



http://ovm-kassel.de Information	
Information AE-TOOLS-I-1.1 Git	
Code	AE-TOOLS-I-1.1
Autor	André Bauer <a(dot)bauer(at)ovm-kassel(dot)de>
Datum	26. September 2018
Links	<ul style="list-style-type: none"> • Bücher <ul style="list-style-type: none"> ◦ Scott Chacon und Ben Straub: Pro Git Buch ◦ John Wiegley: Git from the Bottom Up ◦ Scott Chacon: Git Internals • Cheat Sheets <ul style="list-style-type: none"> ◦ GitHub Git cheat sheet ◦ GiLab Git cheat sheet ◦ Tower Git cheat sheet • Wikipedia-Eintrag zu Git
Lizenz	 Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz .

Git

1. Das Konzept

Git ist ein verteiltes Versionskontrollsystem (DVCS). Es ist als Entwicklerwerkzeug im Hinblick auf Quelltexte entwickelt worden, kann aber auch für andere Projekte und Daten verwendet werden.

Git speichert Änderungen in Form von Snapschüssen in eine Repository. Im Unterschied zu zentralisierten Versionskontrollsystemen enthält jedes Git-Repository sämtliche Snapschüsse, so dass die meisten Aktionen vollständig lokal ausgeführt werden können und damit auch ohne Netzwerkverbindung Änderungen gespeichert werden können. In einem Projekt-Ordner wird nach der Initialisierung ein Unterverzeichnis **.git** erstellt, das das Repository enthält. Alle anderen Dateien und Unterverzeichnisse bilden das Arbeitsverzeichnis (working tree).

Für Änderungen sieht Git ein zweistufiges Verfahren vor: Änderungen werden zunächst vorgemerkt (staged) und dann gespeichert (commit). Dies ermöglicht eine sehr feine Kontrolle, welche Änderungen in einen Commit einfließen, so dass inhaltlich zusammengehörige Änderungen in

einem Commit zusammengefasst werden können.

2. Ein neues Projekt erstellen (initialisieren)

```
$ mkdir myproject  
$ cd myproject  
$ git init
```

3. Konfiguration

3.1. Globale Konfiguration

```
$ git config --global user.name "André Bauer"  
$ git config --global user.email a.bauer@ovm-kassel.de  
$ git config --list
```

3.2. Lokale Konfiguration

In einem bestehenden Projekt.

```
$ cd myproject ①  
$ git config --local user.email a.bauer@ovm-kassel.de
```

① Das Verzeichnis `myproject` muss bereits unter Versionskontrolle durch Git stehen.

4. Lokaler Workflow

4.1. Neue Datei erstellen

```
$ echo "Hallo git." > hallo ①  
$ git add hallo ②  
$ git commit -m "Add hello." ③
```

① Erstellt eine einfache Textdatei.

② Die Datei `hallo` wird nun von Git versioniert und ist nun für Änderungen vorgemerkt (staged).

③ Übernimmt die vorgemerkten Änderung in das Repository (commit) mit der angegebenen Nachricht.

4.2. Änderungen

Der Ablauf ist wie bei einer neuen Datei. Nach einer Änderung wird diese mit `git add` vorgemerkt und mit `git commit` dauerhaft im Repository gespeichert.

```
$ echo "Hallo git2." >> hallo
$ git add hallo
$ git commit -m "Add hello."
```

4.3. Status anzeigen

Der aktuelle Status, d.h. welche Dateien unversioniert oder geändert wurden und welche Änderung vorgemerkt wurden, kann mit `git status` angezeigt werden.

```
$ git status
```

Die Änderungen, die in das Repository übernommen wurden, können mit `git log` angezeigt werden.

```
$ git log
```

Für diesen Befehl gibt es eine große Anzahl von Parametern, um das Format und den Umfang der Ausgabe festzulegen.

4.4. Änderungen anzeigen

```
$ git diff ①
```

① Zeigt Änderungen im diff-Format an, die noch nicht vorgemerkt wurden.

```
$ git diff --staged ①
$ git diff --cached ②
```

① Zeigt Änderungen im diff-Format an, die vorgemerkt wurden.

② Statt `--staged` kann auch `--cached` verwendet werden.

5. Remotes

5.1. Hinzufügen

Mit einem lokalen Repository können andere Repositories verknüpft werden, die sogenannten Remotes. Dazu erstellt man z.B. unter GitHub oder GitLab ein neues Repository und fügt mit `git remote add` diese dem Bestehenden Repository hinzu.

```
$ git remote add github git@github.com:andrebauer/git-beispiel.git
$ git remote add gitlab git@gitlab.com:andrebauer/git-beispiel.git
$ git remote
github
gitlab
```

5.2. Workflow

Mit **git push**, **git fetch** und **git pull** werden Änderungen (commits) mit remote ausgetauscht.

```
$ git push -u gitlab ①
$ echo "Hallo Git!" >> hallo ②
$ git add hallo ③
$ git commit -m "Add hello message to hallo." ④
$ git push gitlab ⑤
```

- ① Füge die Änderungen aus dem aktuellen Branch, dieser heißt standardmäßig master, dem Remote gitlab hinzu. Falls der Zweig noch nicht auf dem Remote existiert, so wird diese bei Verwendung der Option **-u** angelegt.
- ② Bearbeiten der Datei hallo.
- ③ Vormerken der Änderungen in hallo.
- ④ Übernehmen der Änderungen.
- ⑤ Die Änderungen auf den Remote gitlab übertragen.

5.3. Änderungen von anderen übernehmen

```
$ git fetch gitlab ①
$ git merge gitlab/master
```

- ① Holt Änderungen von anderen vom Remote gitlab.
- ② Mischt die Änderungen von Branch gitlab/master mit dem lokalen Branch master.

Mit **git pull** können **git fetch** und **git merge** in einer Aktion ausgeführt werden.

```
$ git pull gitlab
```

Bei **git merge** und **git pull** können Konflikte auftreten, die Git nicht automatisch beheben kann, wenn z. B. Änderungen in derselben Zeile eines Programms durch verschiedene Parteien eingebracht wurden. Git fügt dann mithilfe von Markierungen beide Varianten in den betroffenen Dateien ein, der Merge muss dann manuell vollendet werden.

5.4. Ein vorhandenes Projekt klonen

```
$ git clone git@github.com:andrebauer/git-beispiel.git ①
```

① Überträgt das Repository in das Verzeichnis **git-beispiel**.

5.5. Einen SSH-Schlüssel erstellen

Um mit Remotes ohne Passworteingabe arbeiten zu können, kann ein SSH-Schlüsselpaar erstellt und der öffentliche Teil bei GitLab bzw. GitHub hinterlegt werden.

- <https://docs.gitlab.com/ce/ssh/README.html>
- <https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>