Hochladen eines bestehenden Git-Repository auf Github

* 5. März 2022

**Meine Projekte verwalte ich mit dem Versionierungssystem Git. Damit habe ich die Kontrolle über den Projektverlauf und ein doppeltes Netz, wenn etwas schiefgeht. Der Online-Dienst Github hat sich als de facto Standard für die Verwaltung von kleinen und grossen Software-Projekten etabliert. Auch ich verwalte meinen Code mit Github. In diesem Post beschreibe ich, wie du ein lokales Git-Repository auf Github hochlädst.**

Das Versionierungssystem Git ist ideal, um die Kontrolle über den Versionsverlauf von Projekten zu haben und sorglos neue Features auszuprobieren. Für den Fall, dass beim Programmieren etwas schiefgeht, kann ich jederzeit auf eine frühere Version zurückkehren.

Für die zentrale Verwaltung meiner Git-Projekte benutze ich Github. Damit kann ich von überall her auf die Projekte zugreifen und mit anderen zusammen daran arbeiten. Oft lade ich die Projekte von hier direkt auf den Produktions-Web-Server.

Bevor ich ein Projekt zentral mit Github verwalte, setze ich das Git-Repository meist Lokal auf und pushe es erst auf den Server, wenn das Projekt einen gewissen Stand erreicht hat. Wie das geht, beschreibe ich hier.

Vorbereitungen auf dem lokalen Computer

Obwohl ich hier beschreibe, wie ich ein bestehendes Git-Repo auf Github hochlade, gehe ich für dieses Tutorial davon aus, dass noch kein lokales Repository besteht. Deshalb erstellen wir als Erstes ein neues Projekt namens **neues\_projekt** im Verzeichnis **Dokumente**.

Navigiere mit der Befehlszeile in das Verzeichnis **Dokumente**:

cd ~/Dokumente

Erstelle ein Verzeichnis namens **neues\_repo**:

mkdir neues\_repo

Wechsle in dieses Verzeichnis:

cd neues\_repo

Bevor wir beginnen das Repo für Git vorzubereiten, erstellen wir am besten ein paar Beispieldateien. Es macht wenig Spass, mit einem leeren Verzeichnis zu arbeiten.

touch hello\_world.py readme.md test.log

Damit haben wir drei neue Dateien in unserem Verzeichnis:

**hello\_world.py**

**readme.md**

**test.log**

Initialisieren des Git-Repository

Um Git für ein Projekt verwenden zu können, müssen wir zuerst das Repository initialisieren, das heisst, die zu versionierenden Dateien dem Versionsverlauf hinzufügen. Das machen wir mit folgendem Befehl:

git init

Damit wird ein neues, leeres, Repository erstellt. Im Ordner liegt nun ein unsichtbarer Ordner mit dem Namen **.git**. Dateien und Verzeichnisse, die mit einem Punkt beginnen, werden normalerweise nicht angezeigt. D.H. sie sind nicht zu sehen, wenn wir Dateien mit dem **-ls** Befehl anzeigen lassen oder wenn wir mit einem Dateimanager auf die Verzeichnisse zugreifen.

Mit dem Befehl **ls -a** kannst du dir auch unsichtbare Dateien und Verzeichnisse anzeigen.

ls -a

Du solltest folgende Ausgabe erhalten:

**. .. .git hello\_world.py readme.md test.log**

Eine .gitignore-Datei anlegen

Bevor wir Projektdateien zum Versionierungsverlauf von Git hinzufügen, empfiehlt es sich, bestimmte Dateien vom Versionierungsverlauf auszuschliessen. Dazu gehören Python **.pyc** Dateien, grosse Mediendaten wie Bilder und Videos (das sind Binärdaten, die nicht von Git versioniert werden können, sogenannte Blobs oder Binary Large Objects), Log-Dateien, etc. Um dies zu erreichen, legen wir eine .**gitignore**-Datei an. In dieser sind Dateien und Ordner aufgelistet, welche du nicht versioniert haben möchtest. Von [Github](https://github.com/github/gitignore/blob/main/Python.gitignore) gibt es eine gute Vorlage für Python-Projekte. Diese kann als Startpunkt dienen für eine eigene personalisierte **.gitignore**-Datei.

Um die **.gitignore**-Datei anzulegen, erstelle eine neue leere Datei mit folgendem Befehl:

touch .gitignore

Damit wird eine leere, unsichtbare, Datei mit dem Namen **.gitignore** angelegt.

Öffne die Datei und schreibe die gewünschten Ausnahmen rein. Das kann in einer rudimentären Form zum Beispiel so aussehen:

# Byte-compiled / optimized / DLL files

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

# Django stuff:

\*.log

local\_settings.py

db.sqlite3

db.sqlite3-journal

# Environments

.env

.venv

env/

venv/

ENV/

env.bak/

venv.bak/

Dateien zu Git hinzufügen

Um die bestehenden Dateien zum Git-Repository hinzuzufügen und mit der Versionskontrolle zu starten, gehe ich in folgender Reihenfolge vor. Zuerst versichere ich mich, ob ich wirklich alle nicht benötigten Dateien ausgeschlossen habe. Um zu sehen, welche Dateien von Git in die Versionierung aufgenommen werden, gib folgenden Befehl ein:

git status

Damit zeigt dir Git eine Auflistung aller Dateien, die noch nicht von der Versionierung getrackt werden in Rot an. Wenn du Dateien siehst, die du nicht haben möchtest, füge diese der **.gitignore**-Datei hinzu.

In unserem Beispiel sollten alle Dateien, inklusive der .**gitignore-Datei**, aber **ohne** die Datei **test.log** angezeigt werden:

**Auf Branch master**

**Noch keine Commits**

**Unversionierte Dateien:**

**(benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)**

**.gitignore**

**hello\_world.py**

**readme.md**

Bist du zufrieden, führe folgenden Befehl aus um die Dateien deines Projektes in die Staging-Area aufzunehmen und für den ersten Commit vorzubereiten:

git add --all

Auch hier kannst du nochmals mit **git status** nachkontrollieren, ob alles deinen Wünschen entspricht. Die drei Dateien werden jetzt mit grüner Farbe dargestellt Das sieht so aus:

**Auf Branch master**

**Noch keine Commits**

**Zum Commit vorgemerkte Änderungen:**

**(benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)**

**neue Datei: .gitignore**

**neue Datei: hello\_world.py**

**neue Datei: readme.md**

Bist zu zufrieden, mache deinen ersten Commit:

git commit -m "Initial commit"

Die Ausgabe dazu sieht wie folgt aus:

**[master (Root-Commit) 3ed76d8] Initial commit**

**3 files changed, 19 insertions(+)**

**create mode 100644 .gitignore**

**create mode 100644 hello\_world.py**

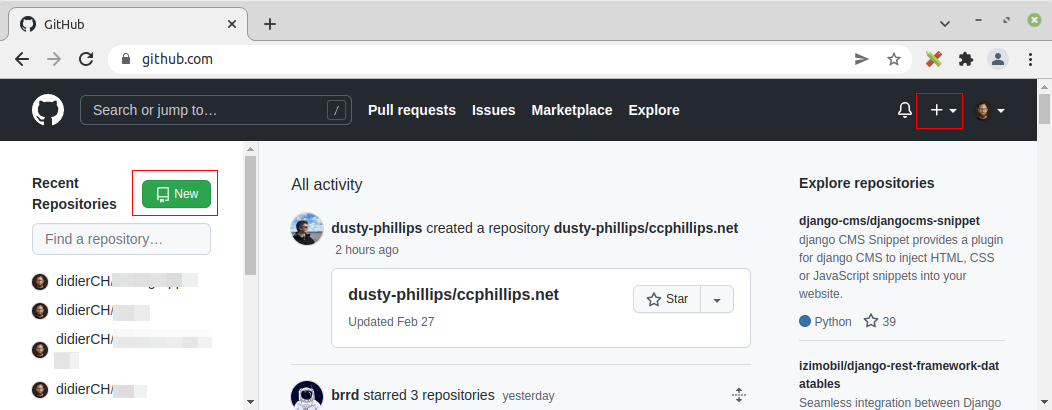
**create mode 100644 readme.md**

Anstelle von **Initial commit** kannst du auch etwas anderes hinschreiben. Wichtig ist, dass es dir später hilft, den Überblick zu behalten. Die Nachricht sollte kurz und aussagekräftig sein.

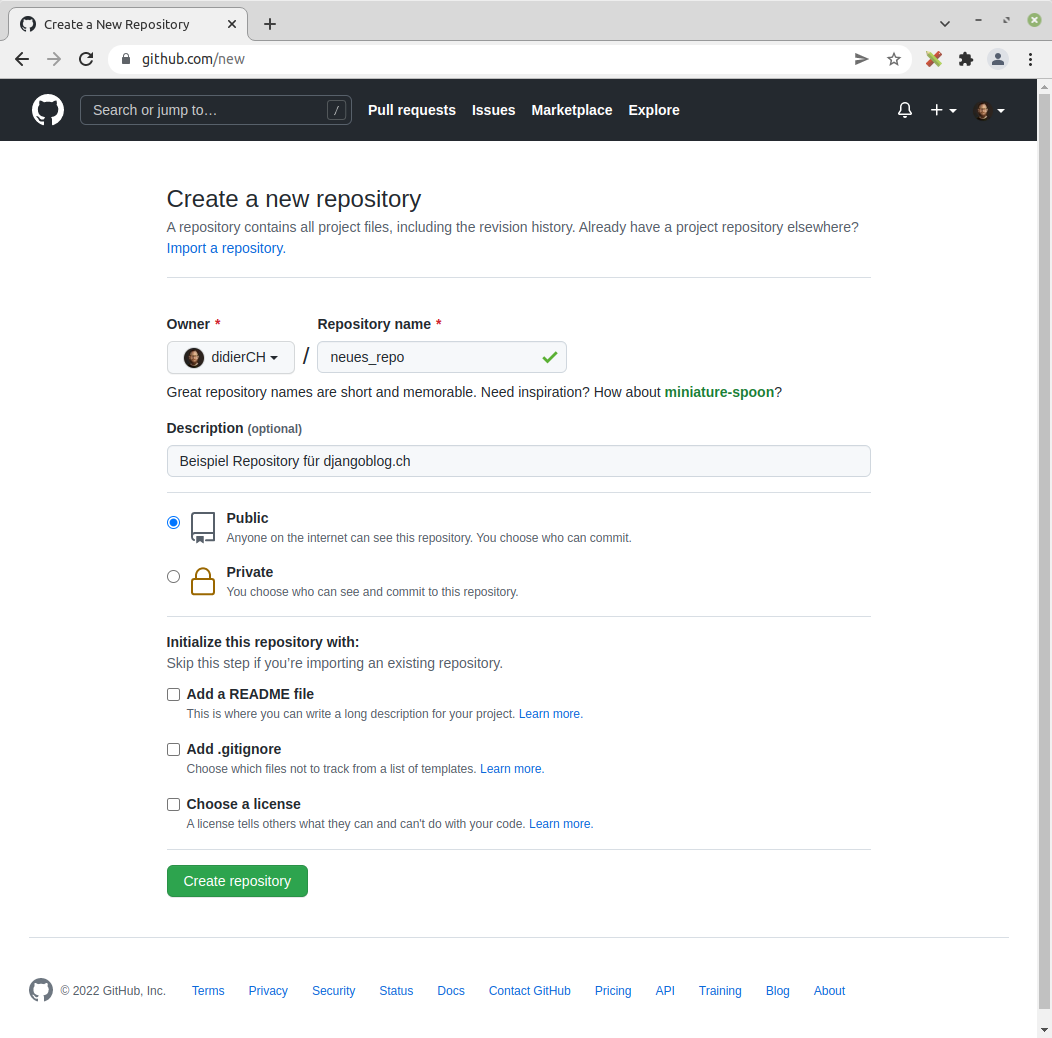
Repository auf Github vorbereiten

Als nächsten Schritt erstellen wir ein leeres Git-Repository auf Github, welches uns im weiteren Projektverlauf als zentraler Hub dient. Achte darauf, dass du keine Dateien darin erstellt, wie z.B. ein Readme oder eine Lizenzdatei. Diese kannst du später hinzufügen. Wenn das Repository leer ist, gelingt das Hochladen deines lokalen Repository am einfachsten.

Melde dich in deinem Github-Account an und klicke links oben auf den grünen Button **New** oder rechts oben neben deinem Profilbild auf das Pluszeichen und dann auf **New repository**.



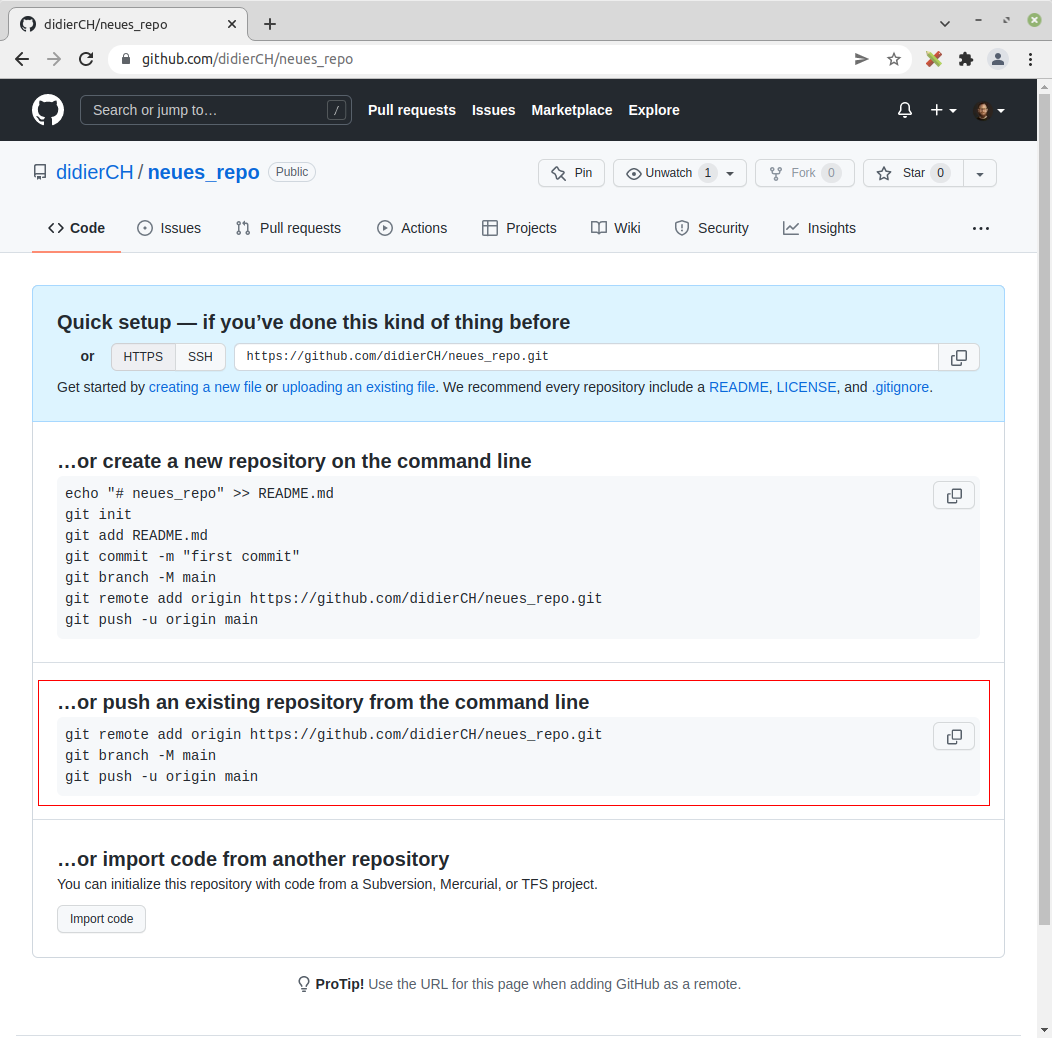
Gib deinem neuen Repository einen aussagekräftigen Namen und Beschreibung. In unserem Fall ist das **neues\_repo**, gleich wie das lokale Verzeichnis.



Entscheide ob es privat oder öffentlich sein soll. Lasse die Checkboxen bei den drei unten stehenden Fragen leer.

* Add a README file (**nein**)
* Add .gitignore (**nein**)
* Choose a license (**nein**)

Im nächsten Fenster schlägt dir Github einige Optionen vor. Da wir bereits alles vorbereitet haben, kopieren wir uns den Code aus der Sektion **...or push an existing repository from the command line**.



Lokales Repository auf Github hochladen

Wechsle wieder zu deinem lokalen Repository (welches den gleichen Namen wie das Github-Repo haben sollte, und gib folgenden Befehl in die Befehlszeile ein:

git remote add origin https://github.com/DEIN\_BENUTZERNAME/neues\_repo.git

Damit setzt du das Github-Repository als «zentrales» Repository. Gegebenenfalls musst du dich mit deinem Benutzernamen und Passwort identifizieren.

**Hinweis:** Zentrales Repository ist im Zusammenhang mit Git vielleicht das falsche Wort, da es sich bei Git um ein «verteiltes» Versionierungssystem handelt ohne «zentralen» Server (wie zum Beispiel bei SVN, einer anderen Software für die Code-Versionierung). Dennoch nenne ich es hier das «zentrale» Repository, da es im weiteren Verlauf des Projektes als Verwaltungszentrale dient.

Führe folgende zwei Befehle aus, um die Daten aus dem lokalen Repo in das zentrale Verzeichnis auf Github hochzuladen:

git branch -M main

git push -u origin main

Mit dem Befehl **git branch -M main** wird der Hauptzweig zu **main** umbenannt. Bis vor Kurzem war der Name für den Hauptzweig in Github **master**. Github hat aus Gründen der kulturellen Sensibilität beschlossen, den Haupt-Branch von **master** zu **main** zu wechseln.

Mit dem Befehle **push** werden die Daten dann von deiner Festplatte auf den Server geschoben. Das wird mit folgender Ausgabe quittiert:

**Objekte aufzählen: 4, fertig.**

**Zähle Objekte: 100% (4/4), fertig.**

**Delta-Kompression verwendet bis zu 8 Threads.**

**Komprimiere Objekte: 100% (3/3), fertig.**

**Schreibe Objekte: 100% (4/4), 436 Bytes | 436.00 KiB/s, fertig.**

**Gesamt 4 (Delta 0), Wiederverwendet 0 (Delta 0)**

**To https://github.com/didierCH/neues\_repo.git**

**\* [new branch] main -> main**

**Branch 'main' folgt nun Remote-Branch 'main' von 'origin'.**

Das wär's. Damit wird dein Git-Repository von Github verwaltet und das lokale Git-Repository ist ein Klon des Github-Repository. Wenn es sich bei deiner Arbeit um ein Open-Source-Projekt handelt, kannst du damit beginnen, dein Projekt bekannt zu machen und andere Entwickler zu finden, die gemeinsam mit dir daran arbeiten. Und auch wenn du alleine am Projekt arbeitest, kannst du von allen Vorteilen von Github profitieren.

Fehler:

# [Gelöst] Fehler: src refspec main stimmt nicht überein

* 23. März 2022

Wenn Sie zum ersten Mal versuchen, in ein gitRepository zu pushen, erhalten Sie möglicherweise die folgende Fehlermeldung:

error: src refspec main does not match any

error: failed to push some refs to

Codesprache: Bash ( bash )

Dies liegt daran, dass Sie Ihre Dateien nicht übertragen haben!

## So beheben Sie diesen Fehler

git commit -m "initial commit"

git push origin main

Codesprache: Bash ( bash )

Der obige Code wird Ihre Staging-Dateien festschreiben, bevor sie in den gewünschten Zweig verschoben werden.

Standardmäßig schieben wir unseren Code in den mainZweig oben. Tauschen Sie dies bei Bedarf gegen einen anderen Zweig aus.