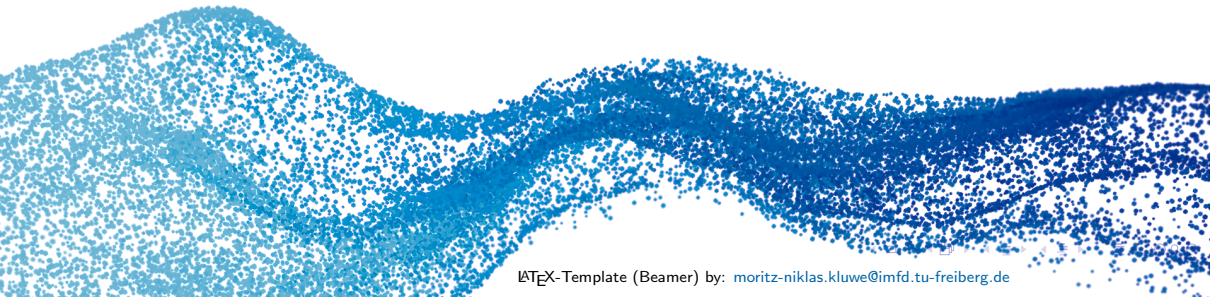
Git: Einführung

Dezentrale Versionsverwaltung



LaTeX-Template (Beamer) by: moritz-niklas.kluwe@imfd.tu-freiberg.de

Gliederung

1. Git $\neq$ GitHub, GitLab, ...
2. Begriffe/Befehle
3. Installation und Client einrichten
4. SSH - Secure Shell
5. Allgemeine Hinweise und Tipps
6. Merge, Rebase, Requests, Konflikte und co.

Git $\neq$ GitHub, GitLab, . . .

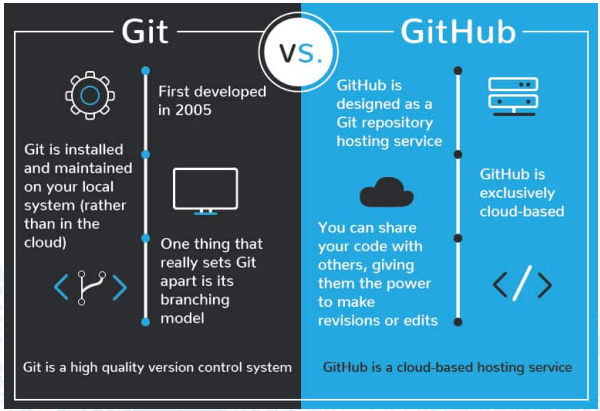


Figure: Unterschied zwischen Git und GitHub/GitLab. Quelle

Begriffe/Befehle

repository	"Container" mit allen gespeicherten Versionen
push	Änderungen zum Server hochladen
pull	Änderungen vom Server herunterladen
working area	Ordner im Filesystem in ausgewählter Version
staging area	Dateien müssen vor dem commit gestaged werden
stash	Änderungen werden gespeichert; dann auf Ausgangszustand
commit	neue "Version" wird angelegt
merge	zusammenführen von Branches
pull/merge-request	Anfrage an Repo-Verwaltende, einen Branch zu mergen
diff	Unterschied zwischen zwei Versionen anzeigen
checkout	Branch wechseln
log	Log

Begriffe/Befehle

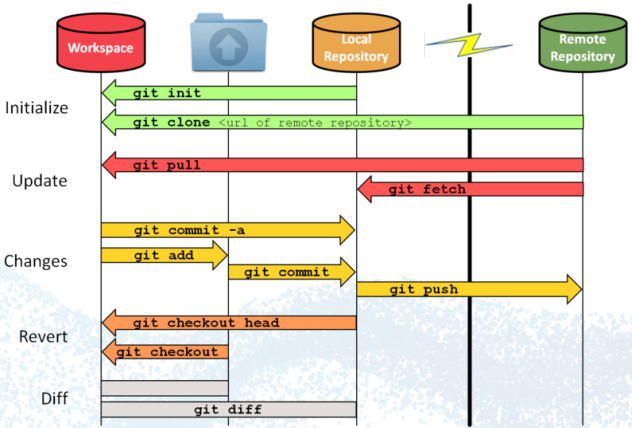


Figure: Remote und local Repository, working directory, staging area und die zugehörigen Befehle. Quelle

Begriffe/Befehle

Dateien können folgende Zustände haben:

- **Untracked:** Datei wird nicht verwaltet bzw. von Git ignoriert. Sollte für größere und binäre Dateien verwendet werden → eingestellt in `.gitignore` → Vorlagen auf GitHub (Bsp. .gitignore Python)
- **Tracked:** Git verfolgt Datei, ist jedoch nicht Staged
- **Staged:** Datei ist bereit committed zu werden
- **Modified/Dirty:** Die Datei ist bearbeitet aber die Änderungen sind noch nicht staged

Installation und Client einrichten

- Installation:
  - Linux: über Paketmanager
  - Windows: offizieller Download → Git-Bash wird mitgeliefert
  - macOS: über Paketmanager (bspw. `Homebrew`)
- GUIs:
  - `SourceTree` (Free), `GitKraken` (teils Free), uvm.
  - In VSCode: `GitLens`, diverse andere Extensions
- Konfiguration Git: `git config --global -e`
- Konfiguration Bash: `nano ~/.bashrc`
  - Vorlagen für .gitconfig und .bashrc als GitHub Gists

SSH - Secure Shell

- SSH wird zur Kommunikation zwischen Git und GitHub/GitLab empfohlen
- SSH benötigt ein Key-Pair, dessen öffentlicher Teil auf dem Host eingetragen wird
- Der private Teil des Key-Pairs **sollte immer** mit Passwort gesichert werden
  - Der private Key wird **NIE** weitergegeben oder dem Host mitgeteilt
- Key-Pair generieren: `ssh-keygen` → Optionen:
  - `-t`: Key-System/Target → Empfehlung: ed25519
  - `-C`: Comment → Frei wählbarer Kommentar, wird in den Keys mit hinterlegt
  - `-f`: File → Pfad und Name der Key-Pair-Dateien
- In Linux muss private-key Privat sein → `chmod 600 [ssh-file]`
- Auf dem Host unter "`Preferences` → `SSH-Keys`" wird dann der Inhalt von [ssh-file]**.pub** hinterlegt
- mit `ssh-add private_key` wird der Key dem SSH-Agent hinzugefügt → danach kann mit `git clone` das Repo geklont werden

Allgemeine Hinweise und Tipps

- Allgemein:
  - Dokumentation ist wichtig!
  - `git commit -m "Nachricht"` geht schnell, lässt aber wenig Platz für die Beschreibung des Commits
  - `git commit` öffnet einen Editor, in dem der Commit ausführlicher beschrieben werden kann
- Konventionen Branch-Benennung:
  - Abschnitt von Namen der Branches werden durch / getrennt
  - kurze Präfixe zur einfachen Identifikation: `feat` = feature; `fix` = bugfix; `junk` = experiment branch zum späteren löschen
  - Branch-Namen kurz halten
  - Beispiel: `feat/read_csv` oder `fix/TypeError_csv_reader`
- Beim Arbeiten mit mehreren Personen an einem Repository: **An einem Branch arbeitet nur eine Person!** & **Main/Master-Branch mergen sollte nur eine Person übernehmen!**

Allgemeine Hinweise und Tipps

- Allgemeines Vorgehen:
  - `git clone URL` → Klone Repo
  - `git pull` → Lade alle neuen Änderungen runter
  - `git branch -a` → Zeige alle Branches an
  - `git checkout main` → Wechsel zu main Branch
  - `git checkout -b feat/aufgabe_1` → Erstelle neuen Branch „feat/aufgabe_1“
    - ausgehend von aktuellem Branch → ggf. vorher zu entsprechenden Branch wechseln
  - `git add .` → Füge alle Dateien der Staging Area hinzu
  - `git status -s` → Kontrolliere Status aller Dateien
  - `git commit` → Commite alle Änderungen der Dateien in der Staging Area
  - `git push` → Lade neue Commits auf Server hoch

Merge, Rebase, Requests, Konflikte, ...

Merge: Zusammenführen von zwei oder mehr Branches

- Generell wird unterschieden zwischen zwei Arten von Merges:
  - `Fast-Forward-Merge`: Erstellt keinen neuen Commit; Standardoption von `git merge [branch]`
  - `3-Way-Merge`: Erstellt neuen Commit; `git merge --no-ff [branch]`
    → *no-Fast-Forward*
- Kleinere Features und Bug-Fixes werden oft mit `Fast-Forward-Merge` behandelt
- Größere Features werden oft mittels `3-Way-Merge` behandelt
- Wenn kein `Fast-Forward-Merge` möglich ist, kann auch erst `Rebase` angewendet werden, um den `Fast-Forward-Merge` möglich zu machen

Merge, Rebase, Requests, Konflikte, ...

3-Way-Merge: (weitere Beispiele auf Atlassian)

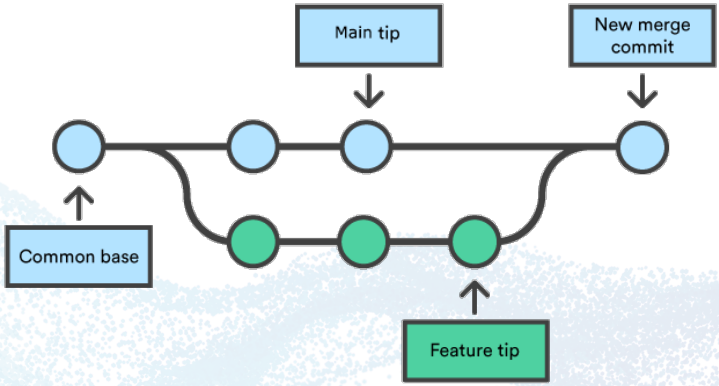


Figure: Quelle

Merge, Rebase, Requests, Konflikte, . . .

Rebase: Verschieben eines Branches auf einen neuen Basis-Commit

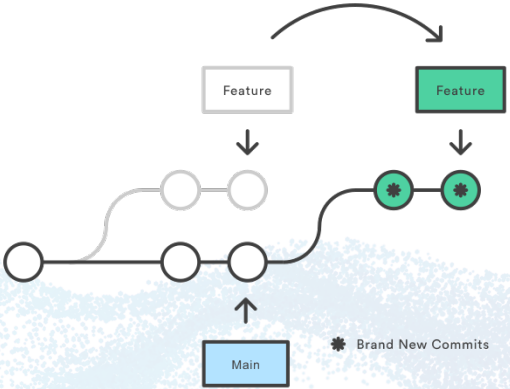


Figure: Quelle

Merge, Rebase, Requests, Konflikte, ...

Requests: Arbeiten mehrere Personen an einem Projekt, sollte der `main`-Branch geschützt sein. Nur eine Person (oder mehrere wenige Personen) sollten Branches in `main` mergen dürfen. Die anderen müssen dann Merge-/Pull-Requests stellen.

- Requests können direkt in GitLab/GitHub erstellt werden (Beispiel GitLab):
  - Im Repository in linker Menüspalte → `Merge Requests`
  - Entsprechend Source- und Target-Branch auswählen
  - Zusätzliche Informationen geben
- Repo-Admin muss dann den Merge-/Pull-Request bearbeiten
  - Konfliktfrei kann direkt gemerged werden
  - Treten Konflikte auf, muss manuell gemerged werden

Merge, Rebase, Requests, Konflikte, ...

Konflikte: Treten auf, wenn Git nicht selbstständig Branches Mergen oder Rebasen kann

- Standard Merge-Tool: vimdiff...
- VSCode kann als Mergetool genutzt werden
  - Dafür muss `git config --global -e` angepasst werden (Link)
- Mit `git mergetool` wird VSCode mit dem problematischen File geöffnet
  - unten rechts auf *Im Merge-Editor öffnen* klicken
  - oben links: Source; oben rechts: Target; Unten: Merge
  - mit Haken auswählen, was behalten wird
  - abschließen *Merge akzeptieren* → VSCode kann wieder geschlossen werden
  - `git merge --continue`
- `git mergetool` kann ebenso bei Rebase-Konflikten benutzt werden

Merge, Rebase, Requests, Konflikte, ...

Sammlung wichtiger Befehle:

`git branch`	list branches
`git checkout [branch]`	switch to branch
`git checkout -b [branch]`	initiate and switch new branch
`git merge [branch]`	merge [branch] into current branch
`git rebase [branch]`	rebase current branch onto [branch]
`git merge --continue`	continue merge (after solving conflicts)
`git rebase --continue`	continue rebase (after solving conflicts)
`git mergetool`	helping tool for manual merging/rebasing
`git status`	file status
`git checkout [branch] [file]`	choose which conflicting file to use
`git log`	show log