



# Git & Github Anfänger Tutorial

Autor

Marc Borkowski

Datum

04.10.2022

## Übersicht

- [wie Git funktioniert](#)
- [Arbeiten mit anderen](#)
- [Wie GIT Dateien speichert](#)
- [SSH Einrichten](#)
- [die Git Basics \(init, status, add und commit\)](#)
- [Repository auf Github schieben mit git push](#)
- [Dateien vom Server holen mit git pull](#)
- [Dateien aus der Datenbank ausschließen](#)
- [Datei Änderungen rückgängig machen](#)
- [Staged Markierung entfernen](#)
- [Stash](#)
- [Branches](#)
- [Branches mergen](#)
- [GitHub Page](#)

## wie Git funktioniert

Git ist ein Versionierung Werkzeug, mit solch einem Werkzeug ist es möglich, Versionen von Dateien zu erstellen und in einer Datenbank zu hinterlegen. Praktisches Beispiel: Du hast keine Ordner mehr mit den Namen V1, V2 usw., mit Git ist es möglich in einem Ordner mehrere Versionen von einer Datei zu speichern und zu verwalten. Git ist nicht nur für Softwareentwickler geeignet, sondern auch für Designer, da mit Git auch Bilder gut verwaltet werden können. Du bekommst eine History für deine Dateien und kannst auch nach mehreren Änderungen die Dateien wieder auf den alten Stand setzen.

## Arbeiten mit anderen

Des weiteren gibt es ein weiteren großen Vorteil bei Git, das Arbeiten mit anderen wird viel einfacher. Es ist möglich einen Git Server zu erstellen, auf diesen Server werden dann die Änderungen hochgeladen und jeder (der Zugriffsrechte hat) kann sich die aktuelle Änderung runterladen. Das charmante an der Sache ist, dass nicht jedesmal das komplette Projekt runtergeladen werden muss, sondern nur die Änderung wird runtergeladen. Des weiteren wird der Git Server auf den lokalen Maschinen von den Designern und Programmieren gespiegelt. Das hat den Vorteil, dass bei einem Serverausfall, jeder weiter arbeiten kann. Falls der Git Server ein Festplattenfehler hat, sind die Daten auch nicht weg, da ja jeder ein Backup von diesem Server bei sich hat und aus diesem Backup wieder ein neuer Server erstellt werden kann.

## Wie GIT Dateien speichert

Wenn du eine neue Version von dem Quellcode hast und den Code dann auf Git ablegst, wird nicht jedes mal der komplette Code gespeichert. Es wird nur die geänderte Datei gespeichert und auf die nicht geänderten Dateien wird in der neuesten Version deines Quellcodes nur gezeigt. Sprich es werden Snapshots von denen Quellcode erstellt und diese ergeben dann die aktuelle Version.

## Snapshots



## die 3 Datei Status

**Jetzt kommt ein wichtiger Part, diesen solltest du dir einprägen!** Bei Git gibt es 3 Status, modified, staged, committed. Modified ist eine Datei, wenn sie bearbeitet wurde aber noch nicht in der Datenbank hinterlegt ist. Staged ist eine Datei, wenn sie als fertig markiert wurde und in den Commit rein kann. Committed ist eine Datei, wenn sie lokal in der Datenbank abgelegt wurde. Das bedeutet, du bearbeitest eine Datei, dann ist sie modified. Danach markierst du diese Datei und sie ist staged. Dann erstellst du ein commit mit allen staged Dateien und diese rutschen dann in die Datenbank.

## Video Tutorial



## SSH Einrichten

In diesem Tutorial werde ich nur mit der Konsole arbeiten, sprich du brauchst erstmal die Git Bash. Auf die Installation gehe ich nicht weiter ein, da es hier sehr gut erklärt wird: <https://git-scm.com/downloads>. Um zu testen ob Git richtig installiert wurde, öffnest du Git Bash und gibst folgenden

Befehl ein:

```
git --version
```

Bei mir kommt dann folgende Meldung:

```
git version 2.13.0.windows.1
```

Es ist kein Problem wenn du eine höhere Version hast, da die Git Entwickler auf Abwärtskompatibilität achten.

Es ist wichtig Git mitzuteilen wer man ist, da dies bei den Commits mitgespeichert wird. Dies hat den Vorteil, dass im Verlauf der Programmierer angezeigt wird und dadurch der passende Ansprechpartner für Änderungen am Code zur Verfügung steht. Um das zu machen, in der Konsole folgende Befehle eingeben:

```
git config --global user.name "Snowflame"  
git config --global user.email github@boolie.org
```

Um deine Einstellungen zu prüfen gibst du folgenden Befehl ein:

```
git config --list
```

```
core.symlinks=false  
core.autocrlf=true  
core.fscache=true  
color.diff=auto  
color.status=auto  
color.branch=auto  
color.interactive=true  
help.format=html  
rebase.autosquash=true  
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt  
diff.astextplain.textconv=astextplain  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.required=true  
filter.lfs.process=git-lfs filter-process  
credential.helper=manager  
user.email=github@boolie.org  
user.name=Snowflame  
credential.helper=manager
```

Jetzt brauchst du noch ein Server, zum testen und ausprobieren, ich empfehle dir [GitHub](#). Sicherheit geht vor! Deswegen sichern wir die Kommunikation mit GitHub erstmal ab. Dafür brauchst du ein rsa Schlüssel, den kannst du in der Konsole generieren.

```
ssh-keygen -t rsa -f ~/.ssh/id_rsa_github -C "github@boolie.org"
```

Jetzt musst du ein Passwort eingeben, mit diesem schaltest du später immer dein privaten Schlüssel frei, also gut merken. Jetzt sollte der Schlüssel generiert worden sein, schaue im .ssh Ordner nach, ob dort 2 Schlüssel (id\_rsa\_github und id\_rsa\_github.pub) liegen

## .ssh Config anlegen

Damit dein Rechner auch weiss, welcher Key verwendet werden soll, kannst du ein Config Datei anlegen mit einem Host.

**~/ssh/config:**

```
Host github.com  
    IdentityFile ~/.ssh/id_rsa_github
```

## Verbindung aufbauen

Bei mir wird der `id_rsa_github.pub` als Libre Office Dokument angezeigt da die Endung dort verwendet wird. Jetzt öffnest du `id_rsa_github.pub` mit dem Texteditor und kopierst den kompletten Inhalt. Jetzt gehst du auf <https://github.com/settings/keys> und trägst dort unter „new SSH Key“ deinen Schlüssel ein. Um zu testen ob der Schlüssel funktioniert, gibst du in der Konsole folgendes ein:

```
ssh -T git@github.com
```

„Are you sure you want to continue connecting (yes/no)?“ mit yes bestätigen und dein Passwort eingeben. Danach sollte „Hi [USERNAME]! You've successfully authenticated, but GitHub does not provide shell access.“ kommen und auf der Github Seite <https://github.com/settings/keys> ist das Schlüssel Symbol von schwarz auf grün geändert.

FERTIG! Du hast dein Git und Github erfolgreich konfiguriert.

## die Git Basics (init, status, add und commit)

Zuerst wollen wir ein Projekt anlegen

### git init

Git speichert Projekte in Repositorys, übersetzt heißt das Lager, Depot oder Quelle. Es gibt 2 Varianten eine Repository zu erhalten:

- clonen einer existierenden Repository
- eine lokale Repository erstellen

Jetzt erkläre ich erstmal wie du eine lokale Repository erstellst und später wie du eine existierende clonst.

### initialisieren einer lokalen Repository

Öffne deine Konsole und steure mit `cd` dein Zielordner an, Beispiel:

```
cd Desktop\Projekte\boolie
```

Jetzt brauchst du noch ein Unterordner, den erstellst du mit `mkdir`:

```
C:\Users\Marc\Desktop\Projekte\boolie> mkdir gittut
```

Jetzt wechselst du, mit `cd`, in den Ordner:

```
C:\Users\Marc\Desktop\Projekte\boolie> cd gittut
```

Und nun tippst du `git init` in die Konsole.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git init
Initialized empty Git repository in C:/Users/Marc/Desktop/Projekte/boolie/gittut/.
```

### Datei anlegen und git status

Jetzt wurde die Datenbank erstellt, sie liegt im Ordner „.git“. Dort werden jetzt deine Snapshots gespeichert. Jetzt öffne deinen bevorzugten Editor, in meinem Fall ist das Atom, und erstelle eine Datei namens `README.md`. Schreibe noch nichts rein erstmal nur erstellen und in den Ordner speichern. In der Konsole gibst du jetzt den Git Status befehl ein:

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
```

```
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
```

## git add

Jetzt wird die README.md angezeigt, eine „untracked“ Datei, das bedeutet das Git diese Datei ignoriert und sie nicht in die Repository speichert. Das soll jetzt geändert werden, mit dem add Befehl kannst du die Datei in den staged Status versetzen.

```
git add README.md
```

## git commit

Jetzt muss die README.md Datei noch in die Datenbank eingetragen werden, da ja mit staged nur eine Markierung gesetzt wurde. Um die Datei jetzt in die Datenbank zu schieben muss ein commit gemacht werden:

```
git commit -m "README erstellt"
```

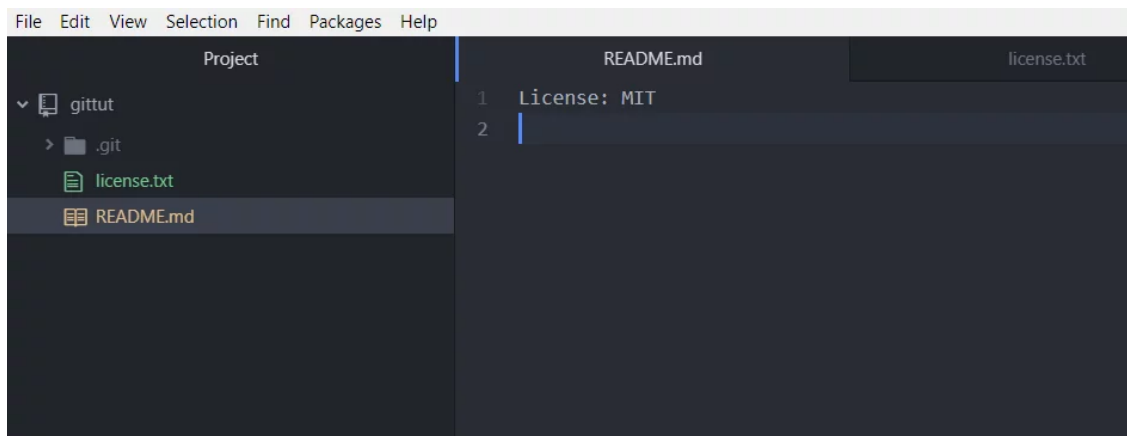
Jetzt ist die Datei in der Datenbank. Das -m steht für message, sprich ein Kommentar in dem stehen sollte was geändert würde. Später kann man dann im Verlauf nachschauen, was für Änderungen am Projekt vorgenommen wurde. Mit dem Befehl log kannst du dir die Änderungen an der Repository anschauen.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git log
commit 3612a834ae48596e0cdf697a82c823a037b90efa (HEAD -> master)
Author: Marc <borkowskiberlin@gmail.com>
Date:   Sat Jul 28 10:35:41 2018 +0200

    README erstellt
```

## mehrere Dateien ändern

Um diesen Prozess einzuprägen, werden wir jetzt mehrere Dateien in die Datenbank eintragen. Erstelle in deinem Editor die Datei license.txt und bearbeite die README.md, trage einfach ein Hinweis auf die Lizenz ein.



Nun schaue dir die Änderungen in Git, mit dem Befehl status an.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        license.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Wie du siehst, werden beide Dateien angezeigt. Die README.md als modified und die license.txt als untracked. Setze beide Dateien jetzt in den staged Modus. Wieder mit dem Befehl add. Danach schaue dir die Änderungen wieder mit status an.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git add README.md
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git add license.txt
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
        new file:   license.txt
```

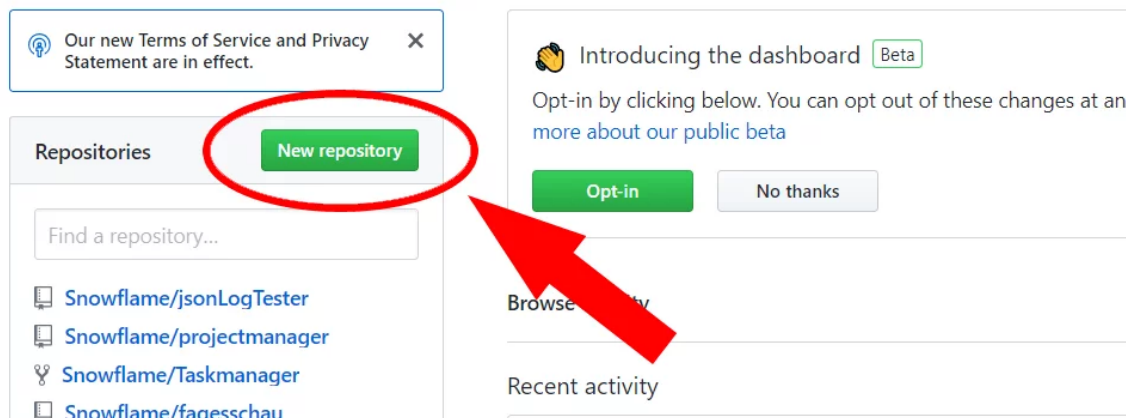
Gut, beide Dateien sind im staged Modus, jetzt der commit. Wieder mit dem Befehl commit -m „added a license“.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git commit -m "added a license"
[master 6c5b8f4] added a license
 2 files changed, 1 insertion(+)
 create mode 100644 license.txt
```

## Repository auf Github schieben mit git push



### Repository erstellen


Öffne [github.com](https://github.com), und klicke auf „New Repository“.



## Create a new repository

A repository contains all the files for your project, including the revision history.

**Owner**  Snowflake / **Repository name**  

**Name eintragen** 


Great repository names are short and memorable. Need inspiration? How about [glowing-guide](#).


**Description (optional)**

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.





**hier drücken** 


Jetzt hast du eine Repository auf einem Server. Nun werden wir die lokal erstellte Repository auf den Server schieben. Wechsel wieder in die Konsole und setze, mit dem Befehl „remote add origin“, die URL für deine lokale Repository.

Snowflake / GitTutorial Unwatch 1 Star 0 Fork 0

[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Insights](#) [Settings](#)

**Quick setup — if you've done this kind of thing before**

 Set up in Desktop or **HTTPS** **SSH**  

**Hier ist deine URL** 

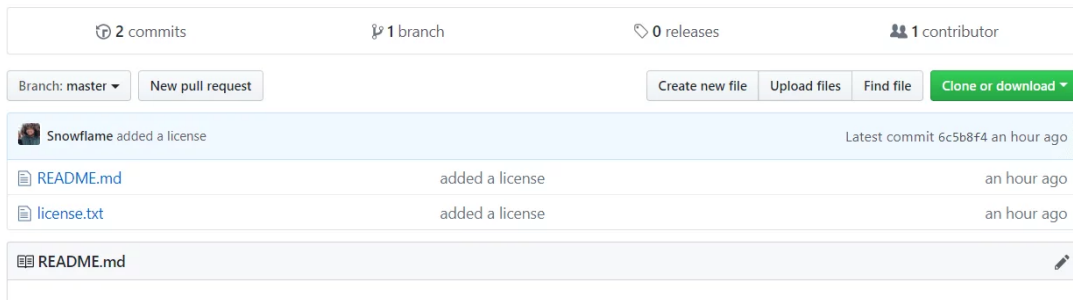
We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git remote add origin https://github.
```

Jetzt haben wir der lokalen Repository gesagt, dass es ein Server für dich gibt und du den unter der Adresse „https://github.com/Snowflake/GitTutorial.git“ findest. Jetzt müssen noch die Dateien hochgeladen werden, mit dem Befehl „push“, kannst du Dateien auf den Server schieben.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git push -u origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 481 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/Snowflake/GitTutorial.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Wenn du jetzt wieder die GitHub Seite öffnest, siehst du, dass deine Dateien auf dem Server liegen.



## Dateien auf den Server pushen

Jetzt bearbeiten wir die README.md und schieben die Änderungen auf den Server. Zuerst die Änderung, in deinem Editor, die README.md Datei bearbeiten, schreibe z.B. GitHub Tutorial, als Überschrift, rein. Wenn du dein Text schön machen willst, kannst du Markdown verwenden, hier ein schönes [Markdown Tutorial](#). Nachdem du jetzt deine README.md bearbeitet hast, musst du diese auf den Server schieben. Zuerst status, dann add, dann commit und push.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

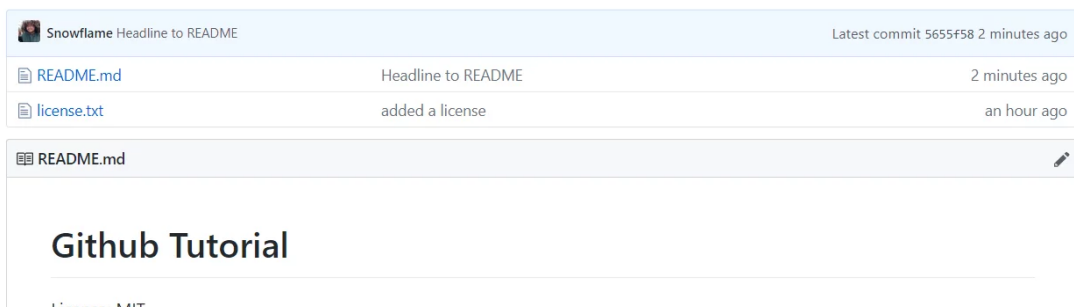
no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git add README.md

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git commit -m "Headline to README"
[master 5655f58] Headline to README
1 file changed, 1 insertion(+)

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Snowflake/GitTutorial.git
6c5b8f4..5655f58 master -> master
```

Fertig du hast die Änderungen auf den Server geladen.



## Passwort speichern



Wenn du nicht immer dein Passwort eingeben möchtest, trage eine remote URL ein.

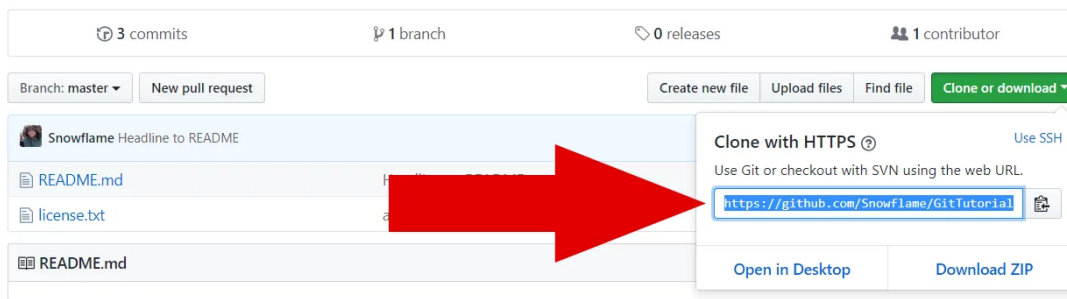
```
git remote set-url origin git@github.com:Snowflake/GitTutorial.git
```

## Dateien vom Server holen mit git pull

Wenn du jetzt auf einem Anderen Computer deine Dateien vom Server laden möchtest, musst du zuerst deine Repository clonen. Wechsel zuerst in ein anderes Verzeichnis. Mit „cd ..“ wechselst du in das darüber liegende Verzeichnis. Dann erstellst du wieder ein Ordner mit „mkdir“, schreibe einfach eine 2 hinter den Namen und wechsele in den Ordner.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>cd ..
C:\Users\Marc\Desktop\Projekte\boolie>mkdir gittut2
C:\Users\Marc\Desktop\Projekte\boolie>cd gittut2
C:\Users\Marc\Desktop\Projekte\boolie\gittut2>
```

Nun clonst du die Repository von Github auf dein lokalen Rechner. Gebe in die Konsole „git clone RepositoryURL“ ein, die URL findest du auf der GitHub Seite.



```
C:\Users\Marc\Desktop\Projekte\boolie\gittut2>git clone https://github.com/Snowflake/GitTutorial
Cloning into 'GitTutorial'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
```

Wechsel jetzt mit „cd“ in die Repository:

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut2>cd GitTutorial
```

Jetzt musst du auch in deinem Editor den Ordner wechseln, achte darauf, das du nicht das Oberverzeichnis auswählst, sondern die Repository. Ändere jetzt die README.md und lade sie auf den Server hoch.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut2\GitTutorial>git add README.md

C:\Users\Marc\Desktop\Projekte\boolie\gittut2\GitTutorial>git commit -m "new line"
[master 412ffc4] new line of nothing
1 file changed, 1 insertion(+)

C:\Users\Marc\Desktop\Projekte\boolie\gittut2\GitTutorial>git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Snowflake/GitTutorial.git
5655f58..412ffc4 master -> master
```

Jetzt wechsel wieder mit „cd“ in den originalen Ordner und lade dir die Dateien mit pull runter.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut2\GitTutorial>cd ../../

C:\Users\Marc\Desktop\Projekte\boolie>cd gittut

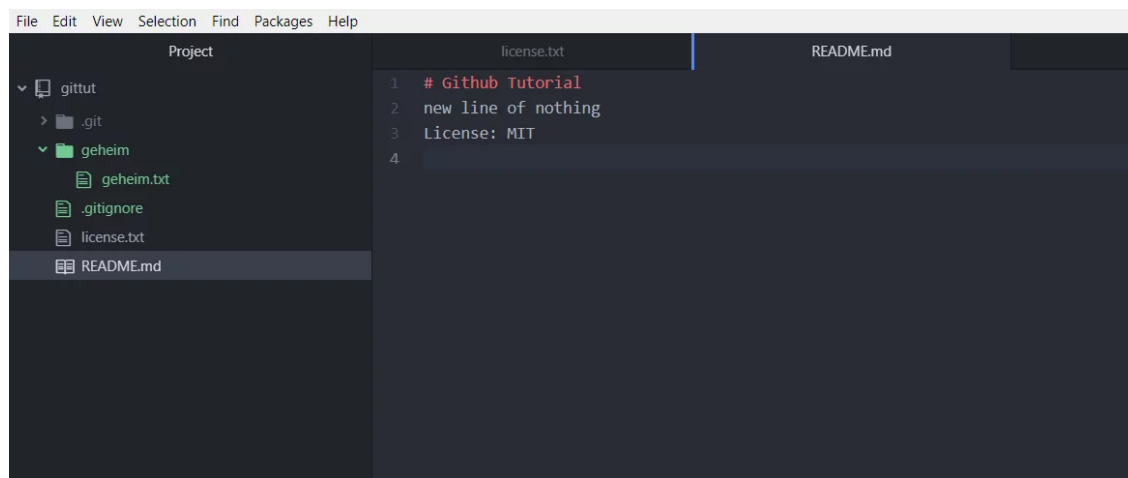
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/Snowflake/GitTutorial
 5655f58..412ffc4 master    -> origin/master
Updating 5655f58..412ffc4
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

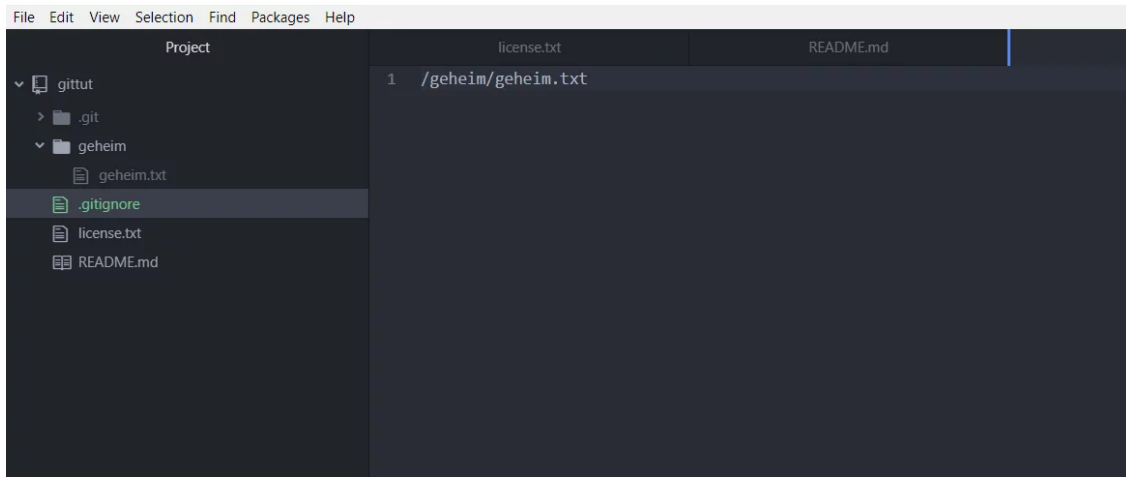
Fertig, deine beiden Ordner sind jetzt identisch. Wenn du jetzt 2 Computer hättest, könntest du über Git deine Daten immer synchronisieren.

## Dateien aus der Datenbank ausschließen

Um eine Datei dauerhaft aus GIT auszuschließen, musst du eine „.gitignore“ Datei anlegen und den Dateipfad dort eintragen. Öffne deinen Editor und lege 2 Dateien an. Zum einen die Datei „.gitignore“ und zum anderen eine Datei die du ausschließen möchtest.



In der „.gitignore“ Datei „/geheim/geheim.txt“ eintragen, um die Datei geheim.txt von der Datenbank auszuschließen.



**Mit \* kannst du einen ganzen Ordner ausschließen, schreibe z.B. „/geheim/\*“ um alle Dateien im Ordner geheim auszuschließen. Damit ist es auch möglich bestimmte Dateiendungen aus der Datenbank auszuschließen z.B. „\*.log“**

Wenn du jetzt „git status“ in deine Konsole eingibst, siehst du, dass die geheim Dateien nicht angezeigt werden und somit von Git ignoriert werden.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git add .

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   .gitignore

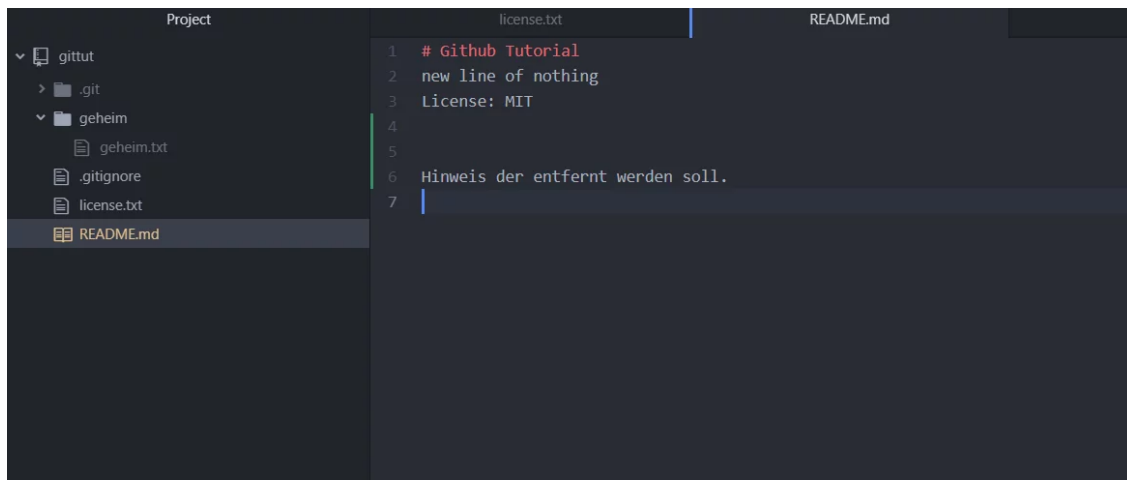
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git commit -m "ignoring secret Files"
[master 252281b] ignoring secret Files
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 337 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Snowflame/GitTutorial

.git
412ffc4..252281b  master -> master
```

## Datei Änderungen rückgängig machen

Mit dem „checkout“ Befehl lassen sich Dateien wieder auf den Ursprung zurück setzen. Ändere die Datei die Datei README.md, trage eine Textstelle ein.



Jetzt gibst du in der Konsole „git checkout README.md“ ein, danach wird die Änderung rückgängig gemacht und die Datei ist wieder auf dem Stand des letzten pulls.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git checkout README.md

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

**Wenn du eine Dateiänderung mit „checkout“ rückgängig machst, ist die Änderung weg, sei also vorsichtig!**

## Staged Markierung entfernen

Falls du schon eine Datei mit „git add“ markiert hast, aber diese doch nicht in den Commit soll, kannst du die Datei mit „git reset DATEI“ wieder aus dem staged Modus herausholen. Falls du die Datei bearbeitest und speicherst, wird die Datei automatisch aus dem staged genommen.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git add README.md

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
```

```

modified:   README.md

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git reset README.md
Unstaged changes after reset:
M       README.md

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

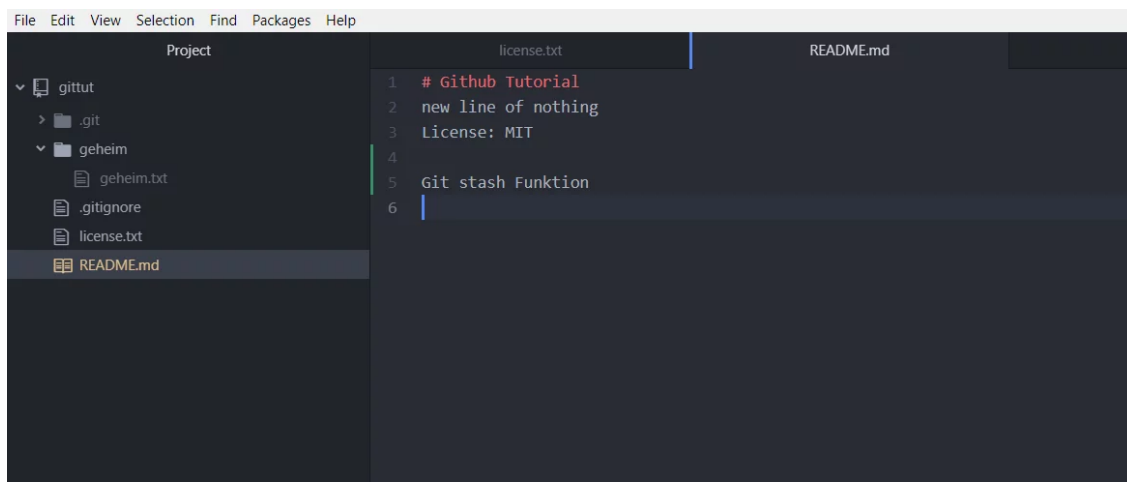
       modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

```

## Stash

Der Stash ist eine Art Zwischenablage, du kannst deine Änderungen in den Stash legen und z.B. ein pull machen. Nach dem pull machst du ein apply und deine Änderungen sind wieder da. Schreibe eine Zeile in die README.md.



Jetzt gibst du in der Konsole „git stash“ ein.

```

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

       modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

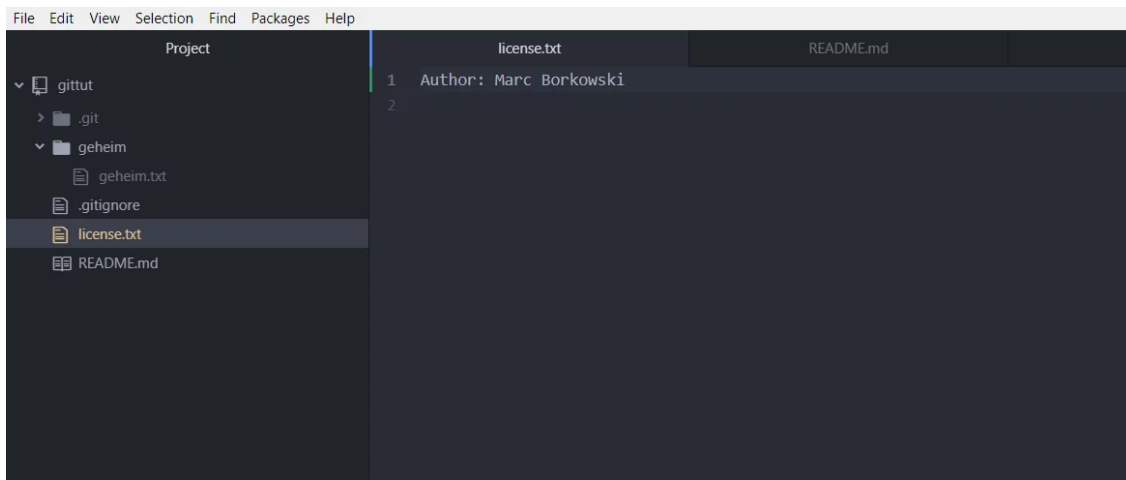
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git stash
Saved working directory and index state WIP on master: 252281b ignoreing secret Fi

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean

```

Bearbeite jetzt die license.txt, schreibe z.B. deinen Namen rein.

**Um es einfach zu halten, bearbeite nicht die README.md,**



Jetzt führe den Befehl „git stash apply“ in der Konsole aus.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   license.txt

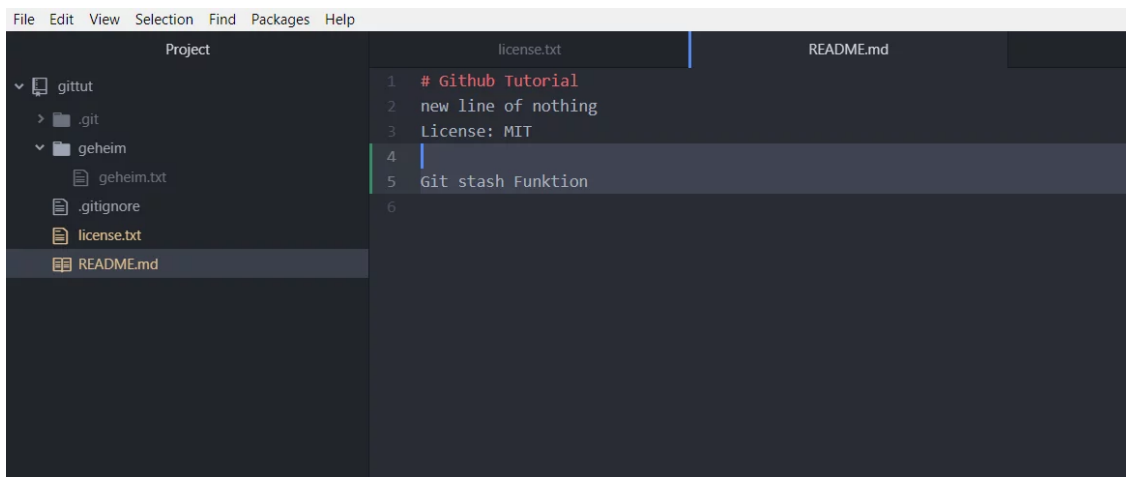
no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git stash apply
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md
        modified:   license.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

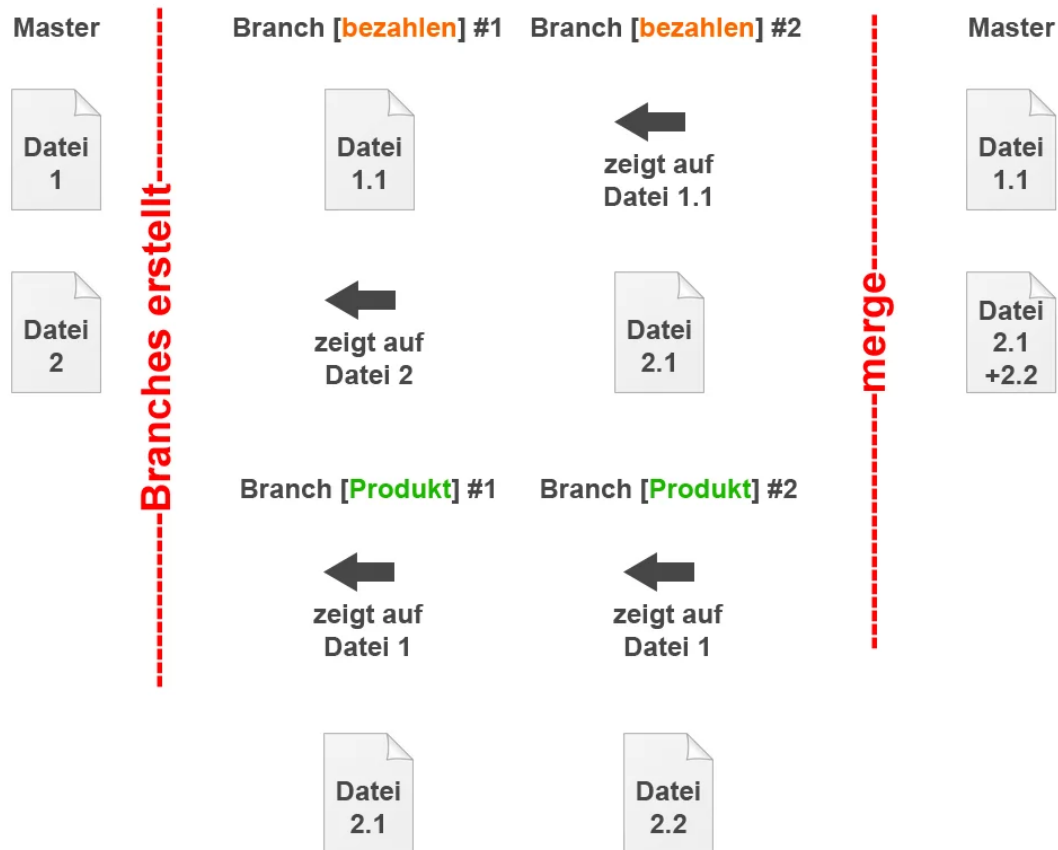
Jetzt ist die Änderung in der REAME.md wieder da!



# Branches

Ein Branch ist eine Sammlung von Commits, die wiederum eine neue Funktion in deiner App darstellen. Sprich jede neue Funktion in deiner App sollte eine eigene Branch besitzen. Wenn Entwickler 1 an der Branch „bezahl Funktion“ arbeitet und Entwickler 2 an „Produktauswahl“, kommen die beiden sich nicht in die Quere, da die beiden in unterschiedlichen Branches arbeiten. Später werden beide Branches dann merged und ergeben ein Paket aus neuen Funktionen für die App.

## Branches

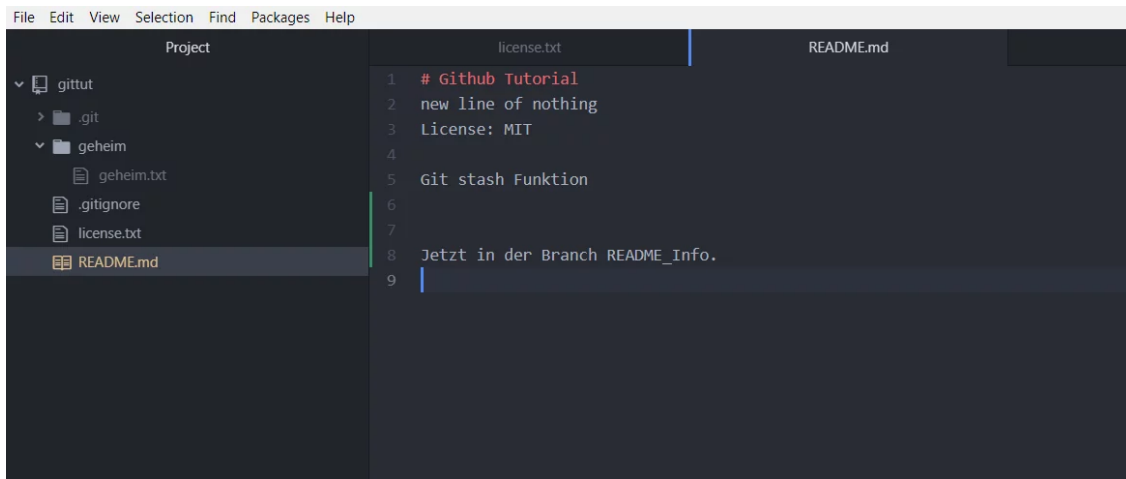


Um eine Branch zu erstellen musst du „checkout -b [NAME]“ in die Konsole eingeben. Mit dem Befehl erstellst du die Branch und wechselst gleich in die Branch. Mit dem Befehl „git branch“ kannst du dir alle Branches anzeigen lassen.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git checkout -b "README_Info"
Switched to a new branch 'README_Info'

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git branch
* README_Info
  master
```

Wenn du jetzt in der Branch README\_Info arbeitest, werden keine Änderungen im master vorgenommen. Ändere jetzt in der README.md wieder eine Zeile bzw. füge eine neue Zeile ein.



Schiebe jetzt deine Branch auf den Server. Aber diesmal musst du bei deinem push ein Upstream setzen. „git push –set-upstream origin README\_Info“ sagt dem Server eine neue Branch wurde erstellt und füge diese auf dem Server hinzu.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git status
On branch README_Info
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

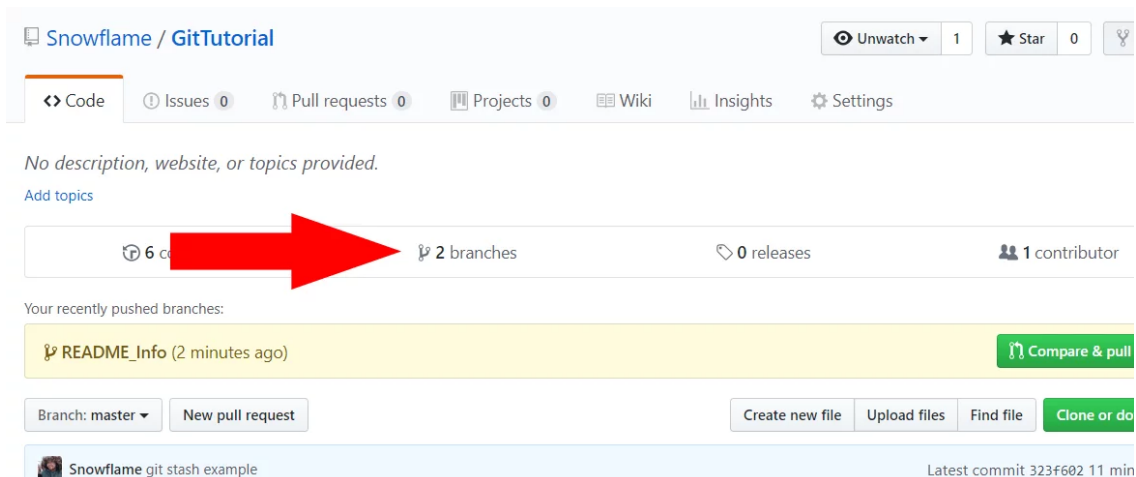
no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git add README.md

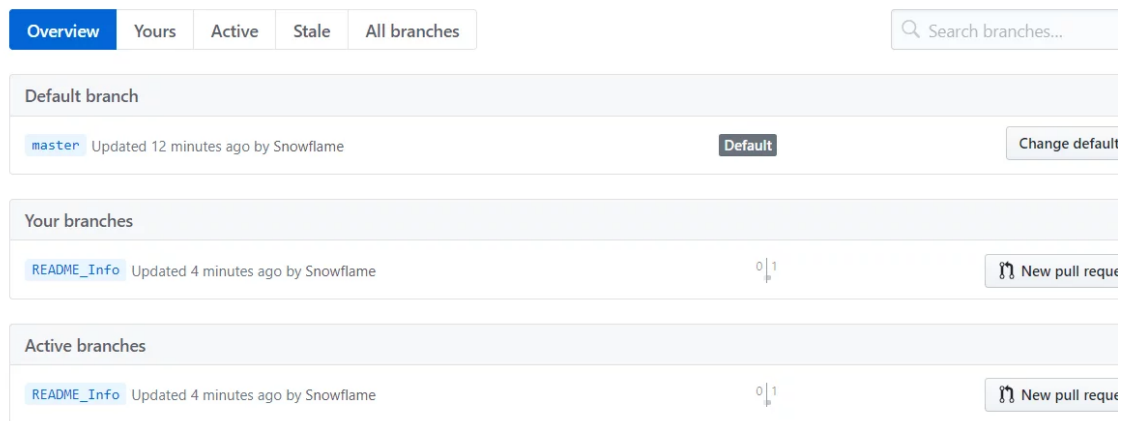
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git commit -m "new working Branch"
[README_Info 4ceb2aa] new working Branch
1 file changed, 3 insertions(+)

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git push --set-upstream origin README
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 416 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Snowflake/GitTutorial.git
 * [new branch]      README_Info -> README_Info
Branch README_Info set up to track remote branch README_Info from origin.
```

Jetzt zeigt GitHub auch die Branch an, öffne deine Branch und klicke auf „branches“.







Du kannst mit dem „git checkout [BRANCHNAME]“ Befehl zwischen den Branches wechseln. Wechsel jetzt zurück auf den Master Branch.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git branch
* README_Info
  master

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

C:\Users\Marc\Desktop\Projekte\boolie\gittut>git branch
  README_Info
* master
```

Jetzt siehst du in deinem Editor, dass die Änderungen weg sind. Aber die Änderungen sind nicht weg, die liegen nur in der anderen Branch.

## Branches mergen

Ein merge ist eine Zusammenfügung von 2 Branches, dabei wird der angegebene Branch in den aktuellen Branch eingefügt. Bei Git machst du das mit dem Befehl „git merge [BRANCH]“. Wechsle also in den Master Branch und merge README\_Info. Der Befehl sieht dann folgendermaßen aus: „git merge README\_Info“.

```
C:\Users\Marc\Desktop\Projekte\boolie\gittut>git merge README_info
Updating 323f602..4ceb2aa
Fast-forward
 README.md | 3 +++
 1 file changed, 3 insertions(+)
```

Fertig, die beiden Branches sind jetzt zusammen gesetzt worden und der master Branch hat den Code vom README\_Info mit drin.

## GitHub Pages erstellen

Webseite kostenlos in das Internet laden mit GitHub Pages



### Autor: **Marc Borkowski**

Moin,  
mein Name ist Marc Borkowski. Ich bin in Berlin geboren und habe mich in eine ruhigere Stadt verkrümelt, **Wismar**. Ich habe mehrere Jahre Erfahrung im Bereich **Frontend** und bin ausgebildeter Mediengestalter. Aktuell **entwickle ich mich zu einem Techlead** und werde hier meine Erfahrungen mit euch teilen.

## Kontakt

E-Mail: [moin@boolie.org](mailto:moin@boolie.org) | [Über mich Seite](#)



YouTube

## Schreibe einen Kommentar

Deine E-Mail-Adresse wird nicht veröffentlicht. Erforderliche Felder sind mit \* markiert  
Kommentar \*

Name \*

E-Mail \*

Website

Kommentar abschicken

Ich bin Kunde bei: