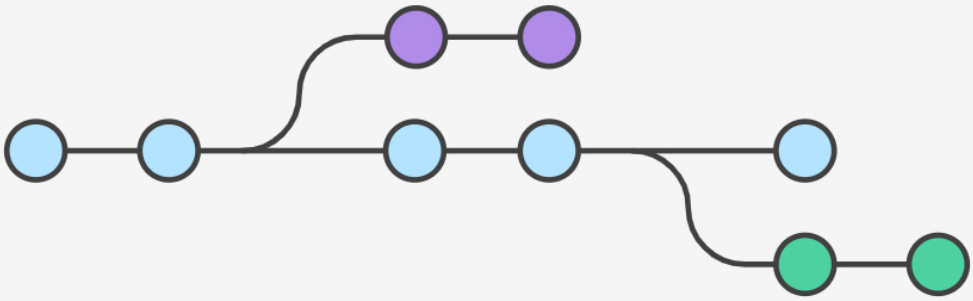
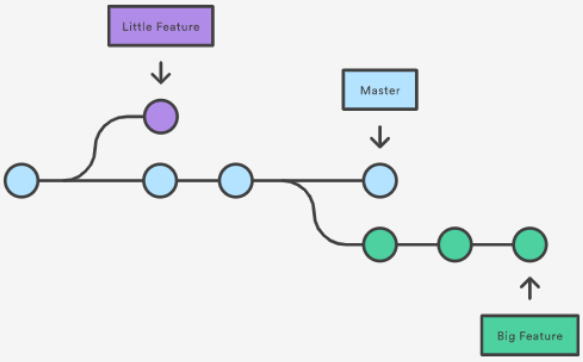
Git kennenlernen
Anfänger
Erste Schritte
Zusammenarbeit
Synchronisierung
Einen Pull Request erstellen
Arbeiten mit Branches
git branch
git checkout
git merge
Konflikte mergen
Merge-Strategien
Workflows vergleichen
Migration zu Git
Tipps für Fortgeschrittene



# git branch

In diesem Artikel behandeln wir den Befehl `git branch` im Detail und geben dir einen Überblick über das Branching-Modell von Git. Branching ist ein Feature, das in den meisten modernen Versionskontrollsystemen verfügbar ist. Es kann in anderen Versionskontrollsystemen allerdings bezüglich Zeit und Festplattenspeicher ein kostspieliger Vorgang sein. In Git sind Branches Bestandteil deines alltäglichen Entwicklungsprozesses. Git-Branches sind quasi Verweise auf einen Snapshot deiner Änderungen. Wenn du ein neues Feature hinzufügen oder einen Fehler beheben möchtest, legst du einen neuen Branch an, der deine (großen oder kleinen) Änderungen enthält. Auf diese Weise ist es weniger wahrscheinlich, dass unstabiler Code in die Haupt-Codebasis gemergt wird, und du hast die Möglichkeit, deinen zukünftigen Verlauf zu bereinigen, bevor du den Merge in den Haupt-Branch durchführst.




Das Schaubild oben zeigt ein Repository mit zwei isolierten Entwicklungslinien: einer Linie für ein kleines Feature und einer Linie für ein langlebigeres Feature. Die Auslagerung der Entwicklung in Branches macht es möglich, an beiden Features parallel zu arbeiten. Außerdem wird verhindert, dass fragwürdiger Code in den Haupt-Branch `master` überführt wird.

Die Git-Implementierung des Branch-Konzepts ist sehr viel schlanker als andere Versionskontrollsystem-Modelle. Statt Dateien von Verzeichnis zu Verzeichnis zu kopieren, speichert Git einen Branch als Verweis auf einen Commit. Ein Branch

repräsentiert also gleichsam die Spitze einer Reihe aufeinanderfolgender Commits. Er fungiert nicht als Commit-Container. Der Verlauf eines Branches wird aus den Commit-Beziehungen abgeleitet.

Beachte dabei, dass es zwischen Git-Branches und SVN-Branches Unterschiede gibt. Während SVN-Branches nur für gelegentliche größere Entwicklungsarbeiten genutzt werden, sind Git-Branches ein wesentlicher Bestandteil des täglichen Workflows. Im Folgenden wird die interne Branching-Architektur von Git behandelt.

# Wie es funktioniert

Branches sind unabhängige Entwicklungslinien. Sie dienen als Abstrahierung des Prozesses "Bearbeitung/Staging/Commit". Du kannst dir Branches als eine Möglichkeit vorstellen, ein vollständig neues Arbeitsverzeichnis inklusive neuer Staging-Umgebung und neuem Projektverlauf einzurichten. Neue Commits werden im Verlauf des aktuellen Branch aufgezeichnet. Das wird im Projektverlauf durch eine Fork abgebildet.

Mit dem Befehl `git branch` kannst du Branches erstellen, abrufen, umbenennen und löschen. Ein Wechsel zwischen Branches oder die Zusammenführung eines abgespaltenen Verlaufs ist mit ihm allerdings nicht möglich. Daher ist `git branch` eng mit den Befehlen `git checkout` und `git merge` verzahnt.

# Allgemeine Optionen

```
git branch
```

Führt alle Branches in deinem Repository auf. Dies ist synonym zu `git branch --list`.

```
git branch <branch>
```

Mit diesem Befehl erstellst du einen neuen Branch mit dem Namen, den du für `<branch>` angibst. Der neue Branch wird jedoch *nicht* ausgecheckt.

```
git branch -d <branch>
```

Löscht den angegebenen Branch. Das ist insofern ein "sicherer" Vorgang, als Git dafür sorgt, dass du keinen Branch löschst, wenn es nicht gemergte Änderungen gibt.

```
git branch -D <branch>
```

Erzwingt das Löschen des angegebenen Branches, selbst wenn es nicht gemergte Änderungen gibt. So sieht der Befehl aus, mit

dem du alle Commits, die mit einem bestimmten Entwicklungszweig in Verbindung stehen, dauerhaft verwerfen kannst.

```
git branch -m <branch>
```

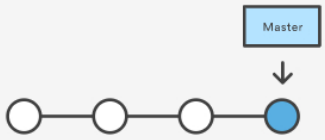
Mit diesem Befehl benennst du den aktuellen Branch in den Wert um, den du für `<branch>` angibst.

```
git branch -a
```

Führt alle Remote-Branches auf.

# Branches erstellen

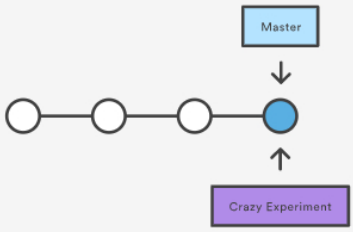
Führe dir dabei immer vor Augen: Branches sind lediglich Pointer, die auf Commits verweisen. Wenn du einen Branch erstellst, muss Git nur einen neuen Pointer erstellen. Es werden keine weiteren Änderungen am Repository vorgenommen. Angenommen, dein Repository sieht wie folgt aus:




Dann erstellst du mit folgendem Befehl einen Branch:

```
Git-Branch crazy-experiment
```

Der Repository-Verlauf bleibt unverändert. Du erhältst lediglich einen neuen Verweis zum aktuellen Commit:




Wichtig: Dieser Befehl *erstellt* den neuen Branch lediglich. Damit du ihm Commits hinzufügen kannst, musst du ihn erst mit `git checkout` auswählen. Anschließend kannst du die Standardbefehle `git add` und `git commit` verwenden.

# Remote-Branches erstellen

Die bisherigen Beispiele haben alle lokale Branch-Vorgänge dargestellt. Der Befehl `git branch` lässt sich aber auch für Remote-Branches anwenden. Hierfür muss zuerst ein Remote-Repository konfiguriert und der lokalen Repository-Konfiguration

hinzugefügt werden.

```
$ git remote add new-remote-repo https://bitbucket.com
# Add remote repo to local repo config
$ git push <new-remote-repo> crazy-experiment~
# pushes the crazy-experiment branch to new-remote-rep
```

Dieser Befehl pusht eine Kopie des lokalen Branches `crazy-experiment` in das Remote-Repository `<remote>`.

# Branches löschen

Sobald du deine Arbeit an einem Branch abgeschlossen und ihn in die Haupt-Codebasis gemergt hast, kannst du den Branch einfach löschen, ohne dass Verlaufsinformationen verloren gehen:

```
git branch -d crazy-experiment
```

Wenn der Branch jedoch nicht gemergt wurde, gibt der obige Befehl eine Fehlermeldung aus:

```
error: The branch 'crazy-experiment' is not fully merg
If you are sure you want to delete it, run 'git branch
```

Dieser Fehler verhindert, dass du den Zugriff auf diese gesamte Entwicklungslinie verlierst. Ist dein Experiment fehlgeschlagen und du möchtest den Branch wirklich löschen, kannst du das großgeschriebene Flag `-D` verwenden:

```
git branch -D crazy-experiment
```

Dadurch wird der Branch unabhängig von seinem Status und ohne Warnung gelöscht. Verwende den Befehl also mit Bedacht.

Mit den vorherigen Befehlen wird eine lokale Kopie eines Branches gelöscht. Der Branch ist jedoch möglicherweise noch in einem Remote-Repository vorhanden. Ein Remote-Branch wird folgendermaßen gelöscht:

```
git push origin --delete crazy-experiment
```

oder

```
git push origin :crazy-experiment
```

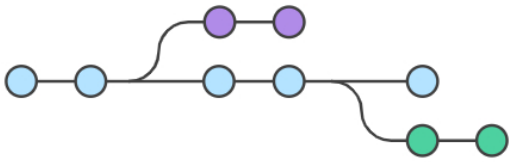
Hiermit wird ein Löschsignal an das Remote-Ursprungs-Repository gesendet und der Remote-Branch `crazy-experiment` wird daraufhin gelöscht.

# Zusammenfassung

In diesem Artikel haben wir das Branching-Verhalten von Git und den Befehl `git branch` behandelt. Die Hauptfunktionen des Befehls `git branch` sind die Erstellung, Auflistung, Umbenennung und Löschung von Branches. Für weitere Vorgänge in den daraus resultierenden Branches wird der Befehl häufig in Kombination mit anderen Befehlen wie `git checkout` verwendet. Weitere Informationen zu den Branch-Vorgängen mit `git checkout`, wie z. B. das Wechseln und Mergen von Branches, findest du auf der `git checkout`-Seite.

Im Vergleich zu anderen Versionskontrollsystemen sind die Branch-Vorgänge in Git kostengünstig und werden häufig eingesetzt. Ihre Flexibilität ermöglicht eine leistungsstarke individuelle Anpassung von Git-Workflows. Weitere Informationen zu den Git-Workflows findest du auf unseren erweiterten Workflow-Seiten zum Feature-Branch-Workflow, Git-flow-Workflow und dem Forking-Workflow.





Weiter geht's mit:

## git checkout

NÄCHSTES TUTORIAL BEGINNEN