

Git kennenlernen

Anfänger

Erste Schritte

Ein Repository einrichten

Änderungen speichern

Ein Repository  
überprüfen

Änderungen rückgängig  
machen

git checkout

git clean

git revert

git reset

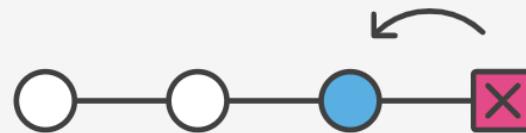
git rm

Verläufe umarbeiten

Zusammenarbeit

Migration zu Git

Tipps für  
Fortgeschrittene



# Commits und Änderungen rückgängig machen

**git checkout / git clean / git revert / git reset / git rm**

In diesem Abschnitt behandeln wir die verfügbaren Git-Strategien und -Befehle zum Rückgängigmachen von Änderungen.

Zuallererst ist zu beachten, dass Git nicht über eine herkömmliche "Rückgängig"-Funktion wie in einem Textverarbeitungsprogramm verfügt. Git-Abläufe sollten also eher nicht mit dem Denkmodell eines herkömmlichen Systems zum Rückgängigmachen von Änderungen verglichen werden. Außerdem hat Git seine eigene Nomenklatur für dies Abläufe des Rückgängigmachens, die im Folgenden sinnvollerweise auch verwendet werden sollten. Diese Nomenklatur umfasst Begriffe wie reset, revert, checkout, clean usw.

Spaßeshalber kann Git als ein Managementtool für Zeitachsen betrachtet werden. Commits sind Snapshots eines Zeitpunkts oder von wichtigen Punkten auf der Zeitachse eines Projektverlaufs. Zusätzlich können mithilfe von Branches mehrere Zeitachsen verwaltet werden. Rückgängigmachen in Git heißt also normalerweise, dass die Zeit zurückgedreht wird oder zu einer Zeitachse gewechselt wird, auf der der Fehler nicht stattgefunden hat.

Dieses Tutorial vermittelt dir alle Kompetenzen, die du benötigst, um mit älteren Überarbeitungen eines Softwareprojekts zu arbeiten. Zunächst zeigen wir dir, wie du alte Commits untersuchen kannst, und erklären dann den Unterschied zwischen dem Rückgängigmachen öffentlicher Commits im Projektverlauf und dem Zurücksetzen nicht veröffentlichter Änderungen auf deinem lokalen Computer.

## Verlorenes wiederfinden: Durchsehen alter Commits

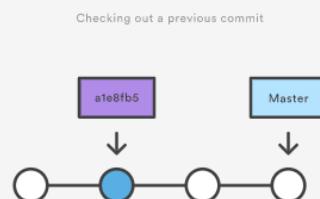
Der grundlegende Sinn und Zweck eines Versionskontrollsystems ist die Speicherung "sicherer" Kopien eines Projekts. Das heißt: Du

musst dir keine Sorgen mehr machen, dass deine Codebasis irreparabel geschädigt wird. Ist erst einmal ein Projektverlauf der Commits erstellt, kannst du jeden Commit im Verlauf erneut aufrufen und prüfen. Eines der besten Dienstprogramme zum Überprüfen des Verlaufs eines Git-Repositorys ist der Befehl `git log`. Im unten gezeigten Beispiel rufen wir mit `git log` eine Liste der letzten Commits für eine gängige Open-Source-Grafikbibliothek ab.

```
git log --oneline
e2f9a78fe Replaced FlyControls with OrbitControls
d35ce0178 Editor: Shortcuts panel Safari support.
9dbe8d0cf Editor: Sidebar.Controls to Sidebar.Settings
05c5288fc Merge pull request #12612 from TyLindberg/ed
0d8b6e74b Merge pull request #12805 from harto/patch-1
23b20c22e Merge pull request #12801 from gam0022/impro
fe78029f1 Fix typo in documentation
7ce43c448 Merge pull request #12794 from WestLangley/c
17452bb93 Merge pull request #12778 from OndrejSpanel/
b5c1b5c70 Merge pull request #12799 from dhritzkiv/pat
1b48ff4d2 Updated builds.
88adbcdf6 WebVRManager: Clean up.
2720fb08 Merge pull request #12803 from dmarcos/paren
9ed629301 Check parent of poseObject instead of camera
219f3eb13 Update GLTFLoader.js
15f13bb3c Update GLTFLoader.js
6d9c22a3b Update uniforms only when onWindowResize
881b25b58 Update ProjectionMatrix on change aspect
```

Jeder Commit verfügt über einen eindeutigen SHA-1-Hash zur Identifikation. Anhand dieser IDs bewegt man sich durch die Zeitachse durchgeföhrter Commits und sieht sie erneut an. Standardmäßig zeigt `git log` nur Commits für den aktuell ausgewählten Branch an. Es ist aber durchaus möglich, dass du nach einem Commit aus einem anderen Branch suchst. Du kannst alle Commits aller Branches ansehen, indem du `git log --branches=*` ausführst. Der Befehl `git branch` wird zur Anzeige und zum Aufrufen anderer Branches verwendet. Der Befehl `git branch -a` gibt eine Liste aller bekannten Branch-Namen zurück. Einer dieser Branch-Namen kann nun mit `git log <branch_name>` protokolliert werden.

Wenn du eine Commit-Referenz zu dem Zeitpunkt, den du besuchen möchtest, gefunden hast, kannst du mit dem Befehl `git checkout` diesen Commit aufrufen. `Git checkout` ist eine einfache Methode, diese gespeicherten Snapshots auf deinen Entwicklungsrechner zu "laden". Während des normalen Entwicklungsprozesses verweist `HEAD` üblicherweise auf den `master` oder einen anderen lokalen Branch, aber wenn du einen alten Commit auscheckst, verweist `HEAD` nicht mehr auf einen Branch – er verweist direkt auf einen Commit. Dieser Zustand wird als "detached HEAD" bezeichnet und kann folgendermaßen dargestellt werden:



Beim Aufrufen einer alten Datei wird der `HEAD`-Verweis nicht verschoben. Er verbleibt beim selben Branch und beim selben Commit, um einen losgelösten `HEAD` zu vermeiden. Du kannst die alte Version der Datei dann in Form eines neuen Snapshots erneut

committen, ganz wie jede andere Änderung auch. Wenn du `git checkout` auf diese Weise für eine Datei einsetzt, führst du also im Grunde genommen eine Zurücksetzung auf eine ältere Version der betreffenden Datei durch. Weitere Informationen zu diesen zwei Modi findest du auf der Seite [git checkout](#).

## Alte Überarbeitung anzeigen lassen

In diesem Beispiel gehen wir davon aus, dass du mit der Entwicklung eines verrückten Experiments begonnen hast, aber nicht weißt, ob du es aufbewahren willst. Um dir die Entscheidung zu erleichtern, willst du dir vor dem Experiment den Projektstatus ansehen. Zunächst musst du die ID der gesuchten Überarbeitung finden.

```
git log --oneline
```

Nehmen wir mal an, dein Projektverlauf sieht etwa so aus:

```
b7119f2 Weiter verrückte Ideen spinnen
872fa7e Etwas Verrücktes ausprobieren
a1e8fb5 Wichtige Änderungen an hello.txt machen
435b61d Erstellen von hello.txt
9773e52 Erster Import
```

Du kannst mit `git checkout` den Commit "Wichtige Änderungen an hello.txt machen" folgendermaßen anzeigen:

```
git checkout a1e8fb5
```

Dein Arbeitsverzeichnis befindet sich dann in exakt demselben Zustand wie der Commit `a1e8fb5`. Du kannst dir Dateien anschauen, das Projekt kompilieren, Tests durchführen und sogar Dateien bearbeiten, ohne befürchten zu müssen, den aktuellen Projektstatus zu verlieren. Keine deiner Aktionen und Änderungen wird im Repository gespeichert. Wenn du weiter an deinem Projekt arbeiten möchtest, musst du wieder in den "aktuellen" Projektzustand wechseln:

```
git checkout master
```

Dies setzt voraus, dass du im Standard-`master`-Branch entwickelst. Sobald du dich wieder im `master`-Branch befindest, kannst du entweder mit `git revert` oder `git reset` unerwünschte Änderungen rückgängig machen.

## Rückgängigmachen eines committeten Snapshots

Im Prinzip gibt es mehrere unterschiedliche Strategien, um einen

Commit rückgängig zu machen:

```
git log --oneline
872fa7e Etwas Verrücktes ausprobieren
a1e8fb5 Wichtige Änderungen an hello.txt machen
435b61d Erstellen von hello.txt
9773e52 Erster Import
```

Wir konzentrieren uns im Folgenden darauf, den Commit 872fa7e Etwas Verrücktes ausprobieren rückgängig zu machen. Die Idee war vielleicht doch ein wenig zu verrückt.

## So wird ein Commit mit "git checkout" rückgängig gemacht

Mit dem Befehl `git checkout` können wir den vorherigen Commit, a1e8fb5, auschecken und das Repository in den Status zurückversetzen, den es vor dem "verrückten" Commit hatte. Durch das Auschecken eines bestimmten Commits wird das Repository in einen Status mit losgelöstem HEAD versetzt. Das heißt, du arbeitest auf keinem Branch mehr. Im losgelösten Status verwaisen alle neu vorgenommenen Commits, sobald du zu einem eingerichteten Branch zurückkehrst. Verwaiste Commits werden bei der nächsten Speicherbereinigung von Git gelöscht. Die Speicherbereinigung erfolgt in konfigurierten Zeitabständen und entfernt verwaiste Commits endgültig. Damit verwaiste Commits nicht der Speicherbereinigung zum Opfer fallen, müssen wir sicherstellen, dass wir uns in einem Branch befinden.

Im Zustand mit losgelöstem HEAD können wir `git checkout -b new_branch_without_crazy_commit` ausführen. Hierdurch wird ein neuer Branch namens `new_branch_without_crazy_commit` erstellt und in diesen Zustand gewechselt. Das Repository befindet sich nun in einer neuen Verlaufszeitachse, in der der Commit 872fa7e nicht mehr existiert. Nun können wir in diesem neuen Branch, in dem der Commit 872fa7e nicht mehr existiert, arbeiten und den Commit als rückgängig gemacht betrachten. Leider ist diese Strategie nicht geeignet, wenn du den vorigen Branch benötigst, weil es vielleicht dein `master`-Branch war. Sehen wir uns deshalb ein paar andere Strategien zum Rückgängigmachen von Änderungen an. Weitere Informationen und Beispiele findest du in unserer detaillierten Behandlung von [git checkout](#).

## So wird ein Commit mit "git revert" rückgängig gemacht

Zurück zu unserem ursprünglichen Beispiel für einen Commit-Verlauf – den Verlauf mit dem Commit 872fa7e. Dieses Mal machen wir ihn rückgängig mit `git revert`. Wenn wir `git revert HEAD` ausführen, erstellt Git einen neuen Commit, der den letzten Commit umkehrt. Hierdurch wird dem aktuellen

Branch-Verlauf ein neuer Commit hinzugefügt, sodass er nun folgendermaßen aussieht:

```
git log --oneline
e2f9a78 Revert "Etwas Verrücktes ausprobieren"
872fa7e Etwas Verrücktes ausprobieren
a1e8fb5 Wichtige Änderungen an hello.txt machen
435b61d Erstellen von hello.txt
9773e52 Erster Import
```

Nun haben wir den Commit 872fa7e im Prinzip wieder rückgängig gemacht. Auch wenn 872fa7e im Verlauf noch vorhanden ist, sind im neuen Commit e2f9a78 die Änderungen von 872fa7e umgekehrt worden. Anders als bei unserer vorherigen Checkout-Strategie können wir weiterhin denselben Branch verwenden. Dies ist eine zufriedenstellende Lösung für das Rückgängigmachen von Änderungen. Es ist die ideale Methode bei der Arbeit mit öffentlichen gemeinsamen Repositorys. Wenn du einen gepflegten und minimalen Git-Verlauf benötigst, ist diese Strategie allerdings nicht zufriedenstellend.

## So wird ein Commit mit "git reset" rückgängig gemacht

Wir nutzen weiterhin unser Beispiel. `git reset` ist ein umfangreicher Befehl mit vielen Anwendungsmöglichkeiten und Funktionen. Wenn wir `git reset --hard a1e8fb5` aufrufen, wird der Commit-Verlauf auf diesen angegebenen Commit zurückgesetzt. Wenn wir mit `git log` einen Blick auf den Commit-Verlauf werfen, sieht dieser nun folgendermaßen aus:

```
git log --oneline
a1e8fb5 Wichtige Änderungen an hello.txt machen
435b61d Erstellen von hello.txt
9773e52 Erster Import
```

Die Protokollausgabe zeigt, dass die Commits e2f9a78 und 872fa7e im Verlauf nicht mehr vorhanden sind. Nun können wir weiterarbeiten und neue Commits erstellen, als ob es die "verrückten" Commits nie gegeben hätte. Diese Methode zum Rückgängigmachen von Änderungen hält den Verlauf am saubersten. Das Zurücksetzen eignet sich bestens für lokale Änderungen, kann aber in einem gemeinsamen Remote-Repository zu Komplikationen führen. Wenn wir ein gemeinsames Remote-Repository verwenden, zu dem der Commit 872fa7e gepusht wird, und wir für einen Branch `git push` verwenden, auf dem wir den Verlauf zurückgesetzt haben, bemerkt Git dies und gibt eine Fehlermeldung aus. Git wird annehmen, dass der gepushte Branch nicht aktuell ist, da Commits fehlen. In solchen Szenarien sollte `git revert` der Vorzug gegeben werden.

## Rückgängigmachen des letzten Commits

Im vorigen Abschnitt haben wir die verschiedenen Strategien zum Rückgängigmachen von Commits besprochen. Diese Strategien können alle auch auf den neuesten Commit angewendet werden. In einigen Fällen musst du jedoch den letzten Commit vielleicht gar nicht entfernen oder zurücksetzen. Vielleicht wurde dies etwas vorschnell gemacht. In diesem Fall kannst du den neuesten Commit berichtigen. Nachdem du weitere Änderungen im Arbeitsverzeichnis vorgenommen hast und diese für den Commit mit `git add` in die Staging-Umgebung überführt hast, kannst du `git commit --amend` ausführen. Hierdurch öffnet Git den konfigurierten System-Editor und du kannst die letzte Commit-Nachricht ändern. Die neuen Änderungen werden dem berichtigten Commit hinzugefügt.

## Rückgängigmachen nicht committeter Änderungen

Bevor Änderungen in den Repository-Verlauf committet werden, befinden sie sich im Staging-Index und im Arbeitsverzeichnis. Du musst die Änderungen wahrscheinlich in diesen beiden Bereichen rückgängig machen. Der Staging-Index und das Arbeitsverzeichnis sind interne Git-Mechanismen für das Zustandsmanagement. Detaillierte Informationen zur Funktionsweise dieser Mechanismen findest du auf der `git reset`-Seite.

## Das Arbeitsverzeichnis

Das Arbeitsverzeichnis ist in der Regel synchron zum lokalen Dateisystem. Zum Rückgängigmachen von Änderungen im Arbeitsverzeichnis kannst du Dateien ganz normal in deinem bevorzugten Editor bearbeiten. Git verfügt über ein paar Dienstprogramme, die beim Management des Arbeitsverzeichnisses behilflich sind. Da wäre der Befehl `git clean`, mit dem Änderungen am Arbeitsverzeichnis bequem rückgängig gemacht werden können. Außerdem kann noch `git reset` mit den Optionen `--mixed` oder `--hard` aufgerufen werden, um das Arbeitsverzeichnis zurückzusetzen.

## Der Staging-Index

Der Befehl `git add` wird zum Hinzufügen von Änderungen zum Staging-Index verwendet. `Git reset` wird vor allem zum Rückgängigmachen von Änderungen am Staging-Index verwendet. Ein `--mixed` Reset-Befehl verschiebt alle ausstehenden Änderungen vom Staging-Index zurück ins Arbeitsverzeichnis.

## Rückgängigmachen öffentlicher Änderungen

Bei der Arbeit in einem Team mit Remote-Repositories müssen beim Rückgängigmachen von Änderungen zusätzliche Aspekte bedacht werden. `git reset` sollte im Allgemeinen als eine lokale Methode zum Rückgängigmachen betrachtet werden. Ein Reset sollte durchgeführt werden, wenn Änderungen an einem privaten Branch rückgängig gemacht werden sollen. So wird die Entfernung von Commits auf sichere Weise von anderen Branches, die möglicherweise von anderen Entwicklern genutzt werden, isoliert. Probleme entstehen, wenn ein Reset in einem gemeinsamen Branch durchgeführt wird und dieser dann remote mit `git push` gepusht wird. Git wird den Push blockieren, da es den Branch als veraltet betrachtet, da ihm gegenüber dem Remote-Branch Commits fehlen.

Die bevorzugte Methode zum Zurücksetzen eines geteilten Verlaufs ist `git revert`. Eine Zurücksetzung mit "revert" ist sicherer als mit "reset", weil dabei keine Commits aus einem geteilten Verlauf entfernt werden. Beim Revert bleiben die Commits, die rückgängig gemacht werden sollen, erhalten. Es wird ein neuer Commit erstellt, der die Wirkung des unerwünschten Commits aufhebt. Diese Methode ist bei der Remote-Zusammenarbeit sicherer, weil ein Remote-Entwickler dann einen Pull für den Branch durchführen kann, um den neuen Commit zu erhalten, der den unerwünschten Commit rückgängig macht.

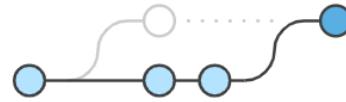
## Zusammenfassung

Wir haben nun einige allgemeine Strategien für das Rückgängigmachen von Änderungen in Git behandelt. Wichtig ist zu bedenken, dass es hierfür mehrere Methoden gibt. Auf dieser Seite wurden Themen angeschnitten, die auf den Seiten zu den jeweiligen Git-Befehlen tiefergehend erklärt werden. Die am häufigsten zum Rückgängigmachen genutzten Befehle sind `git checkout`, `git revert` und `git reset`. Folgende wichtige Punkte solltest du dir merken:

- Sobald für Änderungen ein Commit durchgeführt wurde, sind diese in der Regel permanent.
- Verwende `git checkout`, um den Commit-Verlauf durchzusehen.
- `git revert` ist am besten geeignet, um Änderungen in öffentlichen, gemeinsamen Branches rückgängig zu machen.
- `git reset` ist am besten geeignet, um Änderungen in lokalen, privaten Branches rückgängig zu machen.

Neben den wichtigsten Befehlen für das Rückgängigmachen von Änderungen, haben wir uns noch weitere Git-Dienstprogramme angesehen: `git log` zum Finden von verlorenen Commits, `git clean` zum Rückgängigmachen nicht committeter Änderungen, `git add` zum Ändern des Staging-Index.

Jeder dieser Befehle verfügt über eine eigene ausführliche Dokumentation. Wenn du mehr über einen bestimmten hier erwähnten Befehl zu erfahren möchtest, rufe die entsprechenden Links auf.



Weiter geht's mit:

**git clean**

[NÄCHSTES TUTORIAL BEGINNEN](#)

Entwickelt von

 Bitbucket

**Empfehl uns weiter**



**Interesse an künftigen Artikeln?**

Enter Your Email For Git News

**Website gehostet von**

 **ATLASSIAN**



Wenn nicht anders angegeben, sind alle Inhalte unter einer [Creative Commons-Lizenz 2.5 Australien](#) lizenziert.