

# 2.

---

## Git, GitHub, Rust-Umgebung

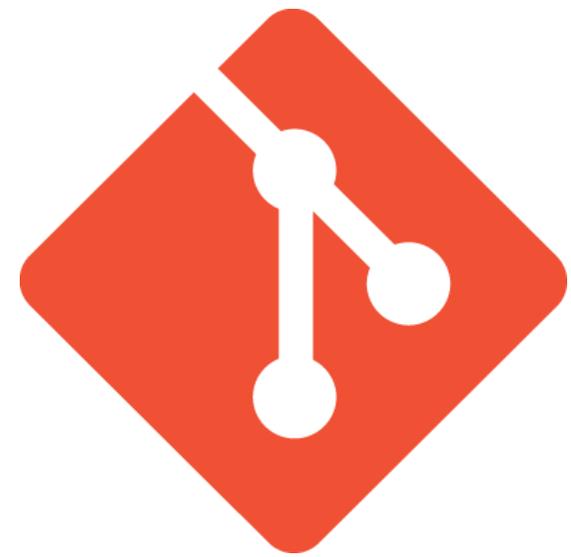
---

- Git
- GitHub
- Rust-Compiler einrichten



# git

Eine kurze Einführung



***Git*** /git/, noun  
a stupid or unpleasant man

— Oxford Dictionaries

- Verteiltes Versionsverwaltungssystem
- Open Source
- Extrem populär

Komplex & kompliziert,  
aber *mächtig!*

→ **Lohnt sich**

Companies & Projects Using Git

Google

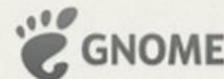
facebook

Microsoft

twitter

LinkedIn

NETFLIX



# Bedienung

- Original: im Terminal ([Download](#))
  - Nicht optimal, aber: komplette Macht von Git
  - Wird in dieser Präsentation benutzt
  - Und „überall im Internet“!
- *GitHub for Windows* ([Download](#))
  - Sehr einfach, hübsch und beschränkt
- *SourceTree* ([Download](#))
  - Wesentlich mehr Funktionen, komplizierter zu bedienen
- Empfehlung: erstmal im Terminal!

## Wichtig:

Ihr dürft nutzen, was ihr wollt, müsst aber die Anforderungen erfüllen können!

# Versionsverwaltung

VCS (Version Control System)

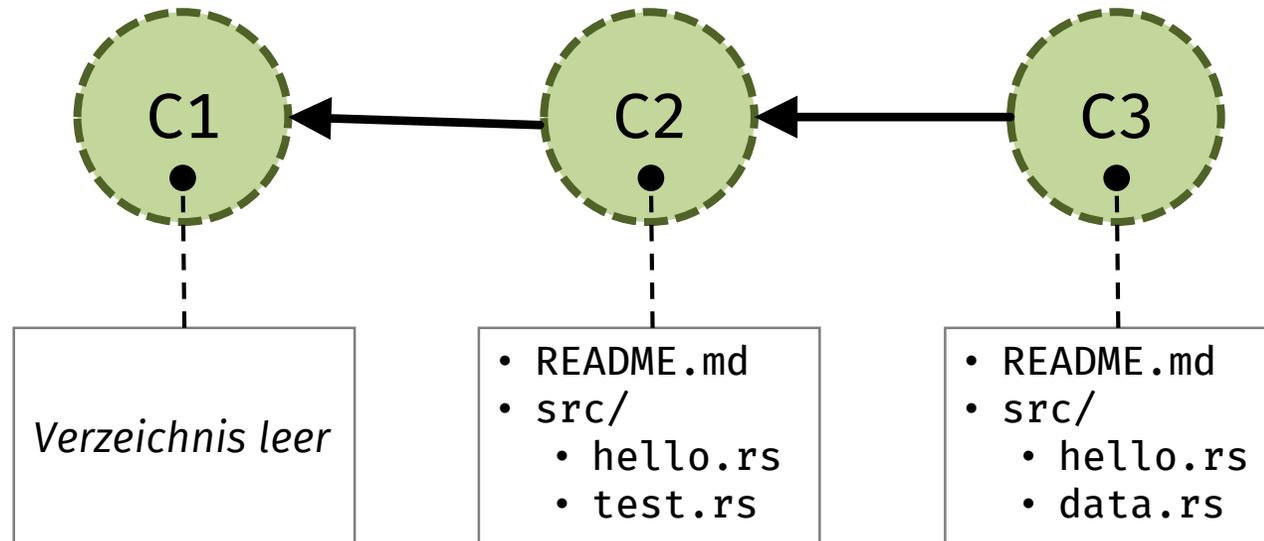
- Protokolliert Veränderungen an Dateien, mit ...
  - Autor
  - Kurzbeschreibung
  - Zeitstempel
- Ermöglicht ...
  - Wiederherstellung alter Zwischenstände
  - Paralleles Arbeiten mehrerer Entwickler
  - Analysen der Codebase

„Alternativen“: Mercurial, SVN, CVS, Perforce ...

# Ein Commit

- Abbildung des ganzen Projekts zu einem Zeitpunkt
- Ein Commit speichert:
  - Alle Dateien (komprimiert: Delta vom Vorgänger)
  - Seinen Vorgänger
  - Textnachricht, Autor, Zeit, ...
- Identifiziert durch Hash (hier: **C1**, **C2**, ...)

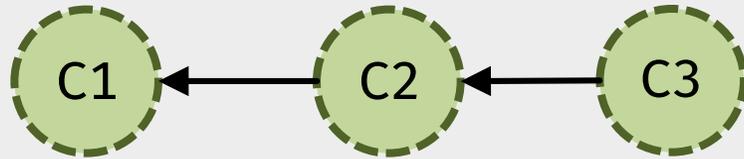
0cc9b3db1e693d42b25af02deb393dcff12865bc



# Das Repository

- Ordnerstruktur auf Dateisystem
- „Blick auf ein Commit“

Version Database:



checkout



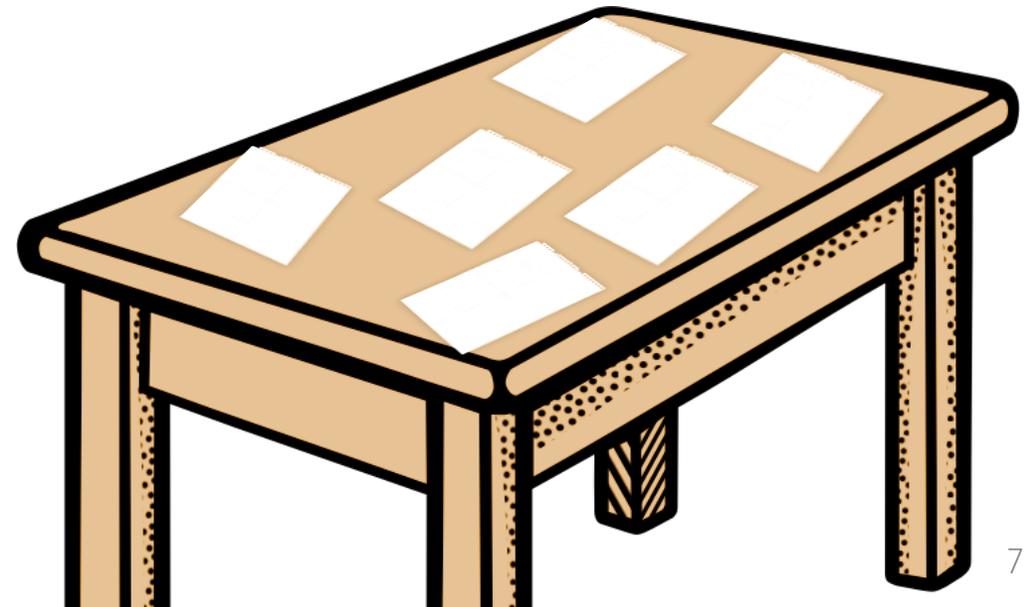
Working Directory:



- Enthält alle Commits
- Versteckt gespeichert in `.git/`



- Aktenordner holen
- Ordnernamen *merken!*
- Jede Seite kopieren
- Kopien auf Tisch ausbreiten



# Das Repository

`$ git commit`

„Staging Area“



`$ git add`

„Staging“

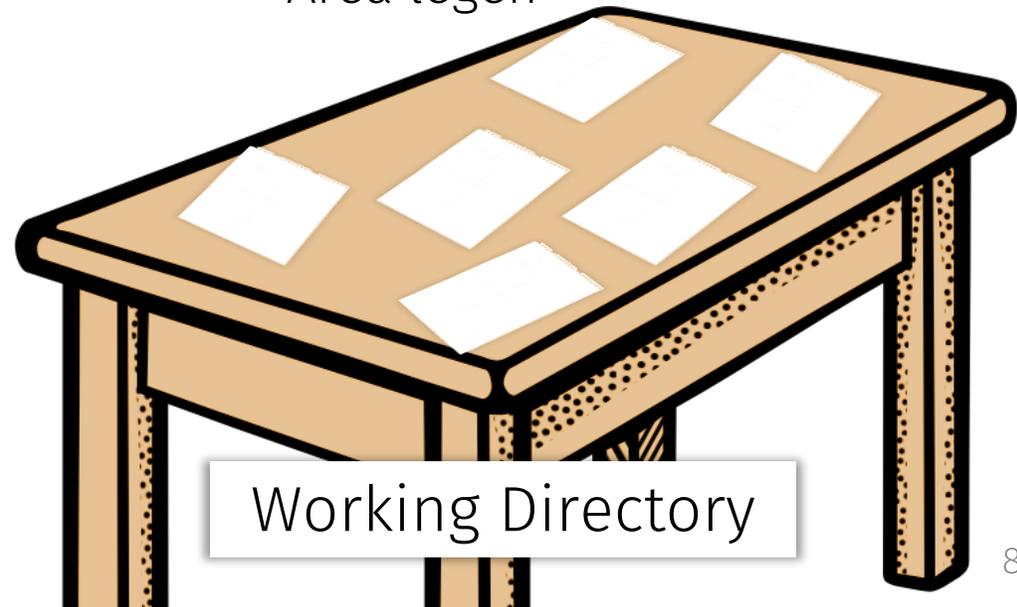
- Zettel kopieren
- Kopie in Staging-Area legen

- Originalordner kopieren
- In Kopie Seiten mit neuer Version aus der Staging Area überschreiben
- Neuen Ordner *merken*

- Aktenordner holen
- Ordnernamen *merken!*
- Jede Seite kopieren
- Kopien auf Tisch ausbreiten

`$ git checkout`

Working Directory



Version Database



# Dateistatus

Wurde kopiert und auf die Staging-Area gelegt

Staged

Datei wird Teil des nächsten Commits („Ich bin bereit!“)

Wurde auf dem Tisch verändert

Modified

Datei registriert und wurde seit letztem Commit verändert.

Frisch kopiert auf dem Tisch

Unmodified

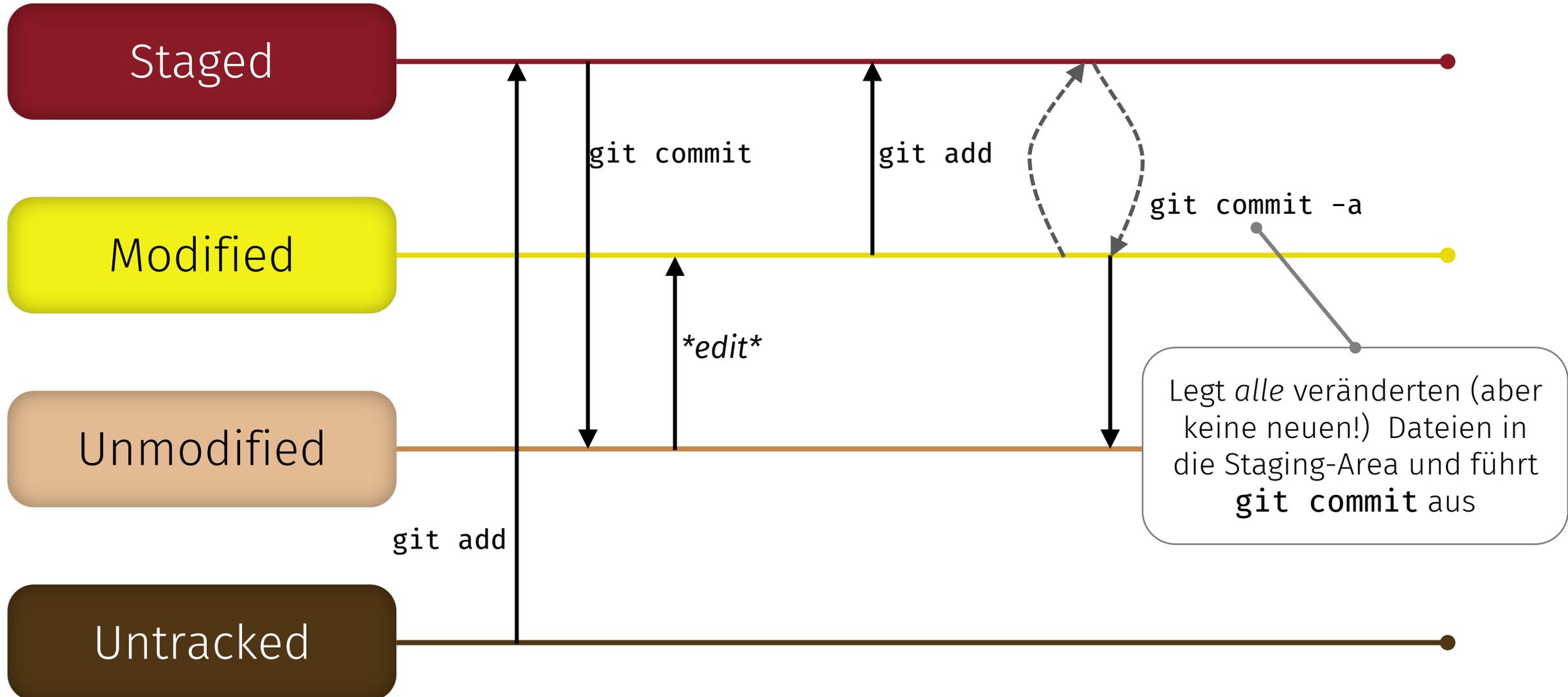
Datei in Git registriert, gleicht aber dem letzten Commit.

Jemand hat was auf den Tisch gelegt und niemand weiß Bescheid

Untracked

Git sieht Datei im Dateisystem, tut aber nichts mit ihr.

# Dateistatus



# Typischer Ablauf

1. Lege Datei **hello.rs** und **bye.rs** *neu* an
2. `git add hello.rs bye.rs`
-  3. `git commit`
4. *Bearbeite* einige Dateien
-  5. `git commit -a`
6. Lege Datei **test.rs** *neu* an
7. *Bearbeite* **hello.rs**
8. `git add test.rs`
-  9. `git commit -a`
10. ...

Beachte: Jeder benutzt  
git anders...

# Status anzeigen

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
```

```
new file: bye.rs
new file: hello.rs
```

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)
```

```
modified: Cargo.toml
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
test.rs
```

git status oft ausführen!  
Immer wissen, was los ist.

# Commit: Wie oft?

- Commit sollte *atomar* sein
- Projekt sollte kompilieren (keine groben Fehler)
- Unit-Tests sollten funktionieren



Bitte *oft*, aber *bewusst* committen!

**Nicht** `git commit -m`  
**nutzen!**

## **Commit Nachrichten** (Ordner Beschriftung)

- Zusammenfassung der Änderungen (nicht: „stuff“)
- Profi Tipps:
  - Erste Zeile sehr *kurze* Zusammenfassung
  - Optional: Durch leere Zeile getrennte weitere Beschreibung

# Repository erstellen/klonen

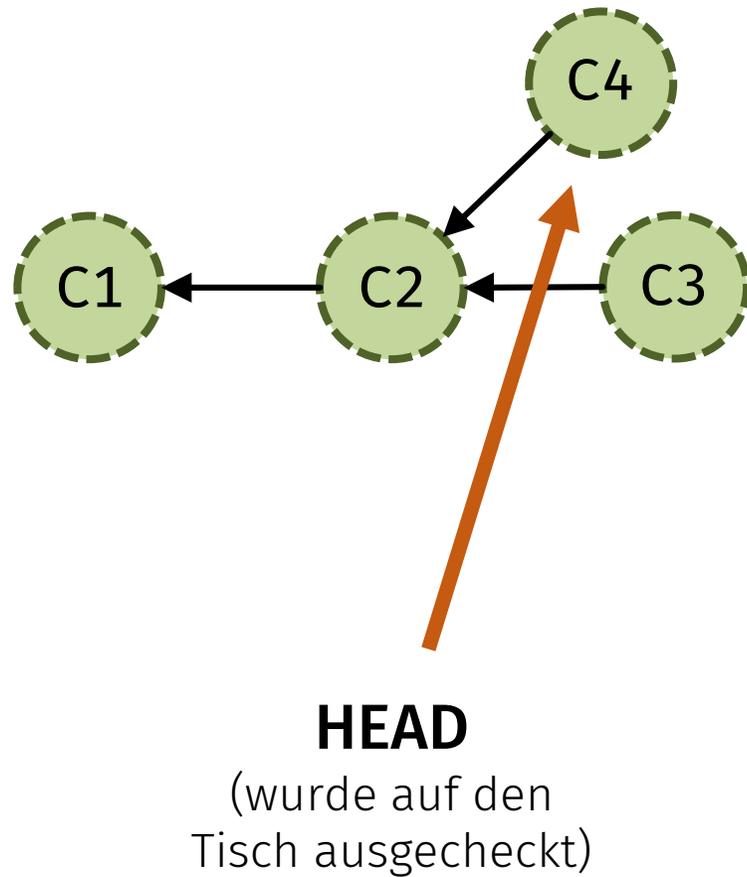
## `git init`

- Legt neues Repository an
- Leerer Tisch, leerer Aktenschrank

## `git clone REPO_URL`

- Kopiert Aktenschrank von angegebenem Repository
- Checkt speziellen Commit aus („legt auf Tisch“)
- Vermerkt Quellrepository als **origin**-Remote

# Einen Blick in den Aktenschrank..

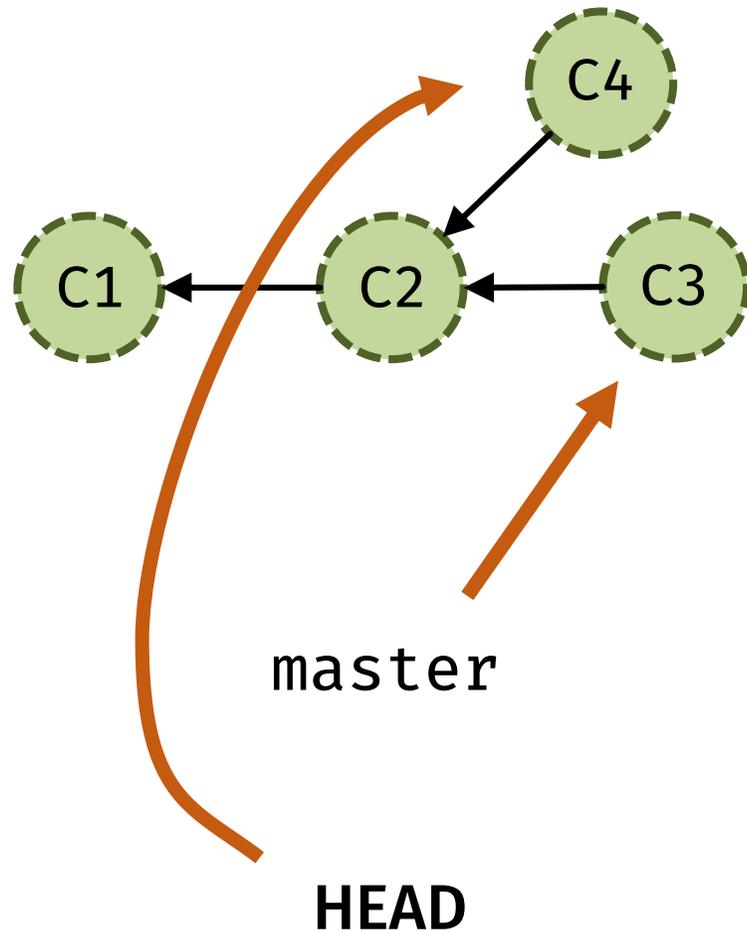


```
$ git commit  
$ git commit  
$ git commit  
$ git checkout C2  
$ git commit
```

- Ein Commit kann Vorgänger mehrerer anderer Commits sein

# Branches

- Pointer auf Commit
- Hat Namen



```
$ git commit  
$ git commit  
$ git commit  
$ git checkout C2  
$ git commit
```

```
$ git branch <name>
```

- Erzeugt neuen Branch
- Zeigt auf den Commit, auf den HEAD zeigt

```
$ git checkout -b <name>
```

- `git branch name` und
- `git checkout name`

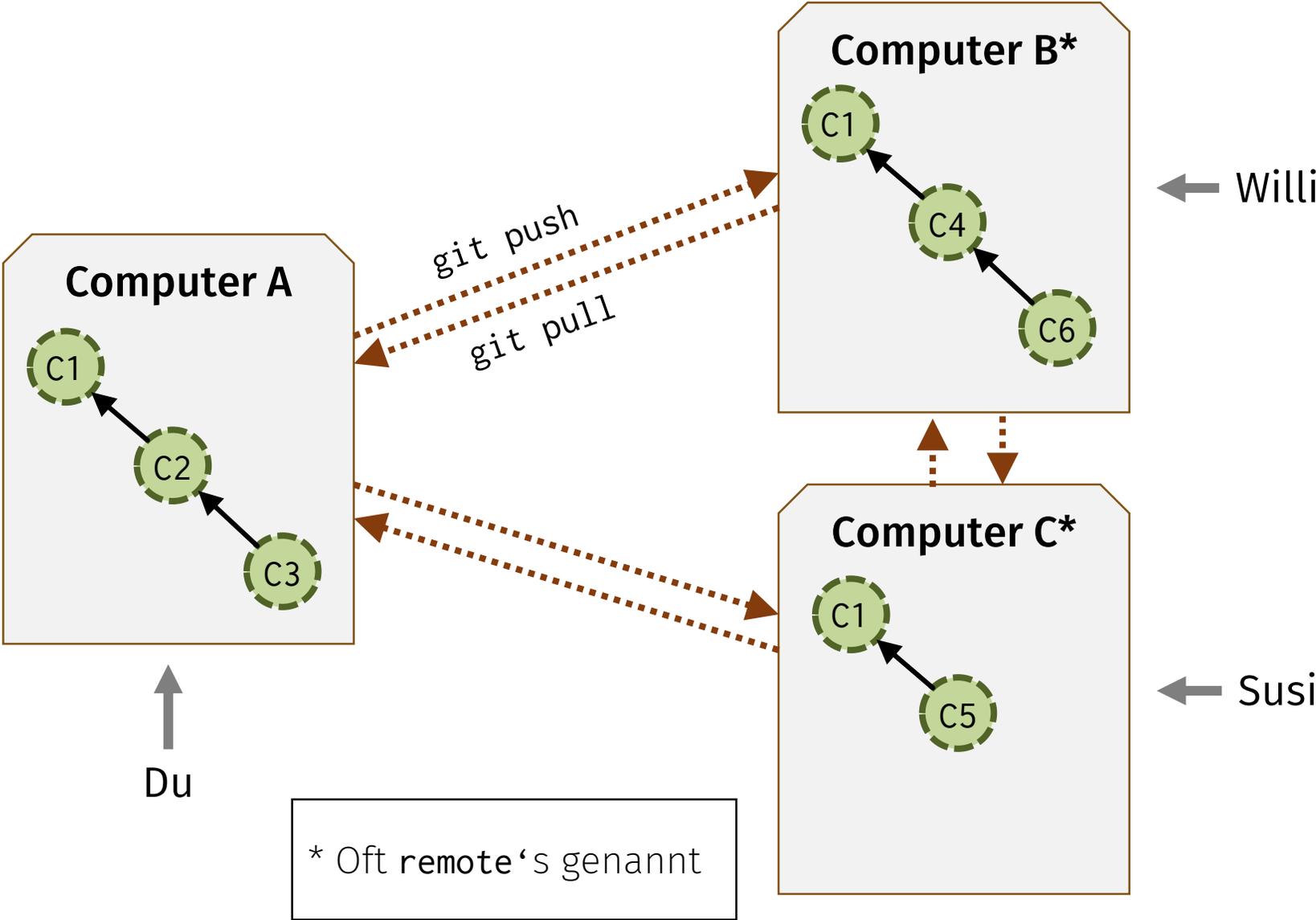
# Git commit im Detail

- Bereitet neuen Aktenordner vor
  - Alter Ordner (worauf HEAD zeigt)
  - Neue Inhalte aus der Staging-Area
- Vorgänger ist der Commit, auf den HEAD zeigt
- Branch, auf den HEAD zeigt, wird auf neuen Commit umgebogen

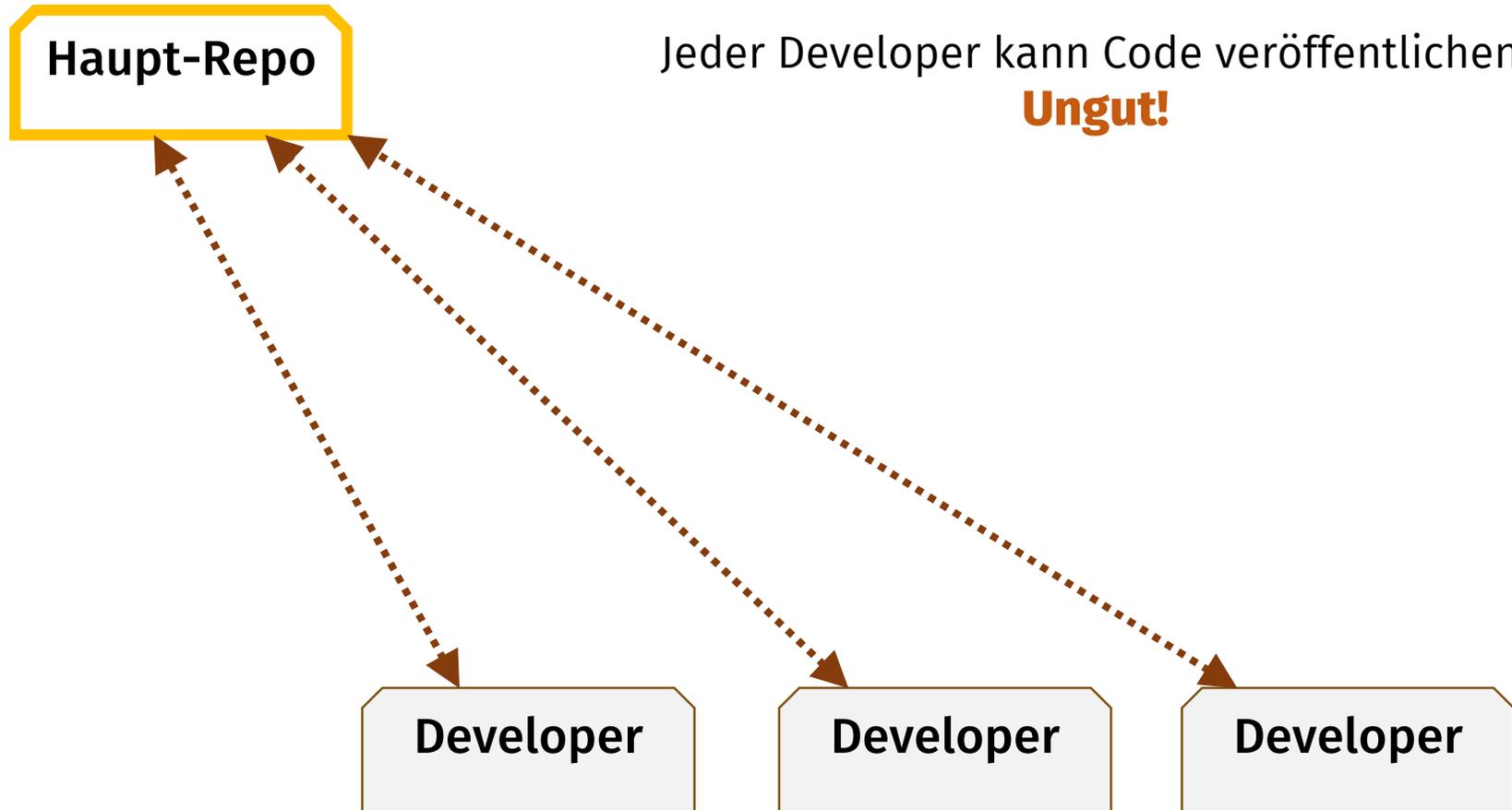
<http://learngitbranching.js.org/?NODEMO>

Das wars erstmal zu Git Branching...

# Verteiltes VCS

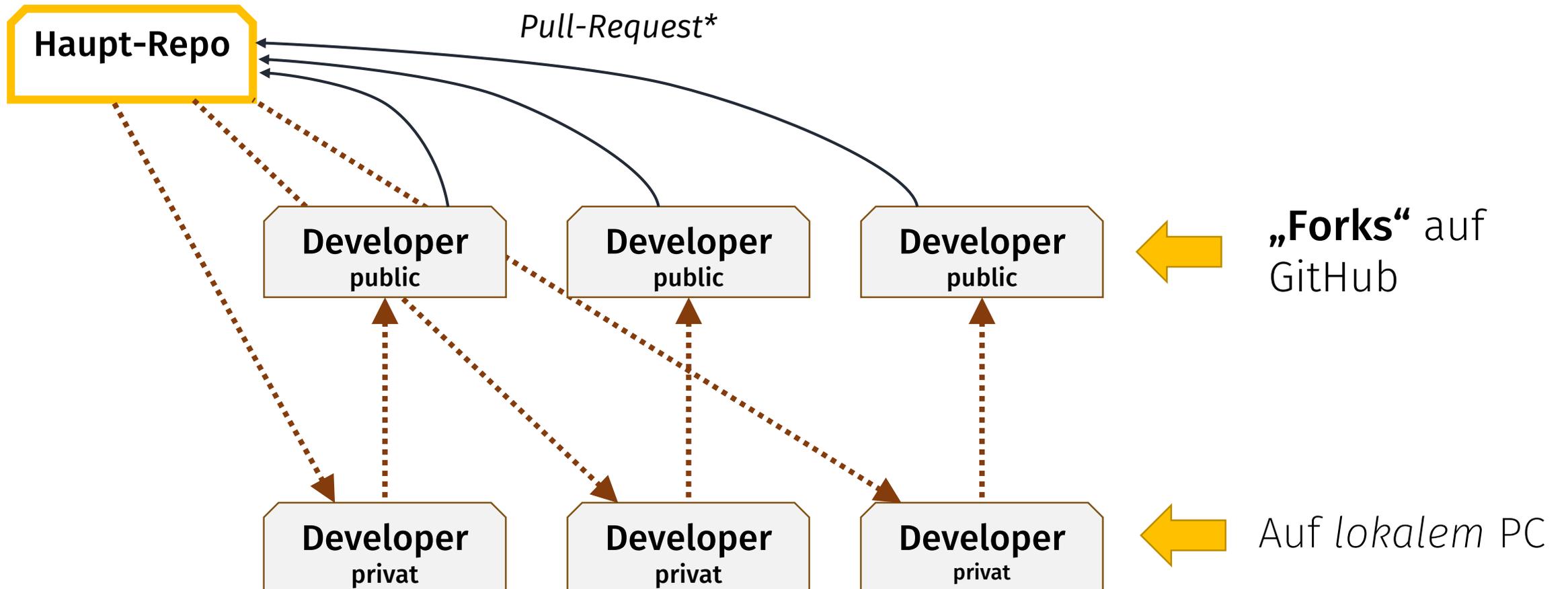


# Centralized Development

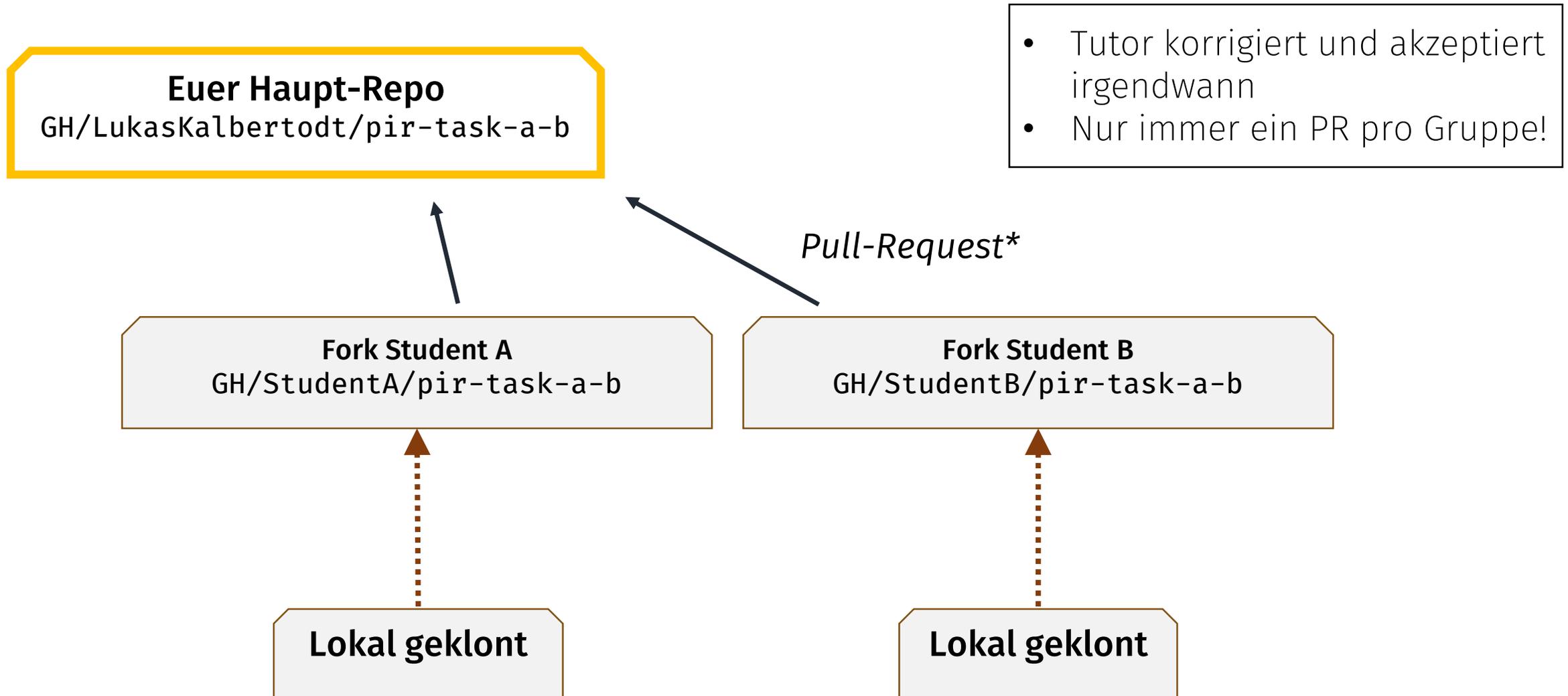


# Github Workflow

\* Vom Maintainer wird **Code Review** durchgeführt & **CI Dienste** werden ausgeführt



# In dieser Veranstaltung



# Ablauf

- Einer pro Gruppe schickt mir E-Mail (siehe Aufgabe 1)
- Ich lade euch zu Repo eurem privaten Repo ein
- Mindestens einer forkt originales Repository
- Wiederhole pro Woche:
  - Aufgaben bearbeiten, comitten und pushen
  - PR erstellen



# Rust einrichten

Compiler & Texteditor

# Voraussetzung

- Betriebssystem
  - Linux
  - OS X
  - Windows (eher wenig benutzt zurzeit)
- Codeeditor
  - Texteditoren: Sublime Text, vi, emacs, Atom, ...
  - IDEs: IntelliJ Rust, Visual Studio mit Visual Rust , ...
- CIP PCs: Werden noch eingerichtet!

## Wichtig:

Alle Editoren/Konfigurationen erlaubt, solange Richtlinien eingehalten werden:

- Keine unnötigen Dateien in git
- Unix-Zeilendenen
- UTF8 kodiert
- ...

# Compiler installieren

- Onlinecompiler: <https://play.rust-lang.org/>
- Via *rustup* (compiler version manager u.v.m.):

<https://rustup.rs/>

- Sollte **rustc 1.12** + **cargo** installieren
- Nach dem 10. November Update auf **rustc 1.13** möglich:
  - **\$ rustup update**

# Texteditor

- Sublime Text
  - Syntax Highlighting immer dabei
  - [„Rust Enhanced“](#) bringt zusätzliche Features
- Andere Editoren: <https://areweideyet.com/>
- Anleitung für Sublime:  
<https://github.com/LukasKalbertodt/programmieren-in-rust/wiki>