

Git kennenlernen

Anfänger

Erste Schritte

Ein Repository einrichten

Änderungen speichern

Ein Repository

überprüfen

Änderungen rückgängig
machen

Verläufe umarbeiten

git commit --amend

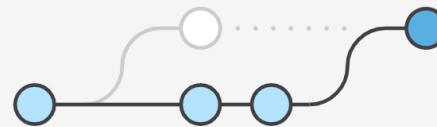
git rebase

git rebase -i

git reflog

Zusammenarbeit

Migration zu Git

Tipps für
Fortgeschrittene

Verläufe umarbeiten

git commit --amend und andere Methoden zum Umarbeiten von Verläufen

Einführung

In diesem Tutorial werden verschiedene Methoden zum Umarbeiten und Ändern von Git-Verläufen behandelt. In Git gibt es verschiedene Methoden zum Erfassen von Änderungen. Wir erläutern die Vor- und Nachteile der einzelnen Methoden und veranschaulichen die Arbeit mit ihnen anhand von Beispielen. In diesem Tutorial erläutern wir die häufigsten Gründe für das Überschreiben bestätigter Snapshots und zeigen dir, wie du diese Fallstricke vermeiden kannst.

Git sorgt vor allem dafür, dass deine bestätigte Änderung nie verloren geht. Mit Git hast du außerdem die volle Kontrolle über deinen Entwicklungs-Workflow. Unter anderem kannst du genau bestimmen, wie dein Projektverlauf aussehen soll. Das birgt jedoch auch das Risiko, dass Commits verloren gehen. Git bietet Befehle zum Umarbeiten von Verläufen unter dem Vorbehalt, dass durch Anwendung dieser Befehle Inhalte verloren gehen können.

Zum Speichern von Verläufen und Änderungen gibt es in Git mehrere Mechanismen. Zu diesen Mechanismen zählen: Die Commits -amend, git rebase und git reflog. Mit ihnen erhältst du leistungsstarke Optionen zur Anpassung des Workflows. Am Ende dieses Tutorials wirst du mit Befehlen vertraut sein, mit denen du deine Git-Commits neu strukturieren kannst, und dadurch in der Lage sein, häufige Fallstricke beim Umarbeiten von Verläufen zu vermeiden.

Letzten Commit ändern: git commit --amend

Der Befehl `git commit --amend` ist eine praktische Möglichkeit zum Ändern des aktuellsten Commits. So kannst du Änderungen auf Staging-Ebene mit dem vorherigen Commit kombinieren, statt einen ganz neuen Commit zu erstellen. Mit diesem Befehl kannst du auch einfach die vorherige Commit-Nachricht bearbeiten, ohne den entsprechenden Snapshot zu ändern. Durch das Korrigieren wird jedoch der aktuellste Commit nicht nur geändert, sondern komplett ersetzt. Das bedeutet, dass der korrigierte Commit eine neue Einheit mit eigener Referenz darstellt. Git behandelt ihn wie einen völlig neuen Commit, was an dem Sternsymbol (*) im unten stehenden Diagramm zu erkennen ist. Es gibt einige häufige Szenarien zur Verwendung von `git commit --amend`. In den folgenden Abschnitten stellen wir einige Beispiele vor.



Die aktuellste Commit-Nachricht in Git

ändern

```
git commit --amend
```

Nehmen wir mal an, du hast gerade einen Commit durchgeführt und dir ist in der Protokollnachricht des Commits ein Fehler unterlaufen. Führst du diesen Befehl aus, wenn noch nichts in die Staging-Ebene verschoben wurde, dann kannst du die Nachricht des vorherigen Commits bearbeiten, ohne den entsprechenden Snapshot zu verändern.

Im Laufe der täglichen Entwicklung werden immer wieder verfrühte Commits durchgeführt. Man vergisst gern einmal, eine Datei auf die Staging-Ebene zu verschieben, oder macht einen Fehler bei der Formatierung einer Commit-Nachricht. Mit der Kennzeichnung `--amend` lassen sich solche kleineren Fehler leicht beheben.

```
git commit --amend -m "an updated commit message"
```

Durch Hinzufügen der Option `-m` kannst du eine neue Nachricht über die Befehlszeile eingeben, ohne einen Editor öffnen zu müssen.

Bestätigte Dateien ändern

Das folgende Beispiel stellt ein häufig vorkommendes Szenario bei einer Git-basierten Entwicklung dar. Nehmen wir mal an, wir haben einige Dateien bearbeitet, für die wir einen Commit mit einem einzigen Snapshot durchführen wollen. Doch dann vergessen wir im ersten Anlauf, eine dieser Dateien hinzuzufügen. Um den Fehler zu beheben, müssen wir einfach die andere Datei in die Staging-Ebene verschieben und den Commit mit der Kennzeichnung `--amend` durchführen:

```
# Edit hello.py and main.py git add hello.py git commit
# Realize you forgot to add the changes from main.py
git commit --amend --no-edit
```

Mit der Kennzeichnung `--no-edit` kannst du Korrekturen an deinem Commit vornehmen, ohne die entsprechende Commit-Nachricht zu ändern. Der daraus entstehende Commit ersetzt den unvollständigen Commit. Es sieht dann so aus, als hätten wir für die Änderungen an `hello.py` und `main.py` einen Commit mit einem einzigen Snapshot durchgeführt.

Öffentliche Commits nicht korrigieren

Korrigierte Commits sind tatsächlich komplett neue Commits. Daher befindet sich der vorherige Commit nicht mehr in deinem aktuellen Branch. Das hat dieselben Folgen wie das Zurücksetzen eines öffentlichen Snapshots. Vermeide es daher, einen Commit zu korrigieren, auf den andere Entwickler ihre Arbeit aufbauen. Für Entwickler ist das eine verwirrende Situation und die Lösung ist kompliziert.

Zusammenfassung

Mit dem Befehl `git commit --amend` kannst du zum Reviewen den aktuellsten Commit abrufen und neue Änderungen aus der Staging-Ebene hinzufügen. Du kannst Änderungen aus der Staging-Umgebung von Git entfernen oder ihr hinzufügen, um einen Commit mit dem Befehl `--amend` darauf anzuwenden. Wurden noch keine Änderungen auf die Staging-Ebene verschoben, wirst du durch `--amend` weiterhin dazu aufgefordert, die letzte Commit-Protokollnachricht zu ändern. Sei daher vorsichtig, wenn du `--amend` auf Commits anwendest, die mit anderen Teammitgliedern geteilt werden. Einen Commit zu ändern, der mit einem anderen Benutzer geteilt wird, kann zu Merging-Konflikten führen und verwirrende und langwierige Lösungen erfordern.

Alte und mehrere Commits ändern

Mit `git rebase` kannst du ältere und mehrere Commits ändern und diese als eine Abfolge von Commits in einem neuen Basis-Commit zusammenführen. Im Standardmodus kannst du mit `git rebase` den Verlauf umarbeiten und Commits in deinem aktuellen Arbeits-Branch automatisch auf den genehmigten Branch-Head anwenden. Da deine neuen die alten Commits ersetzen, ist es wichtig, `git rebase` nicht auf Commits anzuwenden, die in den öffentlichen Bereich verschoben wurden.

Andernfalls scheint es so, als sei dein Projektverlauf verschwunden.

In diesen oder ähnlichen Fällen, in denen ein sauberer Projektverlauf von großer Bedeutung ist, kannst du die Option `-i` zu `git rebase` hinzufügen, um `rebase interactive` auszuführen. Dadurch bist du in der Lage, einzelne Commits im Prozess zu ändern und musst nicht alle Commits verschieben. Mehr zu interaktivem Rebasing und zu weiteren Rebasing-Befehlen erfährst du auf der Seite zu [Git-Rebasing](#).

Bestätigte Dateien ändern

Während des Rebasings wird durch die Änderung oder den Befehl `e` das Abspielen des Rebasings für diesen Commit angehalten, sodass du mit `git commit --amend` weitere Änderungen vornehmen kannst. Git unterbricht dann das Abspielen und zeigt eine Nachricht an:

```
Stopped at 5d025d1... formatting
You can amend the commit now, with
git commit --amend
Once you are satisfied with your changes, run
git rebase --continue
```

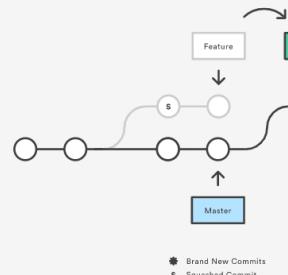
Mehrere Nachrichten

Zu jedem normalen Git-Commit gehört eine Protokollnachricht, die den Commit näher erklärt. Diese Nachrichten liefern einen wertvollen Einblick in den Projektverlauf. Während eines Rebasings kannst du einige Befehle auf Commits anwenden, um Commit-Nachrichten zu ändern.

- Durch Umformulieren oder "r" (von Englisch "reword") kannst du das Abspielen des Rebasings anhalten und die jeweilige Commit-Nachricht währenddessen umarbeiten.
- Alle Commits, die mit "s" (von "squash", Englisch für zusammendrücken) gekennzeichnet sind, werden angehalten und du wirst dazu aufgefordert, die einzelnen Commit-Nachrichten in einer gemeinsamen Nachricht zusammenzuführen. Mehr dazu erfährst du im Abschnitt zu Squash-Commits weiter unten.
- Das Fixup (Englisch für "anordnen") oder "f" führt Commits in ähnlicher Weise zusammen wie der Squash-Commit. Im Gegensatz zu Squash- unterbrechen Fixup-Commits jedoch nicht das Abspielen des Rebasing und es öffnet sich auch kein Editor, in dem Commit-Messages zusammengeführt werden sollen. Die Nachrichten der mit "f" gekennzeichneten Commits werden verworfen. Stattdessen wird die Nachricht des vorherigen Commits übernommen.

Squash-Commits zum Bereinigen des Verlaufs

Der "Squash"-Befehl s verdeutlicht den großen Nutzen des Rebasings. Mit dem Squash-Befehl kannst du festlegen, welche Commits du mit den vorherigen Commits mergen willst. So kann ein "sauberer Verlauf" entstehen. Während das Rebasing abgespielt wird, führt Git für jeden Commit den angegebenen Rebasing-Befehl aus. Bei Squash-Commits öffnet Git deinen konfigurierten Texteditor und fordert dich dazu auf, die angegebenen Commit-Nachrichten zusammenzuführen. Dieser gesamte Prozess kann wie folgt dargestellt werden:



Beachte, dass die Commits, die durch einen Rebasing-Befehl verändert wurden, eine andere ID als die ursprünglichen Commits haben. Die mit "pick" gekennzeichneten Commits haben eine neue ID, wenn die vorherigen Commits umgearbeitet wurden.

Moderne Hosting-Lösungen für Git, wie etwa Bitbucket, bieten jetzt Features für "automatisiertes Squashing" nach dem Mergen. Diese Features führen das Rebasing und Squashing von Branch-Commits automatisch über die Benutzeroberfläche der Hosting-Solution für dich durch. Mehr Informationen findest du unter "[Squash commits when merging a Git branch with Bitbucket](#)" ([Squash-Commits beim Mergen von Git-Branches mit Bitbucket](#)).

Zusammenfassung

Das Git-Rebasing gibt dir die Möglichkeit, deinen Verlauf zu

bearbeiten. Mit dem interaktiven Rebasing hinterlässt du dabei keine "unschönen" Spuren. So kannst du ohne Probleme Fehler machen und korrigieren, deine Arbeit verbessern und dabei einen sauberen, linearen Projektverlauf aufrechterhalten.

Das Sicherheitsnetz: git reflog

Referenzprotokolle oder "Reflogs" sind Mechanismen, die Git zum Erfassen von Aktualisierungen nutzt, die auf Branch-Spitzen oder andere Commit-Referenzen angewendet werden. Mit Reflogs kannst du zu Commits zurückgehen, auch wenn keine Referenz zu einem Branch oder Tag besteht. Nachdem der Verlauf umgeschrieben wurde, enthält das Reflog Informationen zum alten Zustand der Branches und ermöglicht es dir, wenn nötig zu diesem Zustand zurückzukehren. Jedes Mal, wenn deine Branch-Spitzen aus irgendeinem Grund aktualisiert werden (beim Wechseln von Branches, beim Einbringen neuer Änderungen, beim Umarbeiten des Verlaufs oder einfach durch Hinzufügen neuer Commits), wird deinem Reflog ein neuer Eintrag hinzugefügt. In diesem Abschnitt werfen wir einen allgemeinen Blick auf den Befehl `git reflog` und stellen häufige Nutzungsmöglichkeiten vor.

Anwendung

```
git reflog
```

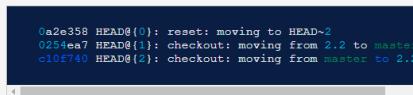
Damit wird das Reflog für das lokale Repository angezeigt.

```
git reflog --relative-date
```

So wird das Reflog mit relativen Zeitangaben (z. B. vor zwei Wochen) angezeigt.

Beispiel

Spielen wir ein Beispiel durch, um uns mit der Funktionsweise von `git reflog` vertraut zu machen.



```
0a2e358 HEAD@{0}: reset: moving to HEAD~2
0254ea7 HEAD@{1}: checkout: moving from 2.2 to master
c10f740 HEAD@{2}: checkout: moving from master to 2.2
```

Die "reflog"-Ausgabe oben zeigt einen Checkout vom Branch "master" zum Branch "2.2" und einen Checkout in umgekehrter Richtung. Anschließend wurde eine harte Zurücksetzung auf einen älteren Commit durchgeführt. Die zuletzt durchgeführte Aktivität ist zuerst genannt und mit `HEAD@{0}` bezeichnet.

Wenn du versehentlich zurückgegangen bist, enthält das Reflog den Verweis des Commit-Master-Branches auf `(0254ea7)`, so wie er war, bevor die beiden Commits aus Versehen verloren gegangen sind.

```
git reset --hard 0254ea7
```

Mit `git reset` kannst du jetzt den Master-Branch auf den vorherigen Commit zurücksetzen. Damit hast du ein Sicherheitsnetz, falls der Verlauf aus Versehen geändert wurde.

Du solltest unbedingt beachten, dass das Reflog nur dann als Sicherheitsnetz dienen kann, wenn Änderungen an dein lokales Repository bestätigt worden sind, und dass nur Verschiebungen bei Branch-Spitzen des Repositorys verfolgt werden können. Zusätzliche Reflog-Einträge erlöschen nach einer Frist. Diese Frist beträgt für Reflog-Einträge standardmäßig 90 Tage.

Weitere Informationen findest du auf unserer Seite zu [git reflog](#).

Zusammenfassung

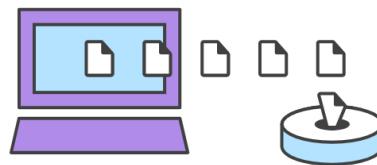
In diesem Beitrag haben wir verschiedene Methoden zum Ändern des Git-Verlaufs und zum Rückgängigmachen von Git-Änderungen besprochen. Wir haben einen allgemeinen Blick auf den Rebasing-Prozess in Git geworfen. Das sind einige der wichtigsten Punkte:

- Es gibt viele Möglichkeiten, Verläufe mit Git umzuarbeiten.

- Mit `git commit --amend` kannst du die neueste Protokollnachricht ändern.
- Mit `git commit --amend` kannst du außerdem den aktuellsten Commit bearbeiten.
- Mit `git rebase` kannst du Commits zusammenführen und den Branch-Verlauf bearbeiten.
- `git rebase -i` ermöglicht dir im Vergleich zu einem standardmäßigen Git-Rebasing eine genauere Kontrolle über die Verlaufsänderungen.

Mehr über die vorgestellten Befehle erfährst du auf den jeweiligen Seiten:

- [git rebase](#)
- [git reflog](#)



Weiter geht's mit:

git rebase

[NÄCHSTES TUTORIAL BEGINNEN](#)

Entwickelt von



[Empfehl uns weiter](#)



[Interesse an künftigen Artikeln?](#)

Enter Your Email For Git News

[Website gehostet von](#)



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