git - Eine Zusammenfassung

Jonas Sander

4. September 2017

1 Konfigurieren (Aussehen und Funktion)

Die Konfigurationswerte werden an drei verschiedenen Orten gespeichert:

1. /etc/gitconfig (Mac: /etc in .private zu finden) enthält Werte, die für jeden Anwender des Systems und alle ihre Projekte gelten.

2. ~/.gitconfig (~entspricht deinem Benutzerordner) enthält Werte, die ausschließlich für dich und deine Projekte gelten.

3. .git/config im Git Verzeichnis deines Projektes enthält Werte die nur für das jeweilige Projekt gelten.

3 hat die höchste und 1 die geringste Priorität. Das heißt wenn Werte in 3 stehen, so werden diese vor den Werten aus 1 und 2 verwendet. Für 1 schreibe `- -system`, für 2 `- -global` und für 3 gar nichts an die Stelle `[Option]`

1.1 Benutzer anlegen

Unterdiesem Namen werden die Commits veröffentlicht!

```
Jonas$ git config [Option] user.name "Jonas Sander"
Jonas$ git config [Option] user.email jonas.sander@...
```

1.2 Die Ausgaben einfärben

```
Jonas$ git config [Option] color.diff true
Jonas$ git config [Option] color.ui true
Jonas$ git config [Option] color.status true
```

1.3 Zeilenende

`[Zeilenende]` → `input` für Mac/Linux → `true` für Windows

```
Jonas$ git config [Option] core.autocrlf [Zeilenende]
```

1.4 Dein Editor

`[Editor]` → gewünschter Editor z.B nano

```
Jonas$ git config [Option] core.editor [Editor]
```

1.5 Dein Diff Programm

`[DiffProg]` → gewünschtes Diff Programm z.B vimdiff

```
Jonas$ git config [Option] merge.tool [DiffProg]
```

1.6 Deine Einstellungen anzeigen

Zeigt alle Einstellungen, die Git an dieser Stelle (z.b in einem `Projekt`) bekannt sind. Manche Variablen werden eventuell mehrfach aufgelistet, da Git sie an mehreren Orten findet. In diesem Fall verwendet Git den zuletzt aufgelisteten Wert.

```
Projekt Jonas$ git config --list
```

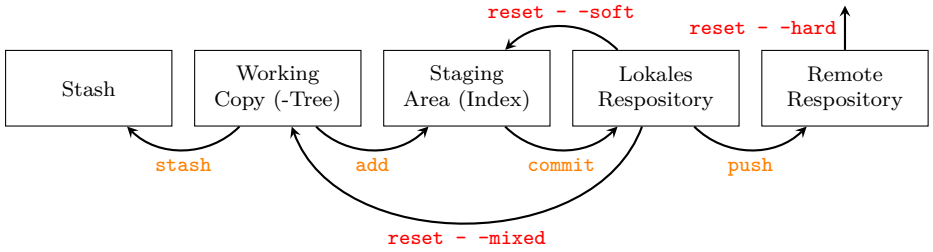
Zeigt den Wert für eine bstimme Variable → `[Variable]`

```
Jonas$ git config [Variable]
```

1.7 SSH-Keys

1. Wo liegen die Keys: /Users/JonasSander/.ssh (Versteckte Ordner im Finder anzeigen: cmd+shift+.)

2 Git Aufbau - Wo landen die Commits?



3 Mit Git arbeiten

3.1 Repos erstellen

```
$ git clone [Adresse]
$ git init
```

3.2 Lokale Änderungen

Veränderungen im Working Copy

```
$ git status
```

Änderungen seit dem letzten Commit; `[Option]` → ohne, die noch nicht in der Stage Area sind → `- - staged`, die bereits in der Stage Area sind → CommitID-1 CommitID-2, Vergleich zwischen zwei Commits

```
$ git diff [Option]
```

Lokale Änderungen zur Staging Area hinzufügen. `[Dateiname]` → . fügt alle Dateien aus der Working Copy in die Satging Area hinzu.

```
$ git add [Dateiname]
```

`[Option]` → Zur Staging Area hinzugefügte Dateien commiten. → `-a` Dateien aus Working Copy direkt commiten (ohne `add`-Befehl). → `- - amend` letzten/neusten commit ändern (Breits veröffentlichte Commits nicht änderbar!)

```
$ git commit [Option]
```

3.3 Die Commit Historie

`[Option]` → ohne, alle Commits chronologisch anzeigen. → `-p` zeigt die Änderungen in einem Commit, ergänzt man zudem noch einen `<file>`, so werden alle Änderungen in diesem File über alle Commits ausgegeben. → `-x` zeigt die letzten x Commits an.

```
$ git log [Option]
```

Wer veränderte was und wann in `<file>`.

```
$ git blame [file]
```

3.4 Branches in Git

Branches sind einfach gesagt Verweise auf bestimmte Commits.
Der **HEAD** ist ein Alias für den Commit der gerade ausgecheckt ist. HEAD zeigt immer auf den neusten Commit. Normalerweise zeigt HEAD auf einen Branch. Wir nun ein Commit hinzugefügt, so wird der Branch auf diesen Commit angehoben (da der HEAD ja gerade auf diesen zeigt).

`[Option]` → `-av` alle Branches auflisten. → `<Branch-Name>` erstellen eines neuen lokalen Branch (auf aktuellem HEAD) → `-d <Branch-Name>` lokalen Branch löschen.

```
$ git branch [Option]
```

Aktuellen HEAD-Branch wechseln

```
$ git checkout [Branch-Name]
```

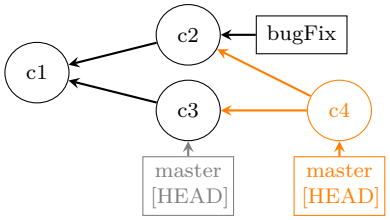
3.5 Zusammenführen von Commits

Merge von `[Branch-Name]` in den aktuellen HEAD.

```
$ git merge [Branch-Name]
```

Beispiel:

```
$ git merge bugFix
```

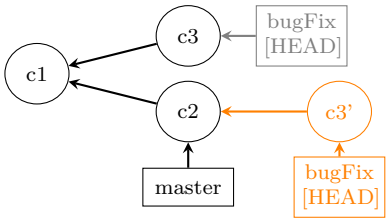


Rebase von `[Branch-Name]` auf HEAD.

```
$ git rebase [Branch-Name]
```

Beispiel:

```
$ git rebase master
```



Was in bugFix gemacht wurde, haben wir jetzt oben auf master drauf gepackt und haben eine lienare Abfolge von Commits erhalten. c3 existiert immer noch. c3' ist prinzipiell bloß eine Kopie von c3 die zusätzlich c2 enthält.

3.6 Navigation durch Git

"Detached HEAD state" bezeichnet den Zusatand, in dem ein bestimmter Commit anstelle eines Branches ausgecheckt ist. **Achtung:** In diesem Zustand committete Änderungen werden von keinem Branch referenziert und können so leicht verloren gehen, wenn man sich nicht gerade den Hash des Commits merkt. **Anwendungsfall:** Diese Methode bietet sich an, falls man in der Zeit zurück gehen möchte um eine frühere Version seines Projektes zu testen.
Um sich aber nicht in die Schwierigkeiten dieser Methode zu manövrieren, bietet es sich an temporär einen neuen Branch anzulegen.

```
$ git checkout [Commit-Hash]
```

Nur mit Hashwerten durch die Git-Historie zu navigieren kann sehr mühselig sein. Mit **relativen Referenzen** kann man bei einprägsamen Branches oder dem HEAD beginnen. `^` checkt den Vorgänger-Commit aus. `^^` den Vorvorgänger usw. `~x` checkt den x-ten Vorgänger aus.

```
$ git checkout master^
$ git checkout master~[Anzahl]
```

Das **Verschieben von Branches** auf einen bestimmten Commit, ohne vorheriges auschecken des betreffenden Branches, gelingt mit folgendem Befehl:

```
$ git branch -f [Branch-Name] [Ziel-Commit]
```

Beispiel:
Bewegt den Branch master auf den Commit drei Vorgänger vor HEAD.

```
$ git branch -f master HEAD~3
```

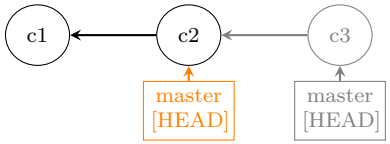
3.7 In Git etwas rückgängig machen

`git reset` nimmt **lokale Änderung** zurück indem es eine Branch-Referenz auf einen anderen Commit setzt.
`[Option]` → ohne oder `- -mixed` nimmt Commit zurück in die Working-Copy. → `- -soft` nimmt Commit zurück in die Staging Area. Nützlich wenn man etwas im Commit vergisst. → `- -hard` verwirft alle Änderungen seit einem Commit.

```
$ git reset [Option] [CommitID]
```

Beispiel:

```
$ git reset --hard HEAD~1
```

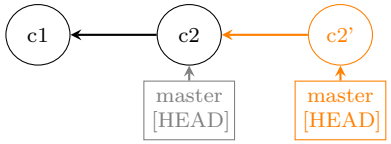


Während sich `git reset` gut im lokalen Kontext verwenden lässt, ist es für Änderung von Commits die bereist im Remote Respository liegen ungeeignet. Um **Änderungen auf dem Remote Respository** durchzuführen und somit mit anderen zu teilen, verwendet man folgenden Befehl:

```
$ git revert [CommitID]
```

Beispiel:

```
$ git revert HEAD
```



Mit c2' werden die Änderungen aus c2 rückgängig gemacht.

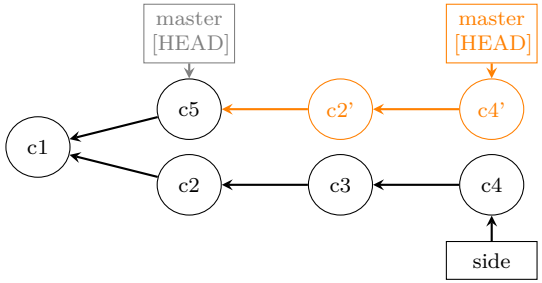
3.8 Inhalte verschieben

Eine Folge von Commits unter den HEAD kopieren.

```
$ git cherry-pick [CommitID-1] [CommitID-2] [CommitID-...]
```

Beispiel:

```
$ git cherry-pick c2 c4
```



3.9 Remote Respositories

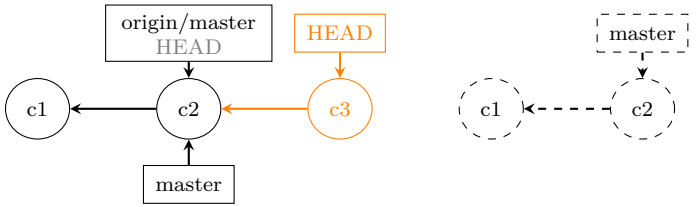
Ein remote Respository clonen.

```
$ git clone [URL]
```

Der Branch master dient zur Veranschaulichung und wurde nicht geclont. **Der Branch origin/master ist ein Remote Branch.** Remote Branches bilden den Zustand des entsprechenden Branches in einem Remote Respository ab (dem Zustand in dem der Branch war, als du zuletzt gefetched hast). Der Remote Branch hilft, den Unterschied zwischen einem lokalen Branch und dem entfernten Gegenstück auf dem Server darzustellen.
Remote Branches besitzen die Eigenschaft dein Respository in den "detached HEAD state„ zu versetzen wenn sie ausgechecked werden. Git tut dies absichtlich, denn so kann man nicht direkt auf Remote Branches arbeiten. Man muss auf Kopien von ihnen Arbeiten und die Änderungen von dort auf den entfernten Server schieben (wonach der Remote Branch dann auch bei einem selbst aktualisiert wird).

Remote Branch auschecken und committen - Ein Beispiel:

```
$ git checkout origin/master
$ git commit
```



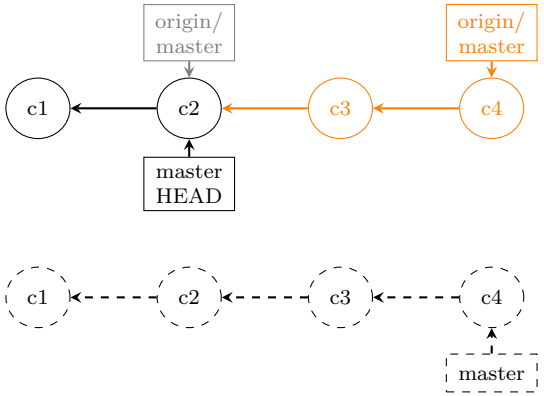
Git wird in den Detached Head Mode versetzt und aktualisiert nach dem Committen nicht den Brach origin/master. Schließlich soll nicht direkt auf Remote Branches gearbeitet werden (siehe vorheriger Kasten).

`[Option]` → `add <shortname> <URL>` erstes oder weiteres remote Respository anbinden. → `set-url <URL>` Repository-URL updaten. → `-v` alle konfigurierten remote Respositorys listen. → `show <remote>` Infos über ein remote Respository ausgeben.

```
$ git remote [Option]
```

`git fetch` **lädt die Commits herunter die im lokalen Respository fehlen und aktualisiert die Remote Branches** wenn nötig. Der Befehl synchronisiert im Prinzip unsere lokale Abbildung des entfernten Repositorys mit dem wie das entfernte Respository in diesem Moment wirklich aussieht. `git fetch` ändert aber nichts an deinen lokalen Branches oder deinem Checkout.

```
$ git fetch
```



Der Befehl fetch mit folgende Optionen funktioniert ähnlich wie der Befehl push mit den selben Optionen. Nur das bei push Dateien hoch- und bei fetch heruntergeladen werden.

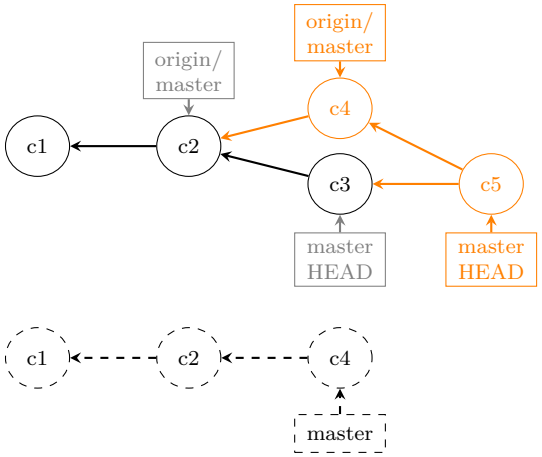
```
$ git fetch [Remote] [Ort]
$ git fetch [Remote] [Quelle:Ziel]
```

Crazy: Lässt man das Ziel weg, wir ein neuer lokaler Branch erstellt :D

Mit `git fetch` können wir Daten vom entfernten Respository holen. Anschließend lassen sich diese Daten mit `git cherry-pick origin/...`, `git rebase origin/...` oder `git merge origin/...` in das lokale Respository integrieren. Herunterladen und Integrieren auf einmal funktioniert mit `git pull`. Folgende Befehle tun also das gleiche:

```
$ git fetch
$ git merge origin/master

$ git pull
```



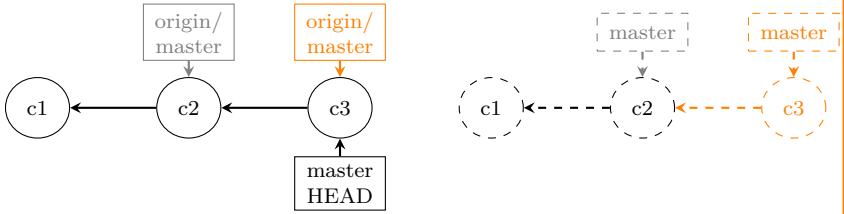
Der Befehl pull mit folgende Optionen funktioniert ähnlich wie der Befehl push mit den selben Optionen. Nur das bei push Dateien hoch- und bei pull heruntergeladen und gemerget werden.

```
$ git push [Remote] [Ort]
$ git push [Remote] [Quelle:Ziel]
```

Existiert das Ziel nicht, so wird ein neuer Branch erstellt.

Dateien zum Remote Respository hinzufuegen:

```
$ git push
```



Das Remote Respository hat den Commit c3 bekommen. Der Branch master im Remote Respository ist aktualisiert worden. Unsere eigene Abbildung des master Branches vom Remote Respository names origin/master ist ebenfalls aktualisiert worden.
Zeigt der HEAD nicht auf einen Branch, der einen Remote Branch trackt, so schlägt der Befehl fehl. Soll ein Branch der nicht ausgecheckt ist gepusht werden, so verwende push mit folgenden Optionen. Folgender Befehl würde den loakelen Branch master auf den Branch master im Remote Respository pushen ohne das master im lokalen Respository ausgecheckt sein muss *(Vorauss.: Der lokale master*

Branch trackt den Remote Branch origin/master).

```
$ git push origin master
```

Und nochmal in allgemeiner Notation:

```
$ git push [Remote] [Ort]
```

Und was wenn die Quelle eine andere sein soll als das Ziel? Dann verwenden wir **Refspec (Referenzspezifikation)**:

```
$ git push [Remote] [Quelle:Ziel]
```

Lässt man die Quelle weg, so wird der entfernte Branch (und sein Remote Branch) gelöscht.
Beispiel:

```
$ git push origin foo^:master
```

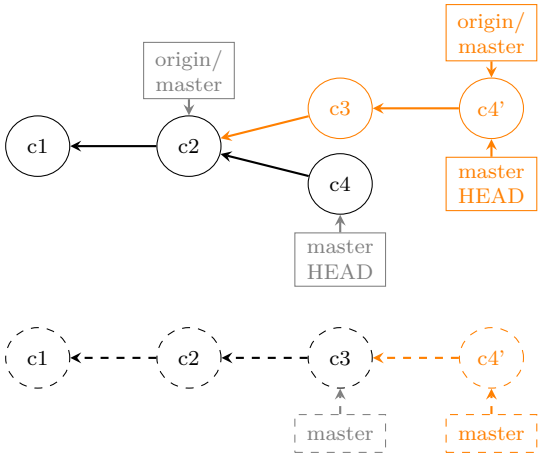
Git löst foo^ zu einem Commit auf, integriert alle einschließlich diesem, die noch nicht im entfernten master Branch waren, in den entfernten master Branch. *Praktisch: Wenn der Ziel Branch noch nicht existiert, so wird er erstellt!*

Dateien zum Remote Respository bei abweichenden Historien hinzufügen: Unterscheidet sich die Historie deines lokalen Repositorys von der des Remote Respositorys, so erlaubt Git keine einfach Ausführung von `git push`. **Zuvor müssen die Änderungen vom Remote Respository heruntergeladen und in dein lokales Respository integriert werden.** Dann kann `git push` ausgeführt werden.
Wie wird das aufgelöst?

1. **Commits per Rebase verschieben:** Folgende Befehlssätze führen zu gleichem Ergebnis.

   ```
   $ git fetch
   $ git rebase origin/master
   $ git push
   ```

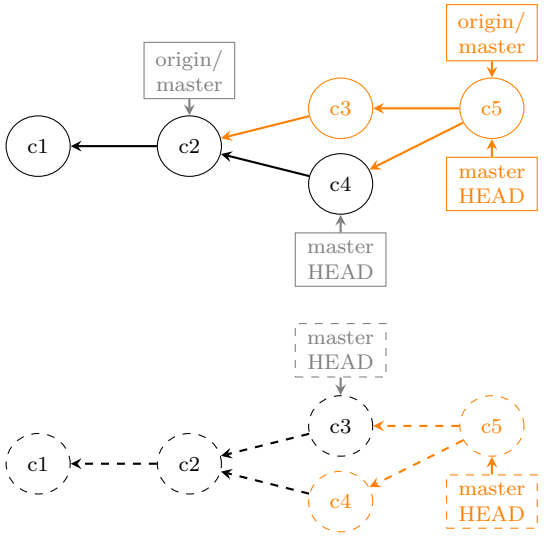
   ```
   $ git pull --rebase
   $ git push
   ```



2. **Commits per Merge verschieben:** Folgende Befehlssätze führen zu gleichem Ergebnis.

   ```
   $ git pull
   $ git merge origin/master
   $ git push
   ```

   ```
   $ git pull
   $ git push
   ```



Remote Branches tracken:
Erstellen eines neuen Branches, der einen Remote Branch trackt.

```
$ git checkout -b [Branch-Name] [Remote Branch-Name]
```

Lokalen Branch einen Remote Branch tracken lassen. [Branch-Name] kann weg gelassen werden, falls der entsprechende Branch gerade ausgecheckt ist.

```
$ git branch -u [Remote Branch-Name] [Branch-Name]
```

3.10 Nicht enthalten

- Interactive Rebase
- Git Tags