Git-Spickzettel: Die wichtigsten Befehle für den Einstieg

9. September 2025

Inhaltsverzeichnis

1 Erste Einrichtung (einmalig)

- Benutzeridentität setzen (für Commits):

```
git config --global user.name  "Dein Name"
git config --global user.email "du@example.com"
```

- Standard-Branch auf `main` setzen (empfohlen):

```
git config --global init.defaultBranch main
```

- Optional Editor konfigurieren (Beispiel VS Code):

```
git config --global core.editor "code --wait"
```

- Aktuelle Konfiguration anzeigen:

```
git config --list --show-origin
```

2 Repository anlegen oder klonen

- Neues Repo im aktuellen Ordner starten:

```
git init
```

- Existierendes Repo klonen:

```
git clone git@github.com:USER/REPO.git
# oder per HTTPS:
git clone https://github.com/USER/REPO.git
```

3 Status, Historie & Unterschiede ansehen

- Kurzstatus:

```
git status -sb
```

- Historie kompakt und mit Branch-Graf:

```
git log --oneline --graph --decorate --all
```

- Unterschiede (Arbeitsverzeichnis vs. Index):

```
git diff          # nicht gestagte nderungen
git diff --staged # Unterschiede zu gestagten nderungen
```

4 Dateien vormerken (`add`) & speichern (`commit`)

- Alles (inkl. Löschungen) zum nächsten Commit vormerken:

```
git add -A    # oder: git add --all
```

- Einzelne Datei/Ordner stagen:

```
git add pfad/zur_datei.txt
git add ordner/
```

- Commit erstellen:

```
git commit -m "Aussagekrftige Commit-Nachricht"
```

- Letzten Commit bearbeiten (z.B. vergessene Datei oder Message):

```
git commit --amend
```

5 Dateien verschieben/umbenennen und entfernen

- Verschieben/Umbenennen (tracked Dateien):

```
git mv alter/pfad/datei.txt neuer/ordner/datei.txt
```

- Entfernen und im nächsten Commit löschen:

```
git rm pfad/zur_datei.txt
```

- Nur aus dem Tracking nehmen (Datei lokal behalten):

```
git rm --cached pfad/zur_datei.txt
```

- Nur Groß-/Kleinschreibung ändern (case-insensitive Dateisysteme):

```
git mv Name.txt __temp__ && git mv __temp__ name.txt
```

6 Branches: erstellen, wechseln, zusammenführen

- Branches anzeigen (mit letztem Commit):

```
git branch -vv
```

- Neuen Branch erzeugen und wechseln:

```
git switch -c feature-x      # modern
# (alt) git checkout -b feature-x
```

- Zwischen Branches wechseln:

```
git switch main
# (alt) git checkout main
```

- Branch `feature-x` in `main` mergen:

```
# zuerst auf main wechseln und aktualisieren
git switch main
git pull --rebase origin main
# dann mergen
git merge feature-x
```

- Konflikte lösen: betroffene Dateien editieren, testen, dann

```
git add <konfliktdatei>
git commit
```

- (Optional, fortgeschritten) Rebase statt Merge in Feature-Branch:

```
git switch feature-x
git fetch origin
git rebase origin/main
# bei Konflikten: Dateien lsen -> git add ... -> git rebase
--continue
```

7 Mit Remotes arbeiten: Push/Pull/Fetch

- Remote prüfen/setzen:

```
git remote -v
git remote add origin git@github.com:USER/REPO.git
# URL ndern:
git remote set-url origin git@github.com:USER/NEUES-REPO.git
```

- Erster Push (Upstream setzen):

```
git push -u origin main
```

- Spätere Pushes:

```
git push                  # auf den jeweils verknpften
Upstream
git push origin feature-x
```

- Aktuellen Stand holen (ohne zu mergen):

```
git fetch origin
```

- Holen und integrieren (rebase empfohlen):

```
git pull --rebase origin main
```

8 Änderungen verwerfen & zurückrollen

- **Nicht gestagte** lokale nderungen verwerfen:

```
git restore pfad/datei.txt    # einzelne Datei
git restore .                  # alles im Arbeitsverzeichnis
```

- Datei aus dem Index entstagen (Änderungen behalten):

```
git restore --staged pfad/datei.txt
```

- Datei auf Stand von `HEAD` zurücksetzen:

```
git restore --source=HEAD -- pfad/datei.txt
```

- Einzelnen Commit rückgängig machen (Historie bleibt linear):

```
git revert <commit-hash>
```

- Auf früheren Stand **hart** zurücksetzen (Achtung!):

```
git reset --hard <commit-hash>
# 'soft' behlt nderungen gestagt, 'mixed' (Standard) behlt
sie ungestagt
```

9 Zwischenspeichern mit `stash`

- Schnell Zwischenspeichern und Arbeitsverzeichnis leeren:

```
git stash push -m "WIP: kurze Beschreibung"
# kurz: git stash
```

- Liste anzeigen / wieder anwenden:

```
git stash list
git stash apply    # nderungen bleiben im Stash erhalten
git stash pop      # nderungen anwenden und Stash-Eintrag
entfernen
```

10 Versionen markieren: `tag`

- Leichtgewicht-Tag:

```
git tag v1.0.0
```

- Annotiertes Tag (mit Message, Autor, Datum):

```
git tag -a v1.0.0 -m "Erstes Release"
```

- Tags pushen:

```
git push --tags
# oder gezielt:
git push origin v1.0.0
```

11 `.gitignore` Basics

- Typische Einträge in `.gitignore`:

```
# Logs und temporres
*.log
*.tmp
.DS_Store

# Abhngigkeiten/Builds
node_modules/
dist/
build/

# Geheimnisse
.env
.env.*
```

- Prüfen, warum eine Datei ignoriert wird:

```
git check-ignore -v pfad/datei
```

12 Nützliche Aliases (optional)

```
git config --global alias.st "status -sb"
git config --global alias.co "checkout"
git config --global alias.sw "switch"
git config --global alias.ci "commit"
git config --global alias.br "branch -vv"
git config --global alias.lg "log --oneline --graph --decorate --all"
git config --global alias.df "diff"
```

13 Häufige Mini-Workflows

1. **Dateien in Unterordner verschieben (inkl. Tracken):**

```
git mv alte_datei.txt neuer_ordner/
git add -A
git commit -m "Verschiebe Dateien in Unterordner"
git push
```

2. **Neuen Branch erstellen und veröffentlichen:**

```
git switch -c feature-x
git push -u origin feature-x
```

3. **Merge-Konflikt lösen (Kurzform):**

```
# Konfliktdateien bearbeiten
git add <datei1> <datei2>
git commit   # Abschluss-Commit
```

4. **Aus Versehen committete Datei wieder entfernen, lokal behalten:**

```
git rm --cached geheim.txt
echo "geheim.txt" >> .gitignore
git commit -m "geheim.txt aus dem Tracking entfernt"
git push
```

Tipp: Verwende `git help <befehl>` oder `git <befehl> --help` für die eingebaute Hilfe.