Git Merge

From: https://www.atlassian.com/de/git/tutorials/using-branches/git-merge

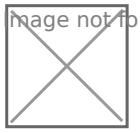
In Git kannst du einen geforkten Verlauf per Merge wieder zusammensetzen. Mit dem Befehl `git merge` kannst du die unabhängigen Entwicklungszweige, die mit `git branch` erstellt wurden, in einen einzigen Branch integrieren.

Beachte, dass alle folgenden Befehle in den aktuellen Branch gemergt werden. Der aktuelle Branch wird aktualisiert, um den Merge wiederzugeben, doch der Ziel-Branch bleibt davon komplett unberührt. Zur Erinnerung: Der Befehl `git merge` wird also oft im Zusammenhang mit `git checkout` zum Auswählen des aktuellen Branchs und `git branch -d` zum Löschen des veralteten Ziel-Branchs genutzt.

Wie es funktioniert

Mit `git merge` werden mehrere Commit-Abfolgen in einen einheitlichen Verlauf zusammengeführt. Vor allem wird `git merge` genutzt, um zwei Branches zu vereinen. Auf dieses Branch-Merging-Muster konzentrieren sich die folgenden Beispiele. In diesen Fällen sucht `git merge` zwischen zwei Commit-Pointern, was für gewöhnlich die Branch-Spitzen sind, einen gemeinsamen Basis-Commit. Sobald Git den gemeinsamen Basis-Commit gefunden hat, wird ein neuer "Merge-Commit" erstellt, um die Änderungen der einzelnen Abfolgen von Merge-Commits in der Warteschlange zusammenzuführen.

Nehmen wir an, wir haben ein neues Branch-Feature, das auf Basis des `master`-Branchs erstellt wurde. Wir möchten diesen Feature Branch nun in den `master` mergen.



Mit diesem Befehl wird das angegebene Branch-Feature in den aktuellen Branch gemergt – in diesem Fall in den `master`. Git bestimmt den Merge-Algorithmus automatisch (später mehr dazu).

Merge-Commits haben zwei übergeordnete Commits und unterscheiden sich damit von anderen Commits. Git versucht beim Erstellen eines Merge-Commits ganz automatisch, die separaten Verläufe zu mergen. Daten, die in beiden Verläufen geändert wurden, können von Git nicht

automatisch zusammengeführt werden. Dieses Szenario führt in Git zu einem Versionskontrollkonflikt und erfordert den Eingriff des Benutzers.

Vorbereitung auf das Mergen

Damit das Mergen problemlos funktioniert, solltest du vorbereitend einige Schritte durchführen.

Den Merge-Ziel-Branch bestätigen

Führe `git status` aus, um sicherzustellen, dass der `HEAD` auch auf den Branch verweist, der wirklich das Ziel des Merge-Prozesses ist. Falls erforderlich, kannst du mit `git checkout` zu dem Branch wechseln, der das Merge-Ziel sein soll. In diesem Fall führen wir `git checkout master` aus.

Neueste Remote-Commits abrufen

Stelle sicher, dass der Merge-Ziel-Branch und der Merging-Branch mit den neuesten Remote-Änderungen aktualisiert wurden. Mit `git fetch` kannst du die neuesten Remote-Commits pullen. Vergewissere dich nach dem Abrufen der Commits, dass der `master`-Branch über die neuesten Updates verfügt, indem du `git pull` ausführst.

Verschmelzung

Wenn die zuvor beschriebenen Schritte zur Vorbereitung auf das Mergen abgeschlossen sind, kann der Merge-Vorgang beginnen. Führe dazu `git merge` aus, wobei der Name des Branchs ist, der in den Merge-Ziel-Branch gemergt wird.

Fast-Forward-Merge

Ein Fast-Forward-Merge findet statt, wenn ein linearer Pfad von der Spitze des aktuellen Branchs zum Ziel-Branch existiert. Statt die Branches tatsächlich zusammenzuführen, muss Git zur Integration der verschiedenen Verläufe lediglich die Spitze des aktuellen Branchs an die Spitze des Ziel-Branchs verschieben ("fast-forward"). Dadurch werden die Verläufe miteinander kombiniert, und alle Commits aus dem Ziel-Branch sind auch über den aktuellen Branch zugänglich. Ein Fast-Forward-Merge eines Features in den `master` könnte etwa so aussehen:

Image not found or type unknown

Ein Fast-Forward-Merge ist jedoch nicht möglich, wenn die Branches voneinander abweichen. Wenn kein linearer Pfad zum Ziel-Branch existiert, kann Git die Branches nur mithilfe eines 3-Way-Merges zusammenführen. Für 3-Way-Merges gibt es einen speziellen Commit, um die beiden Verläufe miteinander zu verknüpfen. Die Benennung geht darauf zurück, dass Git drei Commits zum Durchführen des Merge-Commits verwendet: für die beiden Branch-Spitzen und ihren gemeinsamen Vorgänger.

Image not found or type unknown

Beide Merge-Strategien stehen dir offen. Viele Entwickler bevorzugen jedoch Fast-Forward-Merges (mit Unterstützung von Rebasing) für kleine Features oder Bugfixes und greifen nur zur Integration langlebigerer Features auf 3-Way-Merges zurück. Bei der letzten Strategie dient der entstehende Merge-Commit zum symbolischen Zusammenführen der beiden Branches.

Unser erstes Beispiel zeigt einen Fast-Forward-Merge. Mit dem Code unten werden ein neuer Branch erstellt, zwei Commits zu diesem Branch hinzugefügt und der Branch dann mit einem Fast-Forward-Merge in den Hauptzweig integriert.

```
# Start a new feature
git checkout -b new-feature master
# Edit some files
git add <file>
git commit -m "Start a feature"
# Edit some files
git add <file>
git commit -m "Finish a feature"
# Merge in the new-feature branch
git checkout master
git merge new-feature
git branch -d new-feature
```

Dieser Workflow ist für kurzlebige Themen-Branches üblich, die weniger als Organisationstool für langlebigere Features als vielmehr zur isolierten Entwicklung genutzt werden.

Außerdem sollte Git bei `git branch -d` kein Problem melden, da der Zugriff auf das neue Feature (new-feature) nun vom Master-Branch aus möglich ist.

Falls du während eines Fast-Forward-Merge-Vorgangs zu Dokumentationszwecken einen Merge-Commit durchführen musst, führe `git merge` mit der Option `--no-ff` aus.

```
git merge --no-ff <branch>
```

Mit diesem Befehl wird der angegebene Branch in den aktuellen Branch gemergt. Dabei wird immer ein Merge-Commit erzeugt (auch bei Fast-Forward-Merges). Das ist nützlich, wenn du alle Merges in deinem Repository dokumentieren willst.

3-Way-Merge

Das nächste Beispiel ist sehr ähnlich. Hier ist jedoch ein 3-Way-Merge erforderlich, weil der `master`-Branch fortgeführt und gleichzeitig das Feature bearbeitet wird. Dieses Szenario kommt häufig vor, wenn es um umfangreiche Features geht oder mehrere Entwickler zeitgleich an einem Projekt arbeiten.

```
Start a new feature
git checkout -b new-feature master
# Edit some files
git add <file>
git commit -m "Start a feature"
# Edit some files
git add <file>
git commit -m "Finish a feature"
# Develop the master branch
git checkout master
# Edit some files
git add <file>
git commit -m "Make some super-stable changes to master"
# Merge in the new-feature branch
git merge new-feature
git branch -d new-feature
```

Beachte, dass es für Git unmöglich ist, einen Fast-Forward-Merge durchzuführen, da der `master` ohne Zurückverfolgung nicht zu `new-feature` verschoben werden kann.

In den meisten Workflows wäre `new-feature` ein viel größeres Feature mit einer langen Entwicklungszeit. Daher würden neue Commits in der Zwischenzeit im `master` erscheinen. Wenn dein Feature Branch aber denselben Umfang hat wie im Beispiel oben, würdest du ihn

wahrscheinlich besser auf den `master` rebasen und einen Fast-Forward-Merge durchführen. So vermeidest du, dass unnötige Merge-Commits den Projektverlauf aufblähen.

Konflikte lösen

Wenn du versuchst, zwei Branches zu mergen, und es in beiden Branches Änderungen in demselben Teil derselben Datei gibt, dann weiß Git nicht, welche Version übernommen werden soll. In einer solchen Situation wird der Vorgang vor dem Merge-Commit angehalten und du kannst den Konflikt manuell lösen.

Das Tolle am Merge-Prozess in Git: Mit dem gewohnten Workflow zum Bearbeiten, Stagen und Durchführen von Commits kannst du auch Merge-Konflikte lösen. Führe bei einem Merge-Konflikt einfach den Befehl `git status` aus, um die betroffenen Dateien anzuzeigen. Wenn zum Beispiel in beiden Branches derselbe Abschnitt von `hello.py` geändert wurde, wird dir in etwa Folgendes angezeigt:

```
On branch master
Unmerged paths:
(use "git add/rm ..." as appropriate to mark resolution)
both modified: hello.py
```

Darstellung von Konflikten

Wenn Git während eines Merge-Vorgangs einen Konflikt feststellt, werden in den betroffenen Dateien Anfang und Ende der Inhalte, die in Konflikt stehen, optisch gekennzeichnet. Die optische Kennzeichnung sieht wie folgt aus: <<<<<<<, =======, and >>>>>>>. Es ist hilfreich, während eines Merges nach dieser Kennzeichnung Ausschau zu halten, um mögliche Konflikte zu erkennen.

```
here is some content not affected by the conflict
<<<<<<< master
this is conflicted text from master
=======
this is conflicted text from feature branch
>>>>>>> feature branch;
```

Im Allgemeinen handelt es sich bei dem Inhalt vor der Kennzeichnung `=======` um den Merge-Ziel-Branch, während der Teil danach der Merging-Branch ist.

Sobald du ermittelt hast, welche Abschnitte in Konflikt stehen, kannst du den Merge nach deinen Vorstellungen korrigieren. Wenn du bereit bist, den Merge abzuschließen, musst du lediglich den Befehl `git add` auf die in Konflikt stehende(n) Datei(en) anwenden, um Git mitzuteilen, dass sie gelöst sind. Anschließend führst du einen normalen `git commit` aus, um den Merge-Commit zu erzeugen. Der Prozess ist derselbe wie beim Committen eines gewöhnlichen Snapshots. Normale Entwickler können eigene Merges also ganz einfach verwalten.

Beachte, dass es nur bei 3-Way-Merges zu Merge-Konflikten kommt. In Fast-Forward-Merges können keine Änderungskonflikte entstehen.

Zusammenfassung

Wir haben hier einen Überblick über den Befehl `git merge` gegeben. Das Mergen ist bei der Arbeit mit Git ein wesentlicher Prozess. Dabei haben wir die internen Mechanismen hinter dem Mergen und die Unterschiede zwischen einem Fast-Forward-Merge und einem echten, sogenannten 3-Way-Merge behandelt. Dies sind einige der wichtigsten Punkte:

1. Durch das Mergen in Git werden mehrere Commit-Abfolgen zu einem einheitlichen Commit-Verlauf zusammengeführt.
2. Es gibt allgemein gesehen zwei Merging-Möglichkeiten in Git: Fast-Forward- und 3-Way-Merging.
3. Git kann Commits automatisch mergen, vorausgesetzt, es gibt keine Änderungen, die zu Konflikten mit beiden Commit-Abfolgen führen.

In diesem Tutorial wurden unter anderem auch folgende Git-Befehle genutzt oder auf sie verwiesen: `git branch`, `git pull` und `git fetch`. Mehr über diese Befehle erfährst du auf den jeweiligen Seiten zu diesen Themen.

Revision #2
Created 12 March 2021 10:58:08 by Alois
Updated 12 March 2021 10:59:59 by Alois