

Die vier Orte, an denen Git deine Arbeit aufbewahrt — und was jeder Befehl genau zwischen ihnen kopiert



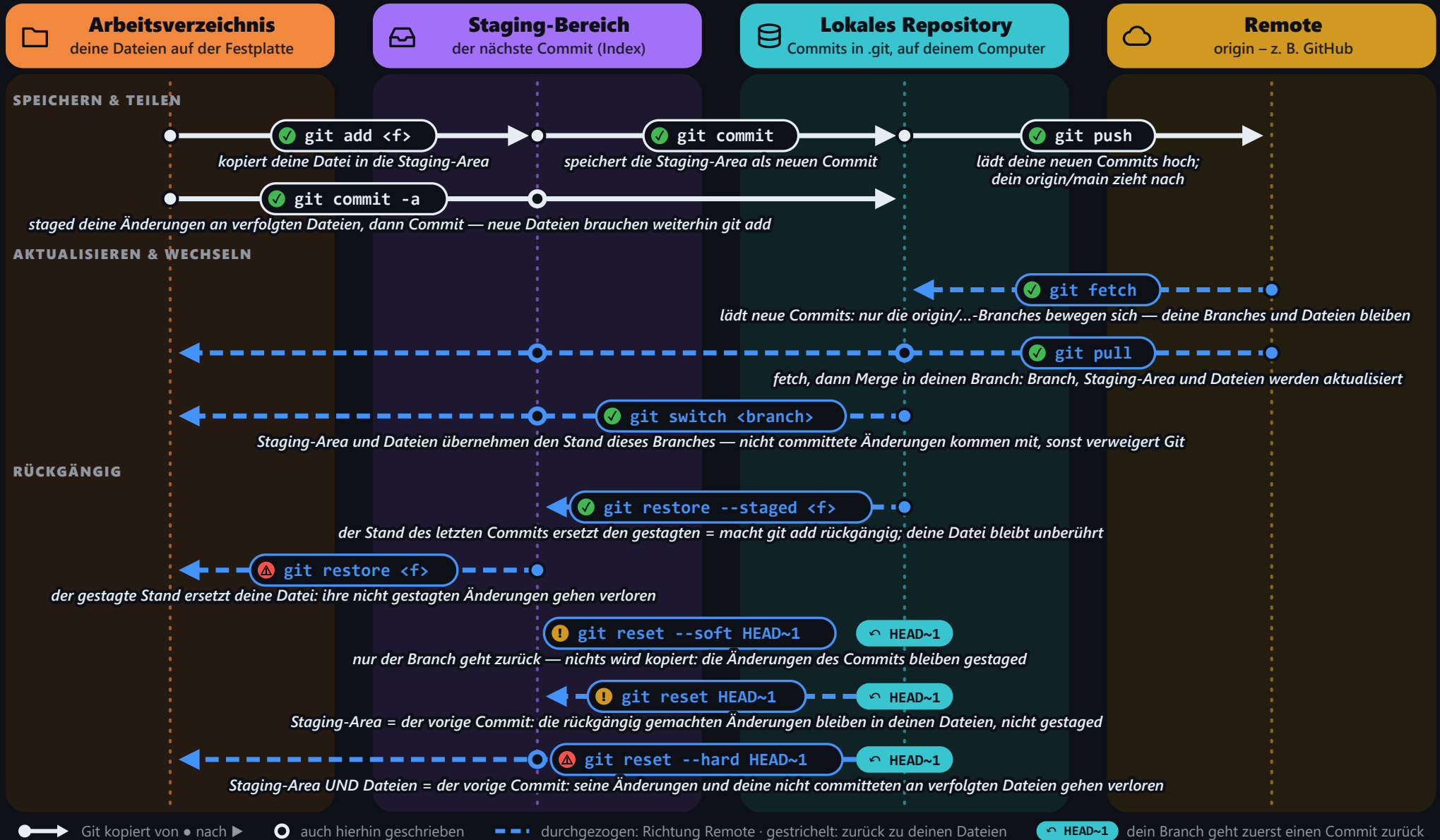
Sicher



Schreibt Historie um



Kann Arbeit kosten



o **<f>** ein Dateipfad: tippe **app.js**, ohne die < >

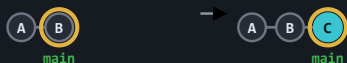
o **HEAD~1** der Commit vor dem, auf dem du bist

o **git pull** wenn du und das Remote beide neue Commits haben, hält Git an, bis du wählst: **git config pull.rebase false** (Merge) oder **true** (Rebase)

Vorher → nachher. Kreise sind Commits, der gelbe Ring zeigt, wo du bist (HEAD).

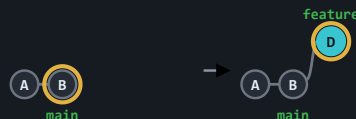
● neuer Commit ○ kein Branch zeigt mehr darauf

### ✓ git commit



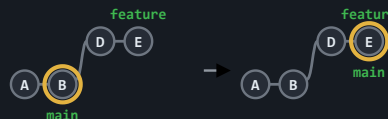
Ein neuer Commit aus dem Vorgemerkten; dein Branch rückt darauf vor.

### ✓ git switch -c feature + commit



Ein neuer Branch beginnt, wo du bist; deine nächsten Commits bewegen ihn, main bleibt.

### ✓ git merge feature (fast-forward)



Auf main, wenn main keine neuen Commits hat: main rückt einfach bis E vor — kein Merge-Commit.

### ✓ git merge feature



Auf main: führt feature mit einem Merge-Commit mit zwei Eltern zusammen.

### ! git rebase main



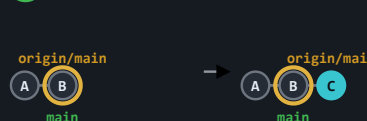
Auf feature: spielt D und E auf main neu ab, als Kopien mit neuen Hashes (D' E'); die Originale verlassen den Branch.

### ✓ git cherry-pick D



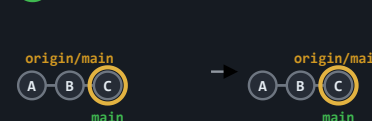
Kopiert einen Commit auf deinen Branch (D', neuer Hash); das Original bleibt.

### ✓ git fetch



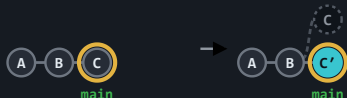
Lädt neue Commits: nur die origin/...-Labels bewegen sich. Dein Branch und deine Dateien bleiben.

### ✓ git push



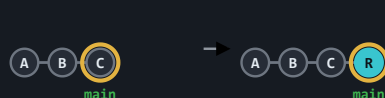
Lädt C hoch: main auf dem Remote — und dein origin/main — springen auf C. Deine Dateien ändern sich nicht.

### ! git commit --amend



Ersetzt den letzten Commit durch einen neuen (C', neuer Hash); C verlässt den Branch. Vor dem Push erledigen.

### ✓ git revert C



Fügt einen Commit R hinzu, der C rückgängig macht. Die Historie bleibt: sicher auf geteilten Branches.

### ! git reset --soft HEAD~1



Setzt deinen Branch auf B zurück; die Änderungen von C bleiben gestaged — erneut committen, um ihn neu zu machen.

### ⚠ git reset --hard HEAD~1



Setzt deinen Branch zurück; Staging-Area und Dateien werden zu B: die Änderungen von C sind weg (git reflog findet C noch).

## START

- git config --global user.name "Your Name"**  
Name, der in deinen Commits steht (einmal pro Rechner)
- git config --global user.email "you@example.com"**  
E-Mail, die in deinen Commits steht
- git init**  
Den aktuellen Ordner zu einem Repository machen
- git clone <url>**  
Ein Repository mit seiner ganzen Historie herunterladen

## UMSEHEN

- git status**  
Was sich geändert hat, was gestagt ist, auf welchem Branch du bist
- git diff**  
Noch nicht gestagte Änderungen (Arbeitsverzeichnis vs. Staging)
- git diff --staged**  
Was der nächste Commit enthalten wird
- git log --oneline --graph --all**  
Historie jedes Branches, eine Zeile pro Commit
- git show <commit>**  
Ein Commit: Nachricht und Änderungen

## ARBEIT SPEICHERN

- git add <file>**  
Eine Datei für den nächsten Commit stagen
- git add -p**  
Nur ausgewählte Teile (Hunks) stagen
- git commit -m "message"**  
Das Gestagte als Commit festhalten
- git commit --amend**  
⚠ Den letzten Commit neu machen (Nachricht oder Inhalt)
- git restore <file>**  
⚠ Nicht gestagte Änderungen einer Datei verwerfen
- git restore --staged <file>**  
Aus dem Staging nehmen (deine Änderung bleibt auf der Festplatte)

## TEILEN

- git fetch**  
Neue Commits herunterladen; deine Branches bewegen sich nicht
- git pull**  
Fetch, dann Merge in den aktuellen Branch
- git pull --rebase**  
⚠ Fetch, dann deine Commits obendrauf neu abspielen (neue Hashes)
- git push**  
Deine Commits zum Upstream des Branches hochladen
- git push -u origin <branch>**  
Erster Push eines neuen Branches (setzt den Upstream)
- git remote -v**  
Mit welchen Remotes dieses Repository spricht

- o **HEAD** der Commit, auf dem du bist (meist die Spitze deines aktuellen Branches)
- o **origin** der Standardname des Remotes, von dem du geklont hast
- o **<file>** ein Dateipfad: tippe **app.js**, ohne die **< >**

## ANLEGEN & WECHSELN

- git branch**  
Lokale Branches auflisten (\* = aktueller)
- git branch -a**  
Auch Remote-Tracking-Branches auflisten
- git switch <branch>**  
Zu einem Branch wechseln (Dateien folgen)
- git switch -c <branch>**  
Hier einen Branch anlegen und zu ihm wechseln
- git switch -**  
Zurück zum vorherigen Branch
- git branch -m <new-name>**  
Den aktuellen Branch umbenennen

## ZUSAMMENFÜHREN

- git merge <branch>**  
Einen Branch in den aktuellen holen
- git merge --no-ff <branch>**  
Immer einen Merge-Commit erzeugen
- git rebase <branch>**  
! Deine Commits auf <branch> neu anwenden (neue Hashes)
- git rebase -i HEAD~3**  
! Die letzten 3 Commits bearbeiten: umsortieren, zusammenfassen, umformulieren, verwerfen
- git cherry-pick <commit>**  
Einen Commit auf den aktuellen Branch kopieren

## KONFLIKTE

- git status**  
Listet die Dateien im Konflikt auf (z. B. „both modified“)
- Die Datei bearbeiten**  
Behalte, was du willst, und lösche die Zeilen  
<<<<<< ===== >>>>>>
- git add <file>**  
Die Datei als gelöst markieren
- git merge --continue**  
Abschließen (oder: git rebase --continue / cherry-pick --continue)
- git merge --abort**  
Aufgeben und zum Zustand davor zurück (rebase -abort, ...)

## AUFRÄUMEN

- git branch -d <branch>**  
Einen gemergten Branch löschen (verweigert, wenn nicht gemergt)
- git branch -D <branch>**  
⚠ Auch ungemergt löschen
- git push origin --delete <branch>**  
⚠ Löscht den Branch auf dem Remote (nicht gemergte Commits darauf gehen mit)
- git fetch --prune**  
Löscht auch die origin/...-Kopien von Branches, die auf dem Remote gelöscht wurden

## TAGS

- git tag**  
Tags auflisten
- git tag -a v1.0 -m "Release 1.0"**  
Annotierter Tag auf dem aktuellen Commit
- git push origin v1.0**  
Tags werden nicht automatisch gepusht
- git push --follow-tags**  
Commits samt ihren annotierten Tags pushen

## HISTORIE LESEN

- git log --oneline**  
Kompakte Historie des aktuellen Branches
- git log -p <file>**  
Jede Änderung an einer Datei
- git log main..feature**  
Commits auf feature, die main nicht hat
- git blame <file>**  
Wer jede Zeile zuletzt geändert hat

| ICH MÖCHTE ...                                                                  | AUSFÜHREN                                                       | GUT ZU WISSEN                                                                                                           |
|---------------------------------------------------------------------------------|-----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| ○ Meine Änderungen an einer Datei verwerfen                                     | <code>git restore &lt;file&gt;</code>                           | ⚠ Nicht gestagte Änderungen sind endgültig weg                                                                          |
| ○ Eine Datei aus dem Staging nehmen (Änderung behalten)                         | <code>git restore --staged &lt;file&gt;</code>                  | ✓ Nur der Staging-Bereich ändert sich                                                                                   |
| ○ Nachricht des letzten Commits korrigieren oder eine vergessene Datei ergänzen | <code>git add &lt;file&gt; &amp;&amp; git commit --amend</code> | ! Neuer Hash – vor dem Push erledigen                                                                                   |
| ○ Letzten Commit rückgängig, Änderungen bleiben gestagt                         | <code>git reset --soft HEAD~1</code>                            | ! Deine Arbeit bleibt gestaged, bereit für einen neuen Commit — nicht nach dem Push                                     |
| ○ Letzten Commit rückgängig, Änderungen bleiben in den Dateien                  | <code>git reset HEAD~1</code>                                   | ! Standard --mixed: nicht gestaged, aber auf der Festplatte — nicht nach dem Push                                       |
| ○ Einen bereits gepushten Commit rückgängig machen                              | <code>git revert &lt;commit&gt;</code>                          | ✓ Fügt einen neuen Commit hinzu, der ihn aufhebt – sicher auf geteilten Branches                                        |
| ○ Meine Änderungen vorerst beiseitelegen                                        | <code>git stash push -m "wip"</code>                            | ✓ Zurück mit <code>git stash pop</code> — neue, nicht verfolgte Dateien brauchen -u                                     |
| ○ Den letzten Commit samt Änderungen verwerfen                                  | <code>git reset --hard HEAD~1</code>                            | ⚠ Seine Änderungen und deine nicht committeten an verfolgten Dateien gehen verloren (git reflog findet den Commit noch) |
| ○ Einen „verlorenen“ Commit finden (nach reset, rebase, gelöschtem Branch)      | <code>git reflog</code>                                         | ✓ Jede Stelle, an der HEAD war – dann <code>git branch rescue &lt;hash&gt;</code>                                       |
| ○ Einen laufenden merge / rebase / cherry-pick abbrechen                        | <code>git merge --abort</code>                                  | ✓ Oder <code>rebase --abort</code> , <code>cherry-pick --abort</code>                                                   |
| ○ Den Remote-Branch nach dem Umschreiben der Historie ersetzen                  | <code>git push --force-with-lease</code>                        | ! Verweigert, wenn sich der Remote-Branch seit deinem letzten Fetch geändert hat                                        |

## STASH

### git stash

Nicht committete Änderungen beiseitelegen

### git stash -u

Nicht getrackte (neue) Dateien einschließen

### git stash list

Alle Stashes, neuester = `stash@{0}`

### git stash pop

Wendet den neuesten wieder an und entfernt ihn (bleibt bei einem Konflikt erhalten)

### git stash apply stash@{1}

Einen wieder anwenden und in der Liste behalten

## RESET – DREI STÄRKEN

### git reset --soft <commit>

! Verschiebt den Branch; Staging und Dateien unberührt

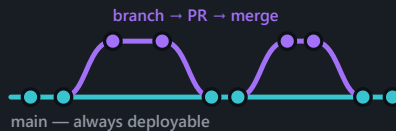
### git reset <commit>

! Verschiebt den Branch und setzt das Staging zurück; Dateien unberührt

### git reset --hard <commit>

⚠ Verschiebt den Branch, das Staging UND die Dateien

## GITHUB FLOW



```
git switch main &&
git pull
```

Von einem aktuellen main starten

```
git switch -c
<feature>
```

Ein kurzlebiger Branch pro Änderung

```
git commit -am "... " &&
git push -u origin
<feature>
```

Committen, pushen, Pull Request öffnen

*Den Pull Request auf GitHub mergen*

Nach dem Review; danach den Branch löschen

```
git switch main &&
git pull --prune
```

Zurück zu main; löscht auch die origin/...-Kopien gelöschter Branches

## GIT FLOW



```
git checkout -b
feature/x develop
```

Features starten von develop

```
git checkout develop
&& git merge --no-ff
feature/x
```

Ein Feature in develop abschließen

```
git checkout -b
release-1.1 develop
```

Ein Release einfrieren

```
git checkout main &&
git merge --no-ff
release-1.1
```

Release nach main ...

```
git tag -a 1.1 -m
"Release 1.1"
```

... taggen und zurück in develop mergen

## TRUNK-BASED



```
git commit -am "small
change"
```

Kleine, vollständige Commits auf main

```
git pull --rebase
```

⚠ Spielt deine auf denen der anderen neu ab (neue Hashes)

```
git add <file> && git
rebase --continue
```

Nach dem Lösen eines Konflikts: als gelöst markieren, weitermachen

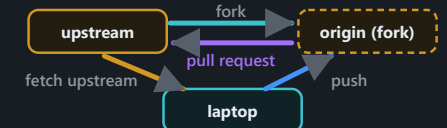
```
git push
```

Mindestens einmal am Tag integrieren

*Unfertige Arbeit abgeschaltet ausliefern*

Hinter einem Feature-Flag, z. B. NEW\_CHECKOUT = false

## FORKING



```
git remote add
upstream
```

```
<original-url>
```

Einmalig: das Original als zweites Remote

```
git fetch upstream &&
git merge
upstream/main
```

Deinen main mit dem Original synchronisieren

```
git push
```

Deinen Fork (origin) aktualisieren

```
git switch -c <fix> ...
git push -u origin
<fix>
```

Auf einem Branch in deinem Fork arbeiten

*Einen Pull Request zum Original öffnen*

Maintainer prüfen und mergen