Verwendung von Git und GitHub in RStudio

Jakob Adler

10.01.2023

Einleitung und Vorarbeit

Dieses Dokument soll eine kurze Einführung in die Arbeit mit Git und GitHub in Verbindung mit RStudio sein.

Um alle 3 Tools in einem sinnvollen Workflow miteinander zu verbinden, ist ein wenig Vorarbeit nötig. Folgende Voraussetzungen müssen erfüllt sein:

1. R muss installiert sein.
2. RStudio muss installiert sein.
3. Ein GitHub-Account muss erstellt werden.
4. Git muss installiert sein.

Als ausführliche Anleitung ist Happy Git with R zu empfehlen.

Kurze Erläuterung zum Workflow

Sinn der Einrichtung der verschiedenen Tools ist die Möglichkeit einer einfachen Versionskontrolle der eigenen Projekte direkt aus RStudio heraus. Hierzu werden die Projekte als lokale Repositories innerhalb der lokalen Git-Instanz (auf dem eigenen Computer) gespeichert. Über die Verknüpfung mit dem GitHub-Account können die lokalen Git-Repositories dann in die Cloud (den GitHub Remote Server) geladen werden, um die Informationen zentral zu speichern. Arbeiten mehrere Entwickler an einem Projekt, kann man so in der Cloud immer den aktuellen Arbeitsstand sichern und regelmäßig die Fortschritte der anderen Entwickler mit seiner lokalen Kopie des Projektes abgleichen.

Folgende Workflows sind damit möglich:

1. Projekt in RStudio coden, dieses lokal mittels Git versionskontrolliert abspeichern, das lokale Git Repository nach GitHub in die Cloud laden.
2. Änderungen lokal vornehmen, lokales Repository updaten, aktuelle Version auf GitHub hochladen.
3. Auf GitHub Repository ändern, auf GitHub gespeichertes Repository herunterladen und damit das lokale Repository updaten.
4. Bereits auf GitHub erstellte Repositories herunterladen und lokal mit Git verwalten.

Auf die Einzelheiten der verschiedenen Workflows wird später weiter eingegangen.

Git als Versionskontrollsoftware in RStudio festlegen

Wenn alle Komponenten installiert sind (R, RStudio und Git) dann muss zuerst Git als Versionskontrollsoftware in RStudio festgelegt werden. Dies geht über folgenden Weg:

> Tools - Global Options - Git/SVN - Ordner in der die git.exe liegt als Git executable einstellen

Nun weiß RStudio, dass wir Git als Versionskontrollsoftware verwenden wollen.

Die Git-Bash als Default-Terminal einstellen

Auch wenn RStudio eine recht komfortable Möglichkeit des Umgangs mit Git bereitstellt, wird zur Einrichtung ein Terminal/eine Konsole benötigt, um die wichtigsten Befehle zur ersten Einrichtung von Git auszuführen. In Windows selbst gibt es mehrere Terminals/Konsolen (die klassische Konsole, PowerShell, usw.). Um die grundlegenden Befehle für Git auszuführen, benutzt man die sogenannte **Git Bash** als Konsole. RStudio gibt uns die Möglichkeit, direkt in der RStudio Oberfläche ein Terminal/eine Konsole zu öffnen. Da in RStudio immer von **Terminal** die Rede ist, wird nachfolgend immer von **Terminal** gesprochen.

Um die **Git Bash** als Standard-Terminal in RStudio einzustellen, nutzen wir folgenden Weg:

Tools - Global Options - Terminal - New terminals open with: **Git Bash** auswählen

Nun startet RStudio immer wenn wir ein neues Terminal öffnen die **Git Bash**.

Git initial einrichten

Nachdem wir RStudio gesagt haben, dass wir Git als Versionskontrollsoftware benutzen und bei öffnen eines neuen Terminals in der Git Bash Kommandos ausführen wollen, müssen wir Git initial konfigurieren. Hierzu müssen wir uns ein neues Git-Bash-Terminal öffnen:

Tools - Terminal - New Terminal

Nun öffnet sich im unteren Teil unseres RStudio Bildschirmes neben der R-Console ein neues Terminal. Gegebenenfalls kann bei Windows die Benutzerkontensteuerung eine aktive Zustimmung verlangen. Dies ist eine Sicherheitsmaßnahme, damit keine Terminals, die z.B. Schadcode ausführen könnten, ohne unsere Zustimmung geöffnet werden. Hier muss man natürlich zustimmen, da RStudio sonst keine Git Bash öffnen kann.

Zur initialen Konfiguration tippen wir im Terminal (also der Git Bash in RStudio):

```
git config --global user.name 'GitHub-Username'
git config --global user.email 'Mailadresse bei GitHub'
```

Bitte beachtet genau die Schreibweise mit *Leerzeichen* und *Bindestrichen*. Mit *'GitHub-Username'* ist unser selbst gewählter GitHub-Username gemeint, den wir hier **in einfachen Hochkommata** einsetzen. Dementsprechend ist in der zweiten Code-Zeile mit *'Maildresse bei GitHub'* die Mailadresse gemeint, mit der wir uns bei GitHub registriert haben. Diese Mailadresse muss ebenfalls **in einfachen Hochkommata** angegeben werden. Zum Abschluss tippen wir zur Überprüfung:

```
git config --global --list
```

Nun wird euch beides noch einmal angezeigt und ihr könnt überprüfen, ob euer GitHub-Username und eure Email-Adresse, die ihr zur Registrierung bei GitHub nutzt, richtig hinterlegt sind. Wenn die Mailadresse nicht stimmt, ist später eine Verbindung eures lokalen Git-Profils mit eurem GitHub-Profil nicht möglich.

Es gibt auch die Möglichkeit, diese initiale Konfiguration mittels eines R-Packages zu machen. Das Package heißt **"usethis"**. Nutzen könnt ihr das Package in der R Console wie folgt:

```
install.packages("usethis")
library(usethis)
use_git_config(user.name = "GitHub Username", user-email = "Mailadresse bei GitHub")
```

Grundlegende Nomenklatur einstellen

Ursprünglich wird der erste **Branch** eines Projektes als **Masterbranch** bzw. kurz als **"master"** bezeichnet. In der Entwicklercommunity hat es sich aber durchgesetzt, dieses **Masterbranch** als **"main"** zu bezeichnen. Deswegen bietet es sich an, die Nomenklatur in Git anzupassen:

```
git config --global init.defaultBranch main
```

Nun wird bei Initialisierung eines neues Repositories der **Hauptbranch** immer als **"main"** benannt.

Optional: Git Client

Viele Entwickler empfehlen, einen Git Client zu installieren. Ein solcher Git-Client ist eine gute Möglichkeit um seine Projekte lokal zu verwalten, den Überblick zu behalten und die Historie eines Projektes anschaulich zu visualisieren. Ein möglicher Git-Client, der in seiner Basic-Version kostenlos ist, ist Gitkraken.

Wenn man möchte, kann man sich **Gitkraken** kostenlos herunterladen, installieren und dann seine Projekte damit ordnen. Für den hier beschriebenen Workflow ist dies aber nicht notwendig. Sobald man allerdings kollaborativ mit mehreren Entwicklern an einem Projekt arbeitet (wofür ja auch eine Versionskontrollsoftware wie Git erschaffen wurde), ist es definitiv zu empfehlen, sich einen solchen Git-Client anzuschaffen.

Verbindung vom lokalen Git mit GitHub

Nun wollen wir RStudio mit unserem lokalen Git und unserem GitHub-Profil verbinden. Warum ist diese Verbindung wichtig? In unserem Workflow geht es darum, lokal auf dem eigenen Computer z.B. in RStudio an einem Projekt zu arbeiten, dieses versionskontrolliert abzuspeichern und zusätzlich zentral in der Cloud zu speichern. An großen Projekten wird mitunter jahrelang gearbeitet, es wirken viele Entwickler mit und die Entwickler wechseln mitunter. Es macht also Sinn, den Fortschritt der gemeinsamen Arbeit zentral auf einem Server (in unserem Fall auf einem GitHub-Server) zu speichern. Alle Entwickler können dann lokal an Ihrem Teil des Projektes weiterarbeiten, den Forschritt in ihrem lokalen Respository speichern und nach abgeschlossener Arbeit den aktuellen Stand auf GitHub hochladen. Ein anderer Entwickler kann sich dann bevor er seine tägliche Arbeit beginnt, den aktuellen Stand von GitHub herunterladen und ist somit immer auf dem aktuellen Stand des Projektes.

Um die Verbindung zwischen Git und GitHub herzustellen muss man sich eindeutig identifizieren können, sodass Git im Hintergrund immer weiß, wer welche Änderung am Code vorgenommen hat. Jede Aktion die man macht, wird mit den eigenen sogenannten **Credentials** versehen. Somit kann die zentrale Git-Version, die auf den GitHub-Servern läuft, jede Aktion mitschreiben und wir sehen in der Historie unseres Projektes wer wann etwas geändert, hinzugefügt oder gelöscht hat.

Diese **Credentials** können wir uns bei GitHub als **Personal Access Token (PAT)** erstellen. Prinzipiell kann die Kommunikation zwischen unserem lokalen Git und GitHub über zwei Protokolle abgewickelt werden: **SSH** und **HTTPS**. Ich empfehle die Kommunikation über **HTTPS**, denn dafür ist der **Personal Access Token** gedacht.

Um unseren **PAT** zu erstellen, müssen wir uns bei GitHub einloggen. Dann folgt:

Settings - ganz unten "**Developer Settings" - personal access token - Tokens (classic)**

Hier könnt ihr euch nun einen neuen **PAT** erstellen. Ihr sollten den Token **kurz benennen** um zu wissen, wofür er gedacht ist und könnt einen **Ablaufzeitraum** einstellen. Hierbei geht es um die Sicherheit. Für kleine eigene Projekte oder Projekte die nicht zu einem zukünftig zu vermaktenden Produkt gehören, könnt ihr den Token ohne "Verfallsdatum" anlegen. Ansonsten empfiehlt es sich den Token regelmäßig ablaufen zu lassen, damit eventuelle gestohlene Token, um sich Zugang zum Projekt zu verschaffen, ihre Gültigkeit verlieren.

Diesen gerade erstellten PAT solltet ihr wie ein Passwort behandeln und euch irgendwo abspeichern/merken/etc.! Er wird nur einmal bei GitHub angezeigt und dann nie wieder!

Nun müssen wir unsere **Credentials** anlegen. Dies geschieht in der R Console wie folgt:

```
install.packages("gitcreds")
library(gitcreds)
gitcreds::gitcreds_set()
```

Nach Ausführen dieser Funktion, werdet ihr nun gebeten euren PAT einzugeben. Kopiert ihn hierzu in die R Console und bestätigt die Eingabe. Nun sind eure **Credentials** angelegt und wir können starten. Falls euer PAT irgendwann abläuft, wird euch GitHub ein paar Tage vor dem Ablauf warnen. Ihr könnt den Token dann erneuern oder einen neuen Token erstellen. Wenn der Token erneuert bzw. ein neuer Token erstellt ist,

müsst ihr die Credentials wieder neu angeben. Dies geschiehrt auf dem gleichen Weg wie das initiale Anlegen der Credentials (gitcreds::gitcreds_set()).

Workflows

Neues Projekt beginnen

Um ein neues Projekt zu beginnen und als lokales Git Repository abzuspeichern empfehle ich folgenden Workflow:

1. File - New Project - New Directory - Je nach Art des Projektes: New project/R package/ etc. - Namen eingeben - erstellen.
2. Nun könnt ihr coden (R-Skript erstellen, R-Markdown erstellen, usw.).
3. Tools - Project Options - Git/SVN - als Version Control System Git auswählen - New Repository - Yes - R startet neu - euer neues Repository ist angelegt.

Um bestehende Skripte in ein Projekt umzuwandeln was ihr mittels Git dann versionskontrolliert speichert, empfehle ich folgendes Vorgehen:

1. File - New Project - Existing Directory - Namen eingeben - erstellen.
2. Nun könnt ihr coden (R-Skript erstellen, R-Markdown erstellen, usw.).
3. Tools - Project Options - Git/SVN - als Version Control System Git auswählen - New Repository - Yes - R startet neu - euer neues Repository ist angelegt.

Nach diesem Workflow habt ihr ein neues Projekt oder ein bereits bestehendes Projekt lokal auf eurem Computer als neues Git Repository angelegt und könnt nun alle Änderungen versionskontrolliert speichern.

Neu angelegtes Repository auf GitHub hochladen

Wenn ihr euer neues/bestehendes Projekt, was nun lokal mit Git verwaltet wird, zentral auf GitHub speichern wollt, empfehle ich folgendes Vorgehen:

1. Öffnet das Projekt in RStudio.
2. Reiter Git (oben rechts neben dem Environment) - bei allen angezeigten Dateien bei "Staged" den Haken setzen (alle Änderungen sind jetzt für das committen (das hinzufügen zu eurem Main-Branch) bereit) - auf **Commit** klicken.
3. Es öffnet sich das **Commit-Fenster**. Hier könnt ihr eure **Commit-Message** eingeben, die möglichst kurz und genau beschreiben sollte, was ihr an der Datei geändert habt (ihr solltet nach zwei Jahren daraus erkennen können, was ihr mit diesem Commit getan habt).
4. Auf "close" drücken und das Commit-Fenster schließen.

Nun sind alle Änderungen in eurem lokalen Git-Repository gespeichert. Um dieses Repository nun auf GitHub hochzuladen hilft uns eine R-Funktion aus dem "usethis"-Package:

```
usethis::use_github()
```

Wenn diese Funktion ausgeführt wird, wird das Repository nun als neues Repository auf GitHub angelegt. Am besten habt ihr euch vorher bei GitHub bereits eingeloggt, denn dann wird euch sofort das neue Repository auf GitHub angezeigt. Ihr könnt nun z.B. direkt auf GitHub eine **README.md** für euer Repository anlegen, um es auf GitHub zu präsentieren.

Pull und Push

Pull Wenn ihr die **README.md** nun direkt auf GitHub angelegt habt, wollt ihr diese README.md natürlich auch in eurem lokalen Repository haben, um den gleichen Stand eures Projektes zentral in der Cloud und lokal auf eurem Computer zu haben. Hierzu könnt ihr die Daten von GitHub wieder abrufen und euer lokales Repository updaten. Dies geht so:

1. Eurer Projekt in RStudio öffnen.
2. Oben rechts den Reiter Git anwählen.

3. Auf **Pull** klicken.

Nun werden alle Daten von GitHub heruntergeladen und Git prüft im Hintergrund was neu ist und z.B. die online erstellte README.md wird heruntergeladen und euer z.B. online angepasstes R-Skript wird direkt lokal auf den neuesten Stand gebracht.

Push Habt ihr nun das README.md noch einmal lokal auf eurem Computer angepasst, so könnt ihr es wieder zu GitHub hochladen. Dies geht so:

1. Im geänderten Projekt bei RStudio sein (je nach Session habt ihr vielleicht auch gerade ein anderes oder gar kein Projekt geöffnet).
2. Oben rechts den Reiter Git anwählen.
3. Auf **Push** klicken.

Nun werden alle lokal vorgenommenen Änderungen auf GitHub hochgeladen und die Git-Instanz die auf den GitHub-Servern läuft, aktualisiert die auf GitHub gespeicherten Dateien.

Auf GitHub bestehendes Repository ins lokale Git herunterladen

Um bereits auf GitHub bestehende Repositories herunterzuladen um lokal daran zu arbeiten und die Änderungen auch lokal mittels Git versionskontrolliert zu speichern empfiehlt sich folgender Workflow:

1. File - New Project - Version Control - Git - Nun möchte RStudio die URL zu eurem Repository wissen.
2. Auf GitHub einloggen und ins gewünschte Repository gehen, dort unter "**Code**" und "**local**" die **HTTPS-URL** kopieren.
3. Die kopierte **HTTPS-URL** bei RStudio einfügen.
4. Projekt erstellen.

Nun habt ihr das bestehende GitHub-Repository auf euren Computer heruntergeladen und mittels Git versionskontrolliert gespeichert. Ihr könnt nun an dem Projekt arbeiten und mit **Push** Änderungen auf GitHub hochladen und Änderungen anderer Entwickler mittels **Pull** von GitHub in euer lokales Repository herunterladen.

Abschluss

Nun habt ihr RStudio erfolgreich mit Git und GitHub verbunden und könnt versionskontrolliert und durch eine zentrale Cloud-Version eures Codes abgesichert ggf. auch mit anderen Entwicklern an eurem Projekt arbeiten.

Falls Probleme auftreten, empfehle ich gezielt bei Happy Git with R nachzulesen.

Viel Spaß beim Coden!