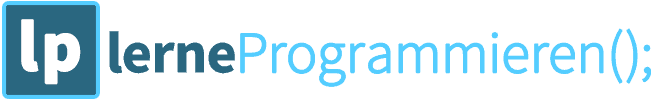
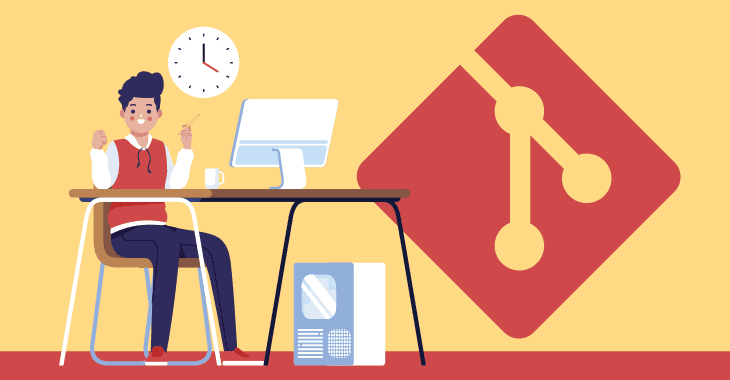
 https://lerneprogrammieren.de/git/

[](https://lerneprogrammieren.de/)

* [Starte HIER](https://lerneprogrammieren.de/start)
* [Produkte](https://kurse.lerneprogrammieren.de/produkte)
* [Blog](https://lerneprogrammieren.de/blog/)
* [Über uns](https://lerneprogrammieren.de/ueber-uns/)
* [Für Unternehmen](https://lerneprogrammieren.de/unternehmen/)
* [LOGIN](http://kurse.lerneprogrammieren.de/)

[8](https://lerneprogrammieren.de/git/#comments)



**Git Lernen in 30 Minuten – Anfänger Tutorial (2022)**

Die Benutzung von Git ist in den letzten Jahren schlagartig explodiert. Das beliebte Versionskontrollsystem wird von großen Open-Source-Projekten wie Linux, Entwickler-Teams, Solo-Entwicklern und sogar Studenten verwendet.

Anfänger tun sich im ersten Moment schwer, da Git viele neue und zum Teil kryptische Befehle mit sich bringt.

In diesem**umfänglichen Git Tutorial** wirst du das Versionskontrollsystem von grundauf lernen. Ein großer Vorteil von Git ist, dass du nicht alles nicht wissen, um damit anzufangen. Du kannst sofort durchstarten sobald du 2-3 Befehle kennst. Dein Wissen kannst du dann nach und nach aufbauen.

Du brauchst nicht die gesamte Anleitung an einem Tag durcharbeiten. Wenn du sofort in die Praxis einsteigen willst empfehle ich dir übrigens unseren Git Schnellstart-Kurs. Wenn du einfach nur einen Git Spickzettel mit allen Git-Befehlen haben möchtest, kannst du das LerneProgrammieren Git Cheatsheet kostenlos in unserem Mitgliederbereich herunterladen.

Los geht's!

**Was ist Git?**

Git ist eine Sammlung von Dienstprogrammen in der Kommandozeile, die Änderungen in Dateien verfolgen und aufzeichnen (meistens Quellcode, aber du kannst alle möglichen Dateien wie Textdateien und sogar Bild-Dateien "tracken".

Durch diese Funktionalität kannst du alte Versionen deines Projekts wiederherstellen, miteinander vergleichen, analysieren, Änderungen zusammenführen (mergen) und vieles mehr.

Dieser Prozess wird als **Versionskontrolle** bezeichnet. Es gibt eine Reihe von Versionskontrollsystemen, die diese Aufgabe übernehmen. Vielleicht hast du schon einige von ihnen gehört - Perforce, Mercurial, CVS, SVN und mehr.

**Git ist dezentralisiert**, das bedeutet, dass es nicht von einem zentralen Server abhängig ist, um alte Versionen deiner Dateien aufzubewahren. Stattdessen funktioniert es vollständig lokal, indem es diese Daten als Ordner auf deiner Festplatte speichert. Das nennen wir auch **Repository**.

Du kannst jedoch eine **Kopie deines Repositorys** online speichern. Dadurch wird es für Teams einfacher, zusammenzuarbeiten und am selben Code zu arbeiten. Meist werden dafür externe Git-Dienste wie [GitHub](https://github.com/) und [BitBucket](https://bitbucket.org/" \t "_blank) verwendet.

**1. Git installieren**

Um mit Git arbeiten zu können, musst du es zunächst auf deinem Computer installieren. Hier eine einfache Anleitung für die gängigsten Betriebssysteme:

* **Windows -** wir empfehlen [Git für Windows](https://git-scm.com/downloads" \t "_blank), da es sowohl einen GUI-Client als auch einen BASH-Kommandozeilenemulator bietet.
* **Linux** - Öffne einfach ein neues Terminal und installiere Git über den Paketmanager deiner Linux-Distribution. Für Ubuntu lautet der Befehl z.B. ***sudo apt-get install git***

Die Installation in WSL ist unbedingt erforderlich, da für die Datenbankapplikation mit einer virtual environment gearbeitet wird, die wir auf den pythonanywhere Server kopieren sollten. Die Windows „venv“ kann auf den Server nicht übernommen werden.

Als einheitliche Linux – Distribution verwenden wir Debian. Dazu ist eine Anpassung von WSL erforderlich:

# Aufgabenstellung –

# Installation von Debian und Festlegung der Standard – Distribution

Selbständig mithilfe des Manuals:   
<https://learn.microsoft.com/de-de/windows/wsl/basic-commands#install-a-specific-linux-distribution>

Dokumentation analog zum untenstehenden Beispielscreenshot:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 1: Beispielscreenshot

**B) Testen der .git Installation:**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung : Testen der git Installation auf Debian

* **Mac OS X** - Der einfachste Weg ist es, [homebrew](https://brew.sh/" \t "_blank) zu installieren und dann einfach brew install git von deinem Terminal aus zu starten.

Wenn du nicht auf der Kommandozeile mit Git arbeiten willst, kann ich dir Client-Programme wie GitHub Desktop und SourceTree (von Atlassian) emfpehlen. Es gibt jedoch noch viele weitere Git-Programme die kostenlos nutzbar sind. Auch wenn du dich für eine GUI-Anwendung entscheidest, solltest du dennoch die wichtigsten Git Befehle kennen.

In diesem Git Tutorial werden wir uns hauptsächlich auf die Kommandozeile konzentrieren, da du so die Git Befehle am einfachsten lernst (learning by doing).

**2. Git konfigurieren**

Nun, da wir git auf unserem Computer installiert haben, müssen wir noch einige Konfigurationen hinzufügen. Es gibt eine Menge Optionen, die dir zur Verfügung stehen (z.B. Aussehen und Funktionalität des Clients), aber wir werden hier nur die zwei wichtigsten Konfigurationen machen. Zum einen setzen wir unseren Benutzernamen und zum anderen unsere E-Mail.

**Öffne dafür das Programm "Git Bash" und führe diese Befehle aus:**



1

2

git config --global user.name "Daniel"

git config --global user.email daniel@meine-email.de

Jede Aktion, die wir in Git machen, wird jetzt einen Stempel dem soeben angegebenen Namen und der Email-Adresse haben. Auf diese Weise wissen andere Benutzer immer, wer was wann gemacht hat. Das bringt Ordnung ins Chaos, besonders bei großen Projekten an denen viele Entwickler arbeiten.

**3. Ein neues Repository erstellen - git init**

Wie wir bereits erwähnt haben, speichert git seine Dateien und seine Geschichte direkt als Ordner in deinem Projekt. Um ein neues Repository einzurichten, müssen wir ein Terminal öffnen, zu unserem Projektverzeichnis navigieren und git init ausführen. Dies wird Git für diesen speziellen Ordner aktivieren und ein verstecktes *.git-Verzeichnis* erstellen, in dem die Geschichte und Konfiguration des Repositorys gespeichert wird.

Erstelle auf deinem Desktop einen Ordner namens *"git-lernen"*, öffne ein neues Git-Terminal und gib Folgendes ein:



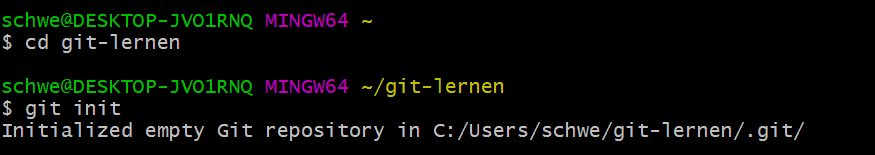
1

2

cd git-lernen

git init

**Die Kommandozeile sollte ungefähr folgendes Output liefern:**



Oder als Text:

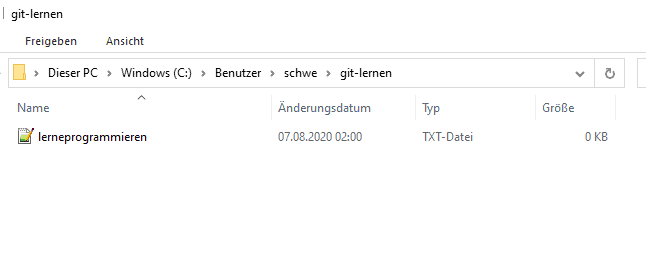


1

Initialized empty Git repository in C:/Users/schwe/git-lernen/.git/

Das bedeutet, dass unser Repository erfolgreich erstellt wurde. Natürlich ist es derzeit noch leer. Erstelle daher eine einfache Textdatei namens *lerneprogrammieren.txt* und speichere sie im Ordner *"git lernen"*.

**Das sollte dann ungefähr so aussehen:**



*Textdatei in Ordner "git-lernen" mit dem Namen "lerneprogrammieren.txt" erstellen*

**4. Status des Repositorys überprüfen - git status**

*Git status* ist ein weiterer wichtiger Befehl, den du kennen musst. Er gibt Informationen über den aktuellen Status des Repositories zurück.

* Ist alles auf dem neuesten Stand?
* Was ist neu?
* Was hat sich geändert?
* usw.

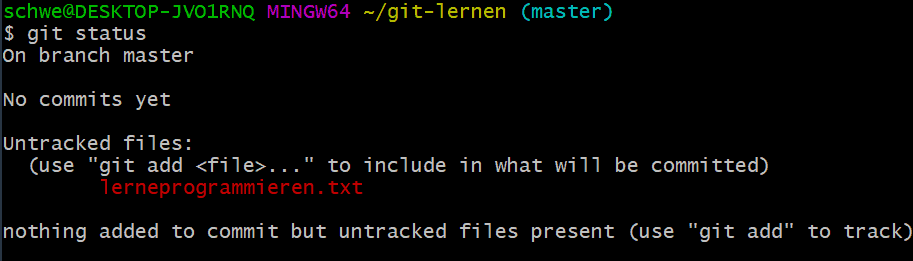
Führe den Status-Befehl wie folgt aus:



1

git status

Das Ausführen von *git status* in unserem neu erstellten Repository sollte ungefähr Folgendes zurückgeben:



**Oder als Text-Output:**



1

2

3

4

5

6

7

8

9

On branch master

No commits yet

Untracked files:

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

lerneprogrammieren.txt

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

    to track)

Die zurückgegebene Nachricht besagt, dass *lerneprogrammieren.txt*nicht verfolgt (genauergesagt nicht getracked) wird. Das bedeutet, dass die Datei neu ist und Git noch nicht weiß, ob es die Änderungen an der Datei verfolgen oder sie einfach ignorieren soll.

Um die neue Datei zu bestätigen, müssen wir sie "stagen".

**5. Staging - git add**

Git kennt das Konzept einer *"Staging Area*". Du kannst dir das wie eine leere Leinwand vorstellen, die die Änderungen enthält, die du gerne committen (festschreiben) möchtest. Am Anfang ist deine Staging-Area (oder Staging-Bereich) leer, aber du kannst mit dem Befehl ***git add*** Dateien hinzufügen (oder sogar einzelne Zeilen und Teile von Dateien) und schließlich mit ***git commit*** alles committen (einen Snapshot erstellen).

In unserem Fall haben wir nur eine Datei. Lass uns diese hinzufügen:



1

git add lerneprogrammieren.txt

Wenn du alles in dem Verzeichnis hinzufügen möchtest, benutzt du den Befehl wie folgt:



1

git add -a

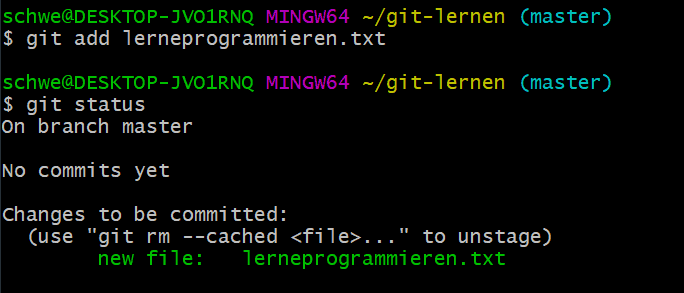
Wenn du den obigen Befehl ausgeführt hast, prüfe erneut den Status mit *git status*. Du solltest dann folgendes angezeigt bekommen:



1

git status

**Output:**



Oder als Text:



1

2

3

4

5

6

7

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: lerneprogrammieren.txt

Unsere Datei ist jetzt im Staging-Bereich. Die Statusmeldung sagt uns auch, was sich an den Dateien im Staging-Bereich geändert hat - in diesem Fall ist eine *neue Datei hinzugekommen*. Weitere Statusmeldungen können sich auch auf die Änderung oder Löschung von Dateien beziehen.

**6. Commit erstellen - git commit**

Ein Commit repräsentiert den Zustand unseres Repositorys zu einem bestimmten Zeitpunkt. Es ist wie ein Schnappschuss, zu dem wir zurückgehen können. Damit können wir sehen, wie das Repository "damals" aussah, als wir den Schnappschuss (Snapshot) gemacht haben.

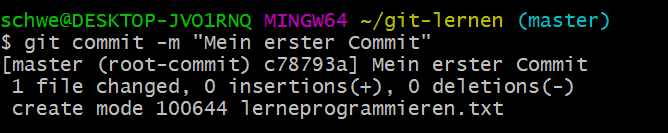
Um einen neuen Commit zu erstellen, müssen wir mindestens eine Änderung im Staging-Bereich hinzugefügt haben (wir haben das gerade mit *git add* gemacht). Erstelle jetzt wie folgt deinen ersten Commit:



1

git commit -m "Mein erster Commit"

**Output:**



Oder als Text:



1

2

3

4

5

[master (root-commit) c78793a] Mein erster Commit

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 lerneprogrammieren.txt

Dies wird einen neuen Commit mit allen Änderungen aus dem Stagingbereich erstellen (Hinzufügen der lerneprogrammieren.txt). Die Option *-m* mit dem Text *"Mein erster Commit"* ist eine benutzerdefinierte Beschreibung, die die Änderungen des Commits zusammenfasst.

Es wird als gute Praxis angesehen, relativ oft zu committen und möglichst spezifische Commit-Nachrichten zu schreiben.

**7. Remote-Repository**

Im Moment ist unser Commit lokal - er existiert nur im *.git* Ordner. Obwohl ein lokales Repository an sich schon nützlich ist, werden wir in den meisten Fällen unsere Arbeit mit anderen Entwicklern teilen und sie auf einem Server oder auf einem Repository-Hosting (wie z.B. Github oder Bitbucket) bereitstellen wollen.

**7.1 Verbindung zu einem Remote-Repository aufbauen - git remote add**

Um etwas in ein Remote Repo hochzuladen, müssen wir zuerst eine Verbindung mit ihm herstellen.

Für die Zwecke dieses Tutorials wird die Adresse unseres Repositorys *https://github.com/lerneprog/tutorial* verwendet. Wir empfehlen dir, dein eigenes leeres Repository bei [GitHub](https://github.com/), [BitBucket](https://bitbucket.org/" \t "_blank) oder einem anderen Dienst zu erstellen. Die Registrierung und Einrichtung des Repos kann eine Weile dauern, aber alle Dienste bieten gute Schritt-für-Schritt-Anleitungen, um dir bei der Erstellung zu helfen.

Um unser lokales Repository mit dem auf GitHub zu verlinken, führen wir die folgende Zeile im Terminal aus:

*(Hinweis: Bitte ersetze die URL mit deiner eigenen Remote-Repository URL)*



1

git remote add origin https://github.com/lerneprog/tutorial.git

Ein Projekt kann mehrere Remote-Repositories zur gleichen Zeit haben. Um sie unterscheiden zu können, geben wir ihnen verschiedene Namen. In der Praxis wird das Haupt-Remote-Repository meist mit dem Wort *"origin"* (Ursprung) bezeichnet.

**7.2 Änderungen auf den Remote-Server übertragen - git push**

Jetzt ist es an der Zeit, unsere lokalen Commits auf den Server zu übertragen. Dieser Prozess wird **Push** genannt und wird jedes Mal durchgeführt, wenn wir das Remote-Repository aktualisieren wollen.

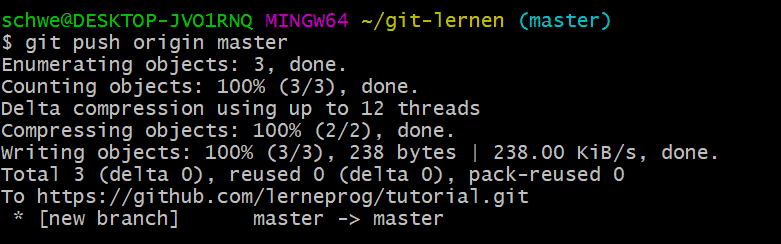
Der Git-Befehl dafür ist ***git push****.* Er benötigt **zwei Parameter** - 1) den Namen des Remote-Repos (wir haben unser Repo *"origin"*genannt) und 2) den *Branch*, auf den gepusht werden soll*(master* ist der Standard-Branch für jedes Repo).



1

git push origin master

Output:



Oder als Text:



1

2

3

4

5

6

7

8

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Delta compression using up to 12 threads

Compressing objects: 100% (2/2), done.

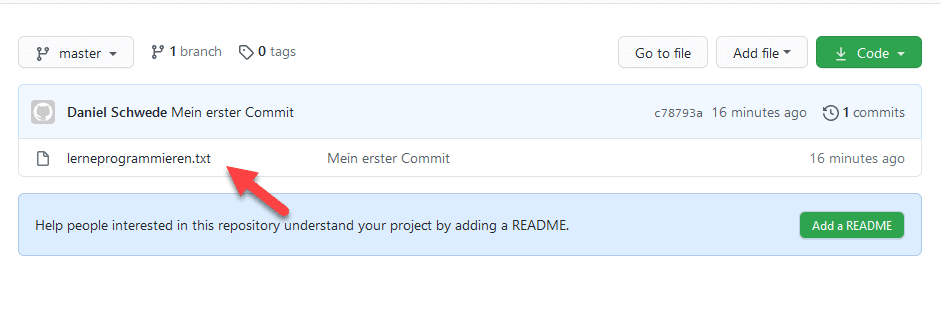
Writing objects: 100% (3/3), 238 bytes | 238.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

To https://github.com/lerneprog/tutorial.git

\* [new branch] master -> master

Abhängig vom Remote-Dienst, den du benutzt, musst du dich vorher einloggen, damit der Push durchgeführt werden kann. Wenn alles richtig gemacht wurde, sollte die *lerneprogrammieren.txt* dort verfügbar sein, wenn du in deinem Webbrowser das zuvor erstellte Remote-Repository aufrufst. In unserem Fall können wir den "Push" unter dieser URL prüfen: <https://github.com/lerneprog/tutorial>



Unter personal settings muss ein Access-token erzeugt werden….

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

*Unser erster Push in das GitHub Remote-Repository*

Anschließend kann der push – Befehl mit Angabe des users und des access – token durchgeführt werden. Die Kontrolle der Dateien erfolgt zusätzlich im github – Webaccount.

Das Test – Repository soll öffentlich zugänglich sein, damit eine Kontrolle der durchgeführten Schritte möglich ist…

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ergebnis auf github:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**7.3 Änderungen vom Remote-Server abrufen - git pull**

Wenn du Updates in deinem Repository machst, können andere Entwickler deine Änderungen mit einem einzigen Befehl herunterladen - ***git pull***.

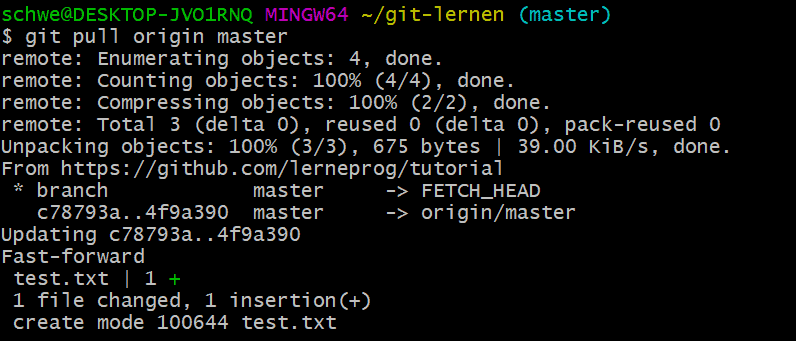
*(Ich habe im folgenden Beispiel die Datei "test.txt" angelegt. Dieser werden wir jetzt vom Remote-Repository "pullen")*



1

git pull origin master

**Output:**



Oder als Text:



1

2

3

4

5

6

7

8

9

10

11

12

13

remote: Enumerating objects: 4, done.

remote: Counting objects: 100% (4/4), done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 675 bytes | 39.00 KiB/s, done.

From https://github.com/lerneprog/tutorial

\* branch master -> FETCH\_HEAD

c78793a..4f9a390 master -> origin/master

Updating c78793a..4f9a390

Fast-forward

test.txt | 1 +

1 file changed, 1 insertion(+)

create mode 100644 test.txt

**8. Ein Repository klonen - git clone**

An diesem Punkt können Team-Mitglieder dein Remote-Repository auf Github sehen und durchsuchen. Sie können es lokal herunterladen und mit dem Befehl *git clone* eine voll funktionsfähige Kopie deines Projekts erstellen:



1

git clone https://github.com/lerneprog/tutorial.git

Ein neues lokales Respository wird automatisch erstellt, wobei die Github-Version als Remote konfiguriert ist.

**9. Branches**

Wenn man ein neues Feature entwickelt, wird es als gute Praxis angesehen, an einer **Kopie des Originalprojekts** zu arbeiten, die als *Branch* bezeichnet wird. Branches haben ihre eigene Historie und isolieren ihre Änderungen voneinander, bis man sich entscheidet, sie wieder zusammenzuführen. Dies geschieht aus verschiedenen Gründen:

* Eine bereits funktionierende, stabile Version des Codes wird nicht von ungewünschten Fehlern beeinträchtigt
* Viele Funktionen können sicher von mehreren Entwicklern gleichzeitig entwickelt werden
* Die Entwickler können an ihrem eigenen Branch arbeiten, ohne das Risiko, dass sich ihre Codebasis durch die Arbeit eines anderen Entwicklers ändert.
* Wenn man sich nicht sicher ist, was das Beste ist, können mehrere Versionen desselben Features auf verschiedenen Branches entwickelt und dann miteinander verglichen werden.

**9.1 Branches erstellen - git branch**

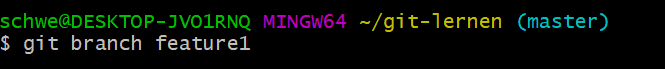
Der Standard-Branch jedes Repositorys heißt *master*. Um zusätzliche Branches zu erstellen, benutze den Befehl ***git branch <name>***:



1

git branch feature1

Durch den obigen Befehl erstellst du den Branch "feature1", der (noch) dem "master"-Branch gleicht.



*(Hinweis: Weitere Meldungen werden dir nach dem erfolgreichen Anlegen des Branches nicht angezeigt)*

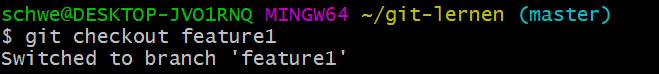
**9.2 Branches wechseln - git checkout**

Wir sind derzeit noch im "master"-Branch. Doch wir wollen jetzt in unseren neu erstellen Branch "feature1" wechseln. Dies geschieht mit dem Befehl *git checkout*, wobei ein Parameter erwartet wird, nämlich der Branch-Name, in den wir wechseln wollen.



1

git checkout feature1

****

**9.3 Branches zusammenführen - git merge**

Unser neues Feature (feature1) wird nur eine weitere Textdatei namens *button.txt* sein. Wir werden sie erstellen, hinzufügen und committen:



1

2

git add button.txt

git commit -m "Neues Feature fertig"

Das neue Feature ist komplett, wir können zurück zum Master-Branch wechseln:



1

git checkout master

Oh nein! Wo ist plötzlich unsere "button.txt" hin? Sie ist spurlos aus dem Dateisystem verschwunden.

***Keine Panik.***

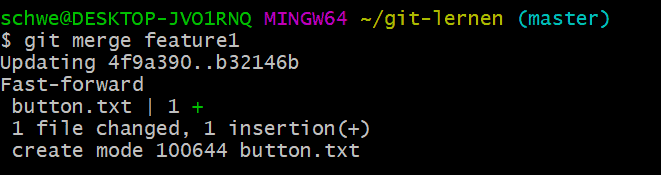
Das liegt daran, dass wir wieder im Masterbranch sind, und hier wurde *die feature.txt* nie erstellt. Um sie im Master-Branch hinzuzufügen, müssen wir beide Zweige zusammenführen (mergen) und die Änderungen, die in *"feature1"*gemacht wurden, auf die Hauptversion des Projekts anwenden. Hört sich kompliziert an, ist es aber nicht. Du brauchst wieder nur eine Zeile, um den Merge-Prozess durchzuführen:



1

git merge feature1

**Output:**



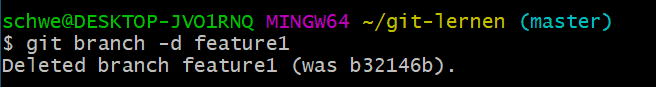
Der Master-Branch ist jetzt auf dem neuesten Stand. Der "feature1"-Branch wird nicht länger benötigt und kann entfernt werden.



1

git branch -d feature1

**Output:**



**Git für Fortgeschrittene**

Im ersten Teil des Git-Tutorials haben wir uns mit den Grundlagen beschäftigt. In diesem Teil werden wir einen Blick auf einige fortgeschrittenere Funktionen von Git werfen.

**10. Unterschied zwischen Commits prüfen**

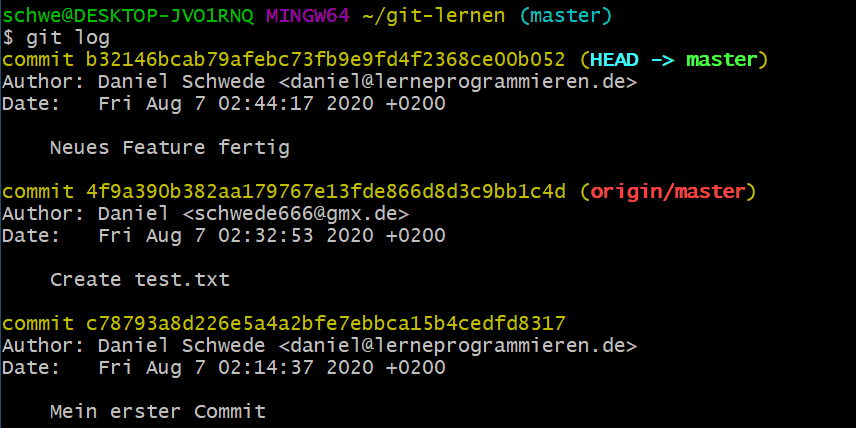
Jeder Commit hat eine eindeutige ID in Form einer Reihe von Zahlen und Symbolen (SHA). Um eine Liste aller Commits und ihrer IDs zu sehen, können wir ***git log*** benutzen:



1

git log

**Output:**



Oder als Text:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

commit b32146bcab79afebc73fb9e9fd4f2368ce00b052 (HEAD -> master)

Author: Daniel Schwede <daniel@lerneprogrammieren.de>

Date: Fri Aug 7 02:44:17 2020 +0200

Neues Feature fertig

commit 4f9a390b382aa179767e13fde866d8d3c9bb1c4d (origin/master)

Author: Daniel <daniel@lerneprogrammieren.de>

Date: Fri Aug 7 02:32:53 2020 +0200

Create test.txt

commit c78793a8d226e5a4a2bfe7ebbca15b4cedfd8317

Author: Daniel Schwede <daniel@lerneprogrammieren.de>

Date: Fri Aug 7 02:14:37 2020 +0200

Mein erster Commit

Wie du siehst, sind die **ID's sehr lang**, aber du brauchst sie nicht abtippen. Du kannst sie mit dem Cursor markieren, um sie in die Zwischenablage zu kopieren. Alternativ reichen auch die ersten paar Zeichen, damit Git den richtigen Commit zuordnet.

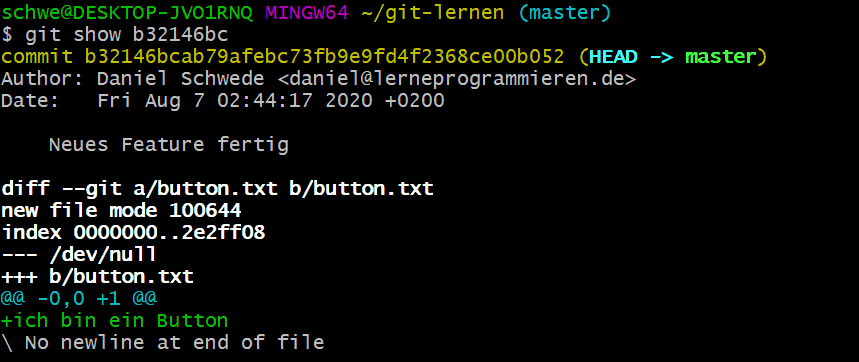
Um zu sehen, was sich in einem Commit befand, können wir den Befehl *git show <Commit ID>* benutzen:



1

git show b32146bc

**Output:**



Oder als Text:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

commit b32146bcab79afebc73fb9e9fd4f2368ce00b052 (HEAD -> master)

Author: Daniel Schwede <daniel@lerneprogrammieren.de>

Date: Fri Aug 7 02:44:17 2020 +0200

Neues Feature fertig

diff --git a/button.txt b/button.txt

new file mode 100644

index 0000000..2e2ff08

--- /dev/null

+++ b/button.txt

@@ -0,0 +1 @@

+ich bin ein Button

\ No newline at end of file

Im obigen Output bekommen wir eine Menge Informationen zu dem Commit angezeigt. Du bekommst Angaben zum Autor, einen Zeitstempel, die Commit-Nachricht sowie allerlei Datei-Informationen, wie bspw. die hinzugefügten Zeilen innerhalb der Textdatei (*"ich bin ein Button"*) angezeigt.

**11.Eine Datei zu einer früheren Version rückgängig machen**

Git erlaubt es uns, jede ausgewählte Datei wieder so herzustellen, wie sie in einem bestimmten Commit war. Dies geschieht über den bekannten Git-Checkout-Befehl, den wir früher zum Wechseln der Zweige benutzt haben, der aber auch dazu benutzt werden kann, zwischen den Commits zu wechseln (es ist in Git ziemlich üblich, dass ein Befehl für mehrere scheinbar nicht zusammenhängende Aufgaben benutzt wird).

Im folgenden Beispiel nehmen wir die Textdatei *lerneprogrammieren.txt*und machen alles rückgängig, was wir seit dem ersten Commit damit gemacht haben. Dazu müssen wir die id des Commits angeben, zu dem wir zurückgehen wollen, sowie den vollständigen Pfad zu unserer Datei. Die jeweilige ID bekommst du über *git log*. Führe danach folgenden Befehl aus:



1

git checkout b32146bcsab79a lerneprogrammieren.txt

**12. Einen Commit rückgängig machen - git revert**

Wenn du bemerkst, dass du in deiner Commit-Nachricht einen Tippfehler gemacht hast oder du vergessen hast eine Datei hinzuzufügen, kannst du den Fehler mit g***it commit --amend*** beheben.

Dadurch wird alles vom letzten Commit wieder in den Staging-Bereich hinzugefügt. Es wird außerdem versucht, eine neue Commit zu machen. Das gibt dir die Möglichkeit, deine Commit-Nachricht auszubessern oder weitere Dateien zum Staging-Bereich hinzuzufügen.

Für komplexere Fixes, die sich nicht im letzten Commit befinden (oder wenn du deine Änderungen bereits gepusht hast), musst du *git revert* benutzen. Dies wird alle Änderungen, die ein Commit gemacht hat, rückgängig machen und einen neuen Commit erstellen.

Auf den neuesten Commit kann über den Alias *HEAD* zugegriffen werden.



1

git revert HEAD

Für andere Commits ist es am besten, eine ID zu verwenden. Die ID des Commits findest du über den *git log* Befehl. Benutze danach folgenden *git revert* Befehl:



1

git revert <Commit ID>

Wenn du ältere Commits rückgängig machst, bedenke, dass Fusionskonflikte sehr wahrscheinlich auftreten können. Dies passiert, wenn eine Datei durch einen anderen neueren Commit geändert wurde und Git nun nicht die richtigen Zeilen zum Zurücksetzen finden kann, da sie nicht mehr vorhanden sind.

**13. Dateien ignorieren mit .gitignore**

In den meisten Projekten gibt es Dateien oder ganze Ordner, die wir niemals committen wollen. Wir können sicherstellen, dass sie nicht versehentlich in unserem git add -A enthalten sind, indem wir eine *.gitignore-Datei* erstellen:

1. Erstelle manuell eine Textdatei namens .gitignore und speichere sie im Verzeichnis deines Projekts.
2. Listet darin die Namen der Dateien/Verzeichnisse auf, die ignoriert werden sollen, jedes in einer neuen Zeile.
3. Die .gitignore selbst muss hinzugefügt, committed und gepusht werden, da sie genau wie jede andere Datei im Projekt ist.

Gute Beispiele für zu ignorierende Dateien sind:

* log-Dateien
* task-Runner-Builds
* der node\_modules Ordner in node.js Projekten
* ordner, die von IDEs wie Netbeans und IntelliJ
* persönliche Anmerkungen der Entwickler

Ein .gitignore, der alle oben genannten Dinge verbietet, wird so aussehen:

* \*.log
* build/
* node\_module/
* .idea/
* meineKommentare.txt

Der Schrägstrich am Ende einiger Zeilen signalisiert, dass es sich um einen Ordner handelt und wir alles darin *rekursiv ignorieren*. Das Sternchen dient seiner üblichen Funktion als Platzhalter (Wildcard).

**Fazit zum "Git Lernen"-Tutorial**

Wenn du es bis zu dieser Stelle im Tutorial geschafft hast, dann Respekt! Solltest du die einzelnen Abschnitte alle durchgearbeitet haben, hast du Git gelernt. Du kannst dieses Tutorial jederzeit als Referenz benutzen, wenn dir ein Git Befehl nicht einfällt oder du ein einfaches Nachschlagewerk benötigst.

Ebenso kann ich dir unseren kostenlosen **Git-Spickzettel** empfehlen. Dort sind alle Git-Befehle in Kürze aufgelistet. Du findest ihn im LerneProgrammieren Mitgliederbereich.

Ich habe in diesem Ratgeber versucht die wichtigsten Informationen rund um das Versionskontrollsystem zu vermitteln. Git ist kann ziemlich komplex wirken und hat eine Menge mehr Features und Tricks zu bieten.

An dieser Stelle hast du bereits 90% der wichtigsten Funktionen und Befehle zu Git gelernt.

**Kostenloses Git-Cheatsheet (PDF)**

Die wichtigsten Git-Befehle auf einen Blick



[**Cheatsheet Download**](https://lerneprogrammieren.de/git/)

*(für immer kostenlos!)*