



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Kapitel 02

REMOTE-TRACKING BRANCHES

Das Problem: Geister-Banches

Nach dem Mergen von PRs siehst Sie möglicherweise Branches, die nicht mehr existieren:

```
$ git branch -a  
main  
remotes/origin/main  
remotes/origin/feature/old-pr ← Wurde auf GitHub gelöscht!
```

Warum passiert das?

Git räumt Referenzen zu gelöschten Remote-Banches nicht automatisch auf.

Die drei Arten von Branches

Wenn Sie `git branch -a` ausführst, siehst Sie **drei verschiedene Dinge**:

```
$ git branch -a
main                ← Lokaler Branch
feature/new-feature ← Lokaler Branch
remotes/origin/main  ← Remote-Tracking Branch
remotes/origin/feature/xyz ← Remote-Tracking Branch
```

Lass uns jede Art verstehen...

Typ 1: Lokale Branches

```
main  
feature/new-feature
```

Lokale Branches:

- Ihre eigenen Branches, an denen Sie arbeiten
- Existieren nur auf Ihrem Computer
- Sie erstellt sie mit `git checkout -b`
- Sie kontrollierst, wann sie erstellt und gelöscht werden

Typ 2: Remote-Tracking Branches

```
remotes/origin/main  
remotes/origin/feature/xyz
```

Remote-Tracking Branches:

- **Lokale Kopien** dessen, was auf GitHub existiert
- Werden automatisch aktualisiert, wenn Sie `git fetch` oder `git pull` ausführst
- Sie arbeiten nicht direkt an diesen – sie sind Lesezeichen, die den Zustand von GitHub zeigen
- Benannt mit `remotes/origin/` Präfix

Typ 3: Das Remote-Repository (GitHub)

Die tatsächlichen Branches auf GitHub:

- Die zentrale Wahrheit
- Was Ihr Team sieht
- Wo PRs gemerged werden
- Wo Branches nach dem Mergen gelöscht werden

Die Lesezeichen-Analogie

Denken Sie an Remote-Tracking Branches wie **Lesezeichen in Ihrem Browser**:

Git-Konzept	Browser-Analogie
GitHub	Die tatsächlichen Webseiten
Remote-Tracking Branches	Ihre Lesezeichen, die auf GitHubs Branches zeigen
Lokale Branches	Ihre eigenen Notizen/Entwürfe

Das Problem: Wenn ein Branch auf GitHub gelöscht wird (nach dem Mergen eines PR), zeigt Ihr Lesezeichen immer noch darauf!

Warum veraltete Referenzen entstehen

Typischer Workflow:

1. ✓ Sie erstellt einen Feature-Branch und pushst ihn zu GitHub
2. ✓ Sie erstellt einen PR und er wird gemerged
3. ✓ GitHub **löscht automatisch** den Branch nach dem Mergen Sie
4. ✗ Ihr lokales Git **hat immer noch** `remotes/origin/feature/xyz` (veraltet!)

Git entfernt diese Referenzen nicht automatisch, weil:

- Es nicht weiß, ob die Löschung beabsichtigt war
- Sie offline gewesen sein könntest, als die Löschung stattfand
- Git auf der sicheren Seite bleibt und Informationen behält

Beispiel: Veraltete Referenzen in Aktion

Nachdem PR gemerged und Branch auf GitHub gelöscht wurde:

```
$ git branch -a  
main  
remotes/origin/main  
remotes/origin/feature/old-pr ← Veraltet! Existiert nicht mehr auf GitHub
```

Das verursacht Verwirrung:

- `git branch -a` zeigt Branches, die nicht existieren
- Sie könnten versehentlich versuchen, an gelöschten Branches zu arbeiten
- Repository sieht unübersichtlich aus

Lösung 1: `git fetch --prune`

Die `--prune` Flag sagt Git: "Entferne lokale Referenzen zu Remote-Banches, die nicht mehr existieren."

```
$ git fetch --prune
```

Was das macht:

1. **Fetch**: Neue Commits und Branches von GitHub herunterladen
2. **Prune**: Remote-Tracking Branches entfernen, die nicht mehr auf GitHub existieren

`git fetch --prune` Beispiel

Vor dem Pruning:

```
$ git branch -a  
main  
remotes/origin/main  
remotes/origin/feature/old-pr  
remotes/origin/feature/another-merged
```

Pruning ausführen:

```
$ git fetch --prune  
From https://github.com/user/repo  
- [deleted]          (none)      -> origin/feature/old-pr  
- [deleted]          (none)      -> origin/feature/another-merged
```

Übersetzung: "Diese Remote-Tracking Branches wurden lokal gelöscht, weil sie nicht mehr auf GitHub existieren."

Lösung 2: `git remote prune origin`

Alternativer Befehl:

```
$ git remote prune origin
```

Unterschied zu `git fetch --prune`:

- `git fetch --prune`: Lädt neue Daten herunter **UND** entfernt veraltete Referenzen (empfohlen)
- `git remote prune origin`: Entfernt **nur** veraltete Referenzen (lädt keine neuen Daten herunter)

Vorschau vor dem Pruning

Sehen Sie, was entfernt werden würde, ohne es tatsächlich zu entfernen:

```
$ git remote prune origin --dry-run
Pruning origin
URL: https://github.com/user/repo
* [would prune] origin/feature/old-pr
* [would prune] origin/feature/another-merged
```

Verwenden Sie dies, wenn Sie nicht sicher bist, was gelöscht wird!

Vollständiger Branch-Aufräum-Workflow

Nach dem Mergen eines PR und Löschen des Branches auf GitHub:

```
# Schritt 1: Zu main wechseln und aktualisieren (mit automatischem Pruning)
git checkout main
git pull --prune

# Schritt 2: Ihren lokalen Feature-Branch löschen
git branch -d feature/my-merged-feature
```

`git pull --prune` verstehen

`git pull --prune` macht DREI Dinge auf einmal:

1. **Fetcht** Updates von GitHub (aktualisiert `origin/main` Lesezeichen)
2. **Mergt** diese Updates in Ihren lokalen main (bewegt main Pointer vorwärts)
3. **Entfernt** veraltete Remote-Tracking Branches (wie `remotes/origin/feature/my-merged-feature`)

Es ist der effizienteste Aufräum-Befehl!

Visuell: Vor dem Aufräumen

Ihr lokales Git:

└─ Lokale Branches:

- └─ main
- └─ feature/old-pr (gemerged, aber Sie haben vergessen zu löschen)

└─ Remote-Tracking Branches:

- └─ remotes/origin/main ✓ (existiert auf GitHub)
- └─ remotes/origin/feature/old-pr ✗ (gelöscht auf GitHub)
- └─ remotes/origin/feature/another-old ✗ (gelöscht auf GitHub)

Visuell: Nach dem Aufräumen

```
Ihr lokales Git:  
├─ Lokale Branches:  
|   └─ main (sauber!)  
└─ Remote-Tracking Branches:  
    └─ remotes/origin/main ✓ (existiert auf GitHub)
```

Viel sauberer!

Automatisches Pruning (Empfohlene Einrichtung)

Konfigurieren Sie Git, um beim Fetchen immer zu prunen:

```
$ git config --global fetch.prune true
```

Jetzt wird jedes `git fetch` und `git pull` automatisch veraltete Referenzen aufräumen!

Vorteile:

- Eine Sache weniger, an die Sie denken müssen
- Immer eine saubere Ansicht der Branches
- Keine veralteten Referenzen, die Ihre `git branch -a` Ausgabe überladen

Prüfe, ob Auto-Prune aktiviert ist

```
$ git config --global --get fetch.prune  
true
```

Wenn es **true** zurückgibt, sind Sie startklar!

Wenn es nichts zurückgibt, aktiviere es:

```
$ git config --global fetch.prune true
```

Was ist mit lokalen Branches?

Wichtig: Remote-Pruning betrifft nur **Remote-Tracking Branches** (`remotes/origin/*`), nicht Ihre **lokalen Branches**.

Sie müssen lokale Branches manuell löschen:

```
$ git branch -d feature/my-merged-feature # Löschen wenn gemerged  
$ git branch -D feature/my-merged-feature # Erzwungenes Löschen
```

Warum Git lokale Branches behält:

- Sie könnten uncommittete Arbeit haben
- Sie möchtest den Branch später vielleicht referenzieren
- Es ist sicherer, explizites Löschen zu verlangen

Häufiges Szenario 1: "Not Fully Merged" Fehler

```
$ git branch -d feature/xyz  
error: The branch 'feature/xyz' is not fully merged.  
hint: If you are sure you want to delete it, run 'git branch -D feature/xyz'.
```

Warum das passiert:

PR wurde mit "Squash and merge" oder "Rebase and merge" gemerged – das erstellt neue Commits anstatt Ihre originalen Commits zu verwenden.

Lösung: Verwende großes `-D` zum erzwungenen Löschen:

```
$ git branch -D feature/xyz
```

! Prüfe zuerst, dass der PR tatsächlich auf GitHub gemerged wurde, bevor Sie erzwungen löschen!

Häufiges Szenario 2: Welche Branches sind gemerged?

Zeige Branches, die in den aktuellen Branch gemerged wurden:

```
$ git branch --merged
```

Zeige Branches, die in main gemerged wurden:

```
$ git branch --merged main
```

Zeige Branches, die NOCH NICHT gemerged wurden:

```
$ git branch --no-merged
```

Häufiges Szenario 3: Massen-Aufräumen

Löschen Sie alle lokalen Branches, die in main gemerged wurden:

```
$ git branch --merged main | grep -v "^\"* main\" | xargs git branch -d
```

! Warnung: Führen Sie dies nur aus, wenn Sie sicher bist, dass alle aufgelisteten Branches tatsächlich gemerged sind!

Besserer Ansatz: Löschen Sie Branches einzeln, um sicher zu sein.

Warum Aufräumen wichtig ist

Vorteile sauberer Branches:

-  **Klarheit:** `git branch -a` zeigt nur Branches, die tatsächlich existieren
-  **Schneller:** Weniger Unordnung bedeutet einfacher zu finden, was Sie brauchst
-  **Professionell:** Zeigt, dass Sie Gits Architektur verstehst
-  **Verhindert Fehler:** Sie wirst nicht versehentlich an gelöschten Branches arbeiten

Es ist nicht kritisch: Veraltete Referenzen brechen nichts – sie verursachen nur Verwirrung.

Die Drei-Ebenen-Synchronisation verstehen

```
GitHub (zentrale Wahrheit)
  ↓ git fetch/pull
Remote-Tracking Branches (lokale Lesezeichen)
  ↓ git merge/checkout
Lokale Branches (Ihre Arbeit)
```

Jede Ebene braucht separates Aufräumen:

- GitHub: Branches werden automatisch nach PR-Mergen gelöscht
- Remote-Tracking: Verwende `git fetch --prune` oder `git pull --prune`
- Lokal: Verwende `git branch -d` manuell

Kurzreferenz: Aufräum-Befehle

```
# Alle Branches anzeigen (inkl. Remote-Tracking)
```

```
git branch -a
```

```
# Vorschau, was gepruned würde
```

```
git remote prune origin --dry-run
```

```
# Veraltete Remote-Tracking Branches entfernen
```

```
git fetch --prune
```

```
# Oder nur prunen ohne fetchen
```

```
git remote prune origin
```

```
# Gemergten lokalen Branch löschen
```

```
git branch -d feature/merged-branch
```

```
# Lokalen Branch erzwungen löschen (wenn squash gemerged)
```

```
git branch -D feature/merged-branch
```

```
# Automatisches Pruning aktivieren (empfohlen)
```

```
git config --global fetch.prune true
```

Der vollständige Post-PR Workflow

Nachdem Ihr PR auf GitHub gemerged wurde:

```
# 1. Zu main wechseln
git checkout main

# 2. Neueste Änderungen holen und Remote-Tracking Branches aufräumen
git pull --prune

# 3. Ihren lokalen Feature-Branch löschen
git branch -d feature/my-merged-feature
# (oder git branch -D wenn squash gemerged)

# 4. Aufräumen verifizieren
git branch -a
```

Sauberes Repository, bereit für das nächste Feature!

Wichtigste Erkenntnisse

1. **Remote-Tracking Branches** sind lokale Lesezeichen zu GitHubs Branches
2. **Veraltete Referenzen** entstehen, wenn Branches auf GitHub gelöscht werden
3. `git fetch --prune` entfernt diese veralteten Referenzen
4. **Lokale Branches** müssen separat mit `git branch -d` gelöscht werden
5. **Aktivieren Sie automatisches Pruning** mit `git config --global fetch.prune true`
6. **Aufräumen ist optional**, hält aber Ihr Repository aufgeräumt

Denken Sie an Branch-Aufräumen wie an das Aufräumen deines Schreibtischs – Ihre Arbeit hängt nicht davon ab, aber es macht alles klarer!

Übungsaufgabe

Probiere diesen Workflow aus:

1. Erstelle einen Feature-Branch und mache einen Commit
2. Pushen Sie zu GitHub und erstelle einen PR
3. Mergen Sie den PR (GitHub löscht den Branch)
4. Führe `git branch -a` aus (siehe veraltete Referenz)
5. Führe `git fetch --prune` aus (siehe, wie sie entfernt wird)
6. Löschen Sie lokalen Branch mit `git branch -d`
7. Aktivieren Sie Auto-Prune: `git config --global fetch.prune true`

Sie wirst nie wieder veraltete Referenzen haben!

Häufige Fragen

F: Wird Pruning meine Arbeit löschen?

A: Nein! Es entfernt nur Lesezeichen zu Branches, die nicht auf GitHub existieren.

F: Was wenn ich versehentlich prune?

A: Nichts Schlimmes passiert – Sie entfernst nur veraltete Lesezeichen.

F: Sollte ich immer Auto-Prune aktivieren?

A: Ja! Es ist eine Best Practice und hat keine Nachteile.

Weiterführende Literatur

Offizielle Git-Dokumentation

- [Git Branching - Remote Branches](#)
- [git-fetch Dokumentation](#)

GitHub-Dokumentation

- [About pull requests](#)
- [GitHub flow](#)

Interaktives Lernen

- [Learn Git Branching](#)

Profi-Tipp: Nach jedem gemergten PR `git pull --prune` ausführen und lokale Branches aufräumen!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

VIELEN DANK