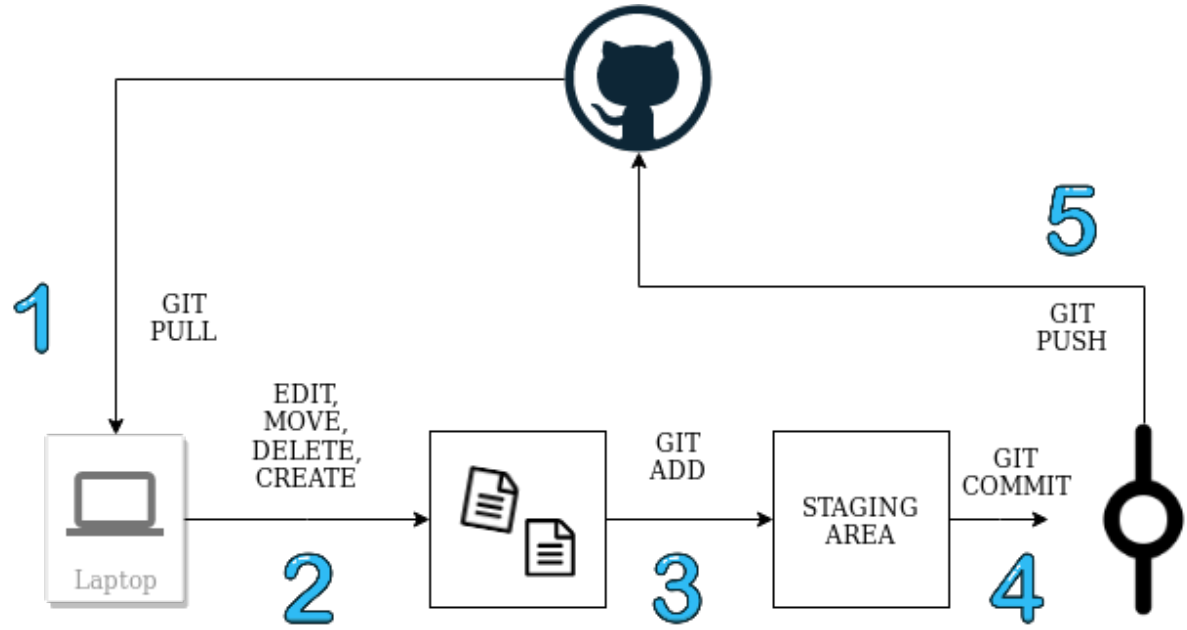

Figure 14: Push Pull

When things go wrong...

When things go wrong...

@ol

- so lange nichts gepusht ist, alles (halbwegs) gut
  - oft committen!
- zur Not: Codestand sichern und neu clonen

@olend

Git stash

@boxbg-blue text-black rounded

Git stash

–> put it away for now!

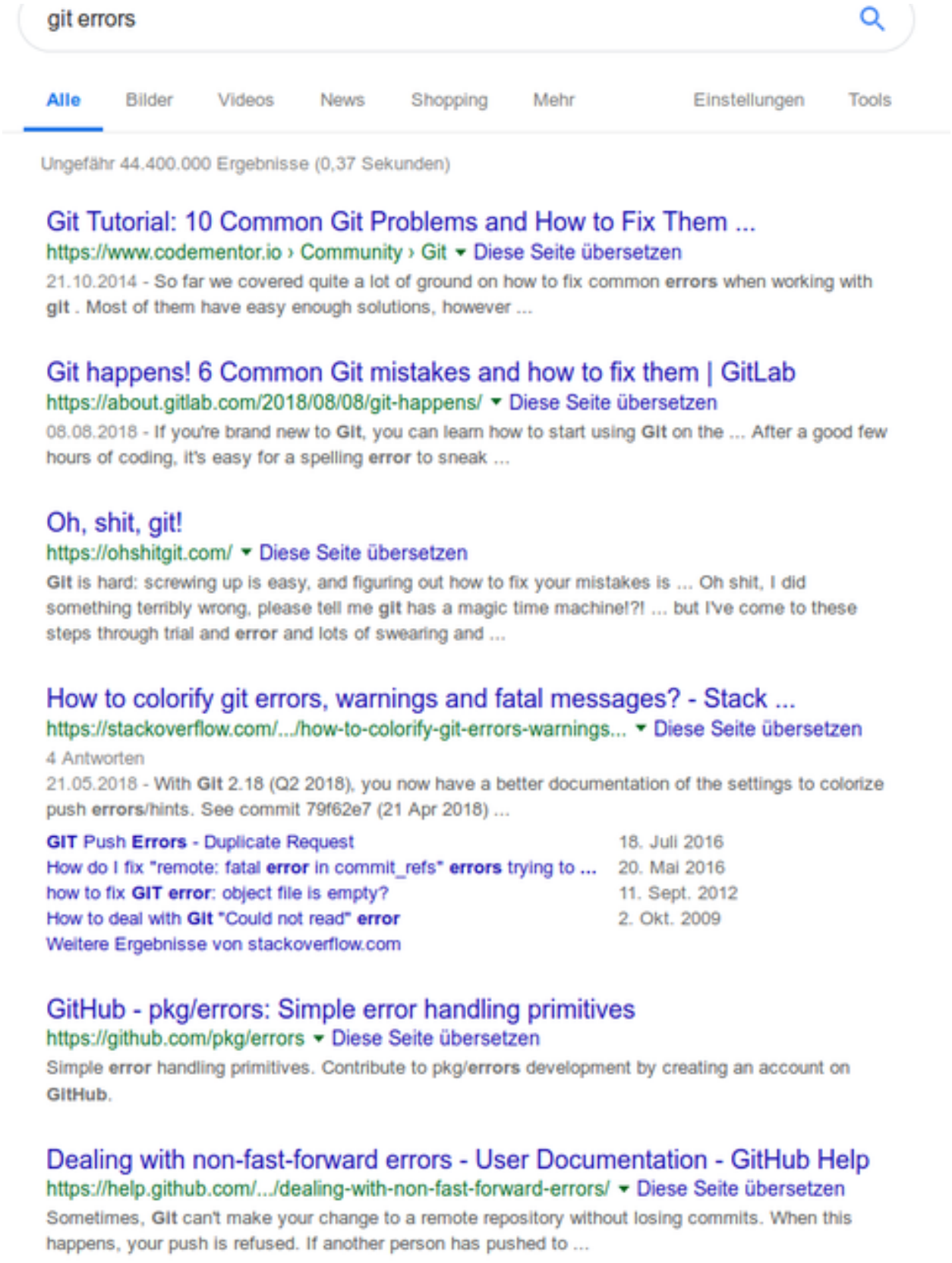

Figure 15: git google

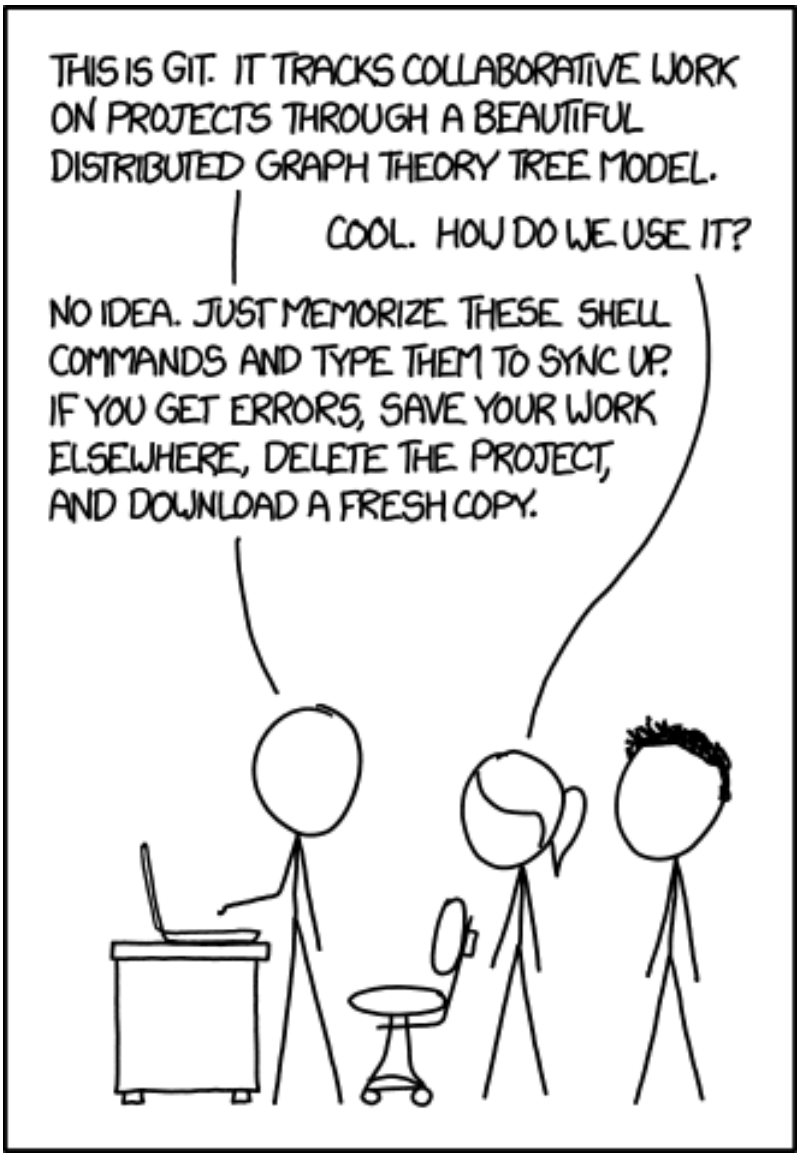

Figure 16: xccd

Figure 17: stash

Git stash bei merge conflicts

@ol

- git stash
- git pull
- apply stash
- solve merge conflicts
- (delete stash)

@olend

Hands on 5: Merge conflicts

---

mit GitHub arbeiten

Issues

@ul

- issues: Todos / Bugs / Ideen

- jeder issue hat eine Nummer
- #issueno in commit message verknüpft commit mit issue

@ulend

Hands On 4: Issue

@ol

- Mona: Issue erstellen: “Grace’s LieblingsGIF fehlt”
- Grace: füge der Präsentation eine neue Folie hinzu mit deinem Lieblingsgif (giphy -> copy link)
- Grace: add + commit. verlinke issue Nummer in der commit message (#issueno)
- Grace: push
- Mona: Issue neu laden (STRG+R)

@olend

Branches

[picture of complicated gitkraken with a lot of branches]

Branches

@boxbg-blue text-black rounded

Branches

@boxbg-blue text-black rounded

Why branches?

@ul

- “master” branch frei von unfertigem Code halten
- unabhängige Entwicklung von Code (“feature branches”)
- Experimente

@ulend

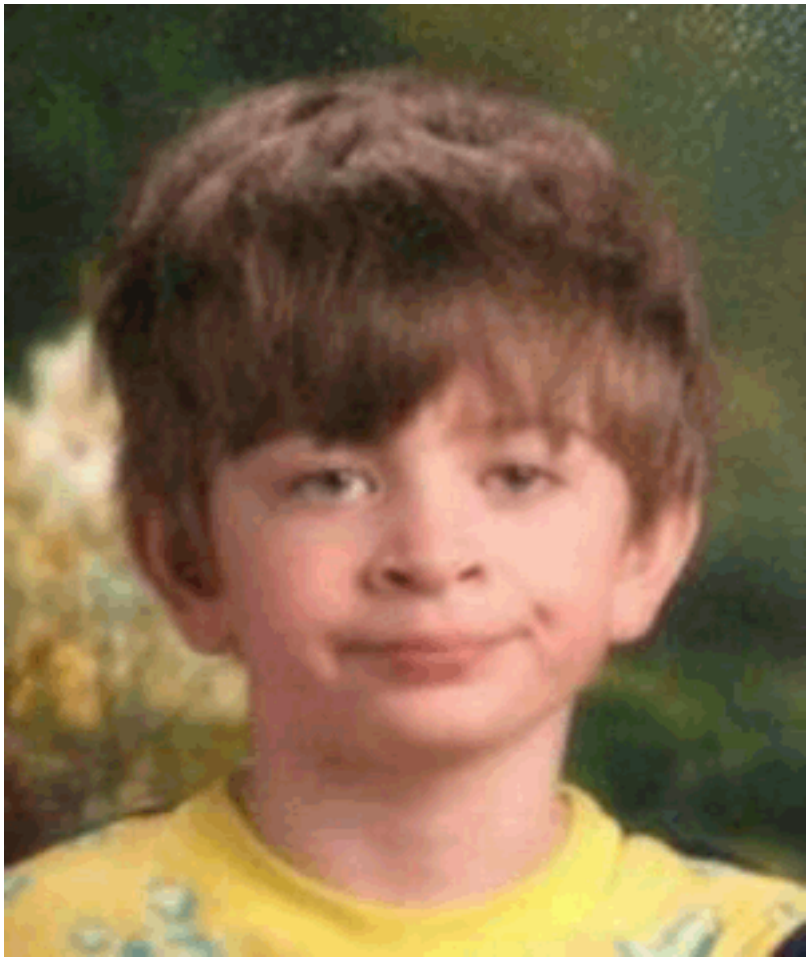

Figure 18: Come on

Branches Workflow

@ol

- Branch erstellen
- normal weiterarbeiten (pull-commit-push cycles)
- (optional: merge andere branches in deinen branch um Updates zu bekommen)
- merge Branch in master branch

@olend

Merging branches

- Rechtsclick auf branch name / master
- hängt davon ab, wer "weiter vorne" ist (?)
  - wenn neue commits auf master: merge master into #1-add-branch-slides -> branch wird geupdatet
  - wenn neue commits auf branch: merge #1-add-branch-slides into master -> master wird geupdatet

---

Branches Fazit

@ul

- besonders nützlich bei Kollaboration
- Entwicklung von Packages
- Relevanz für Datenprojekte (?) -test text

@ulend

Das wars.

gerne den Tag über fragen!