
Git und GitHub - Minimaleinführung

Robert Heise - Education4Industry GmbH



Zuletzt aktualisiert: 2022-05-17

Inhaltsverzeichnis

Inhaltsverzeichnis	i
1 Überblick	1
1.1 Was ist Git?	1
1.2 Was ist GitHub?	1
1.3 Git in der Projektphase	1
1.4 Weiterführende Links	2
2 Einrichten eines Repositorys mit GitHub	3
2.1 Überblick der einzelnen Arbeitsschritte	3
2.2 Erstellen eines Repositorys für das Team in GitHub	4
2.2.1 Anlegen der Accounts für GitHub	4
2.2.2 Erstellen eines Repositorys mit GitHub	4
2.2.3 Freigabe des Repositorys für alle Teammitglieder	7
2.3 Erzeugen und Einrichten der SSH-KEYs für den Zugriff auf GitHub	8
2.3.1 Erzeugen eines ssh-keys auf dem Raspberry Pi	8
2.3.2 Registrierung des ssh-keys in GitHub	9
2.3.3 Erzeugen eines ssh-keys für Windows	11
2.4 Klonen des Repositorys mittels SSH	11
2.5 Konfiguration des Repositorys unter Git	13
3 Anwendung der Software Git	14
3.1 Auswahl grundlegender Git-Kommandos	14
3.2 Demoaufgabe	15
3.3 Auswahl weitere Kommandos	18

1 Überblick

1.1 Was ist Git?

Git ist das am weitesten verbreitete Werkzeug zur Versionsverwaltung in der Softwareentwicklung. Es wurde 2005 vom Linus Torvald entwickelt. Das Softwareprojekt wird dabei in einem zentralen Repository (kurz Repo) verwaltet. Ein Repository kann als Quellcode-Datenbank verstanden werden. Alle beteiligten Entwickler laden jedoch eine lokale Arbeitskopie auf ihren jeweiligen Rechner. Vorgenommen Änderungen können in das zentrale Repository aufgenommen und auf diese Weise ausgetauscht werden. Git erlaubt das Erstellen verschiedener Entwicklungszweige (branches), welche nebeneinander existieren und, sofern notwendig, wieder zusammengeführt werden können. Die Steuerung erfolge über Git-eigene Befehle. Zahlreiche IDEs (integrierte Entwicklungsumgebungen) bieten jedoch die Möglichkeit der direkten Integration Gits.

Auf dem Raspberry Pi ist Software Git bereits installiert. Die Installation Gits auf einem anderen Rechner ist Rahmen der Projektphase 1 nicht notwendig.

1.2 Was ist GitHub?

GitHub ist ein webbasierte Dienst zur Versionsverwaltung für Softwareprojekte. Dabei wird das zentrale Git-Repository auf den Servern des Dienstes zur Verfügung gestellt. Das Unternehmen GitHub hat seinen Sitz in San Francisco, USA. Alternative Anbieter sind z.B. GitLab und Bitbucket.

1.3 Git in der Projektphase

Für die Zwecke der Projektphase **kann** die Versionierung mittel Git geschehen. Dabei dient es der Zusammenarbeit ein gemeinsames zentrales Repository auf z.B. GitHub anzulegen. Im Weiteren wird die Einrichtung eines Repositories mit Git in einzelnen Schritten beschrieben und ein Überblick über wenige grundlegende Git-Kommandos gegeben. Für die Zwecke der Projektphase soll sich auf wenige grundlegende Basisfunktionen Gits beschränkt werden. Der Fokus dieser

Anleitung ist die Herstellung einer Arbeitsfähigkeit mit Git für die Projektphase herzustellen, dh. mittels GitHub ein Repository anzulegen, dieses gemeinsam im Team/Gruppen zu nutzen und so erste Erfahrungen mit Git zu sammeln.

1.4 Weiterführende Links

Im WWW finden Sie zahlreiche Tutorials und Quellen zu Git. Weitergehende Einführungen in deutscher Sprache finden Sie z.B. unter

<https://lerneprogrammieren.de/git/>

oder

<https://www.ionos.de/digitalguide/websites/web-entwicklung/git-tutorial/>.

Eine ausführliche Dokumentation und eine Kommandoreferenz finden Sie unter

<https://git-scm.com/>.

GitHub bietet ebenfalls umfangreiche Tutorials, Erklärungen und Dokumentationen für die Verwendung GitHub's.

<https://docs.github.com/en/get-started>

2 Einrichten eines Repositorys mit GitHub

2.1 Überblick der einzelnen Arbeitsschritte

Erstellen eines Repositorys für das Team in GitHub

- Erstellen jeweils eines Github-Accounts für jedes Teammitglied. Diesen Schritt muss jedes Teammitglied durchführen. Jedes Teammitglied benötigt dazu eine E-Mail-Adresse.
- Erstellen eines Repositorys in GitHub. Diesen Schritt muss nur ein Teammitglied stellvertretende für das Team durchführen.
- Freigeben des Repositorys für alle Teammitglieder.

Erzeugen und Einrichten der SSH-KEYs

Um auf das Repository in GitHub zugreifen zu können muss sich jedes Teammitglied, oder genauer dessen Rechner, authentisieren. Dies kann über verschiedene Möglichkeiten geschehen. Der in diesem Rahmen vorgestellte Weg ist die Verwendung von SSH-KEYs.

- Erzeugung von SSH-KEYs für den verwendeten Computer
- Registrierung der SSH-KEYs in GitHub

Klonen und Konfigurieren des Repositorys

- Das Klonen des Repositorys entspricht dem Anlegen der Arbeitskopie auf dem eigenen Rechner.
- Das Git-Repository muss konfiguriert werden, damit Git Ihren GitHub-Account kennt.

2.2 Erstellen eines Repositorys für das Team in GitHub

2.2.1 Anlegen der Accounts für GitHub

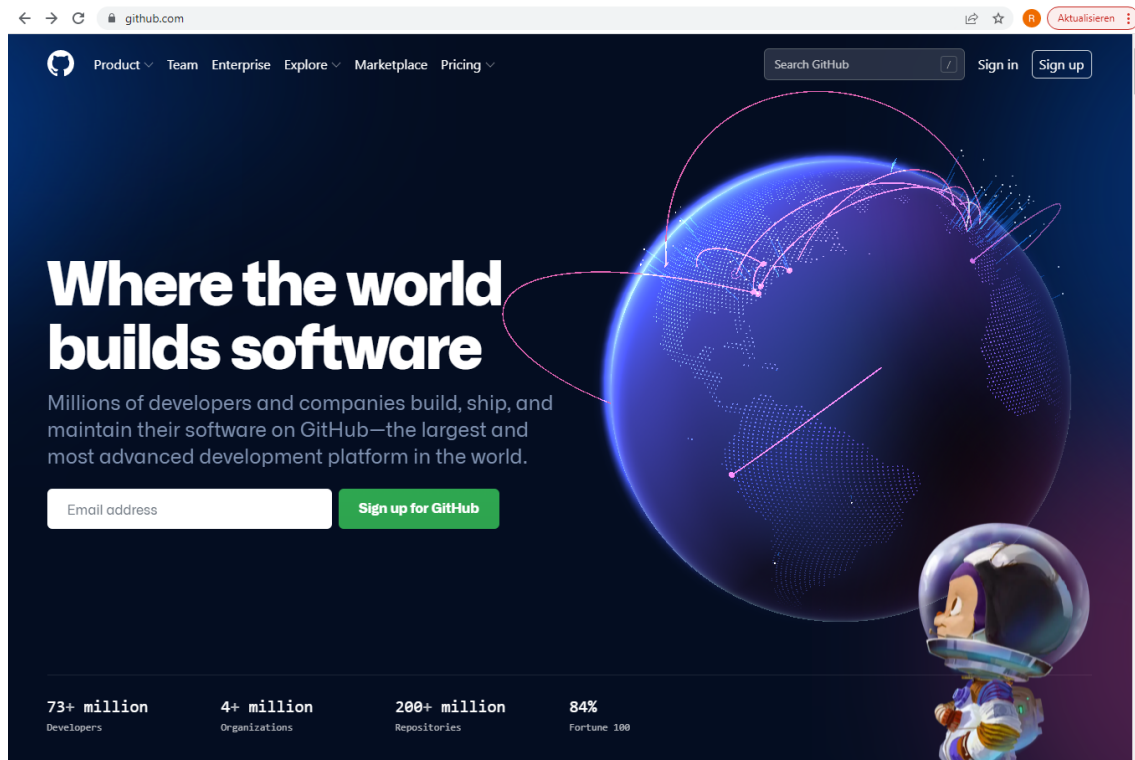


Abbildung 2.1: GitHub SignIn

Jedes Teammitglied muss unter GitHub.com einen Account erstellen. Dazu wird jeweils eine E-Mailadresse benötigt. Loggen Sie sich mit diesem Account in GitHub ein.

2.2.2 Erstellen eines Repositorys mit GitHub

Ein Teammitglied muss nun stellvertretend für das Team ein Repository erstellen, welches später für alle Teammitglieder freigegeben wird.

- Dazu klickt dieses Teammitglied auf den Button **New** (siehe Abbildung 2.2).
- Legen Sie nun einen Namen für das Repository fest (siehe Abbildung 2.3). Optional können Sie eine Beschreibung festlegen. Zusätzlich können Sie die Checkboxes für **Add a README** und **Add .gitignore** aktivieren. Als Template für sollen Sie, in diesem Fall, **Python** wählen.

- Klicken Sie auf **Create Repository**, um die Erstellung des Repositorys abzuschließen. Sie gelangen nun zur Ansicht des noch leeren Repositorys (siehe Abbildung 2.4). Beachten Sie, dass Sie das Repository unter im Browser ersichtlichen URL erreichen können. Diese URL ist die Adresse des Repositorys in GitHub. Sie folgt der Form “github.com/USERNAME/REPOSITORYNAME”. (Alternativ ist unter “Account-> Settings->Repositories” eine Liste alle REpositories verfügbar. Unter “Account->Settings->Your Repositories” finden Sie nur die Repositories, welche unter Ihrem Account erstellt wurden.)

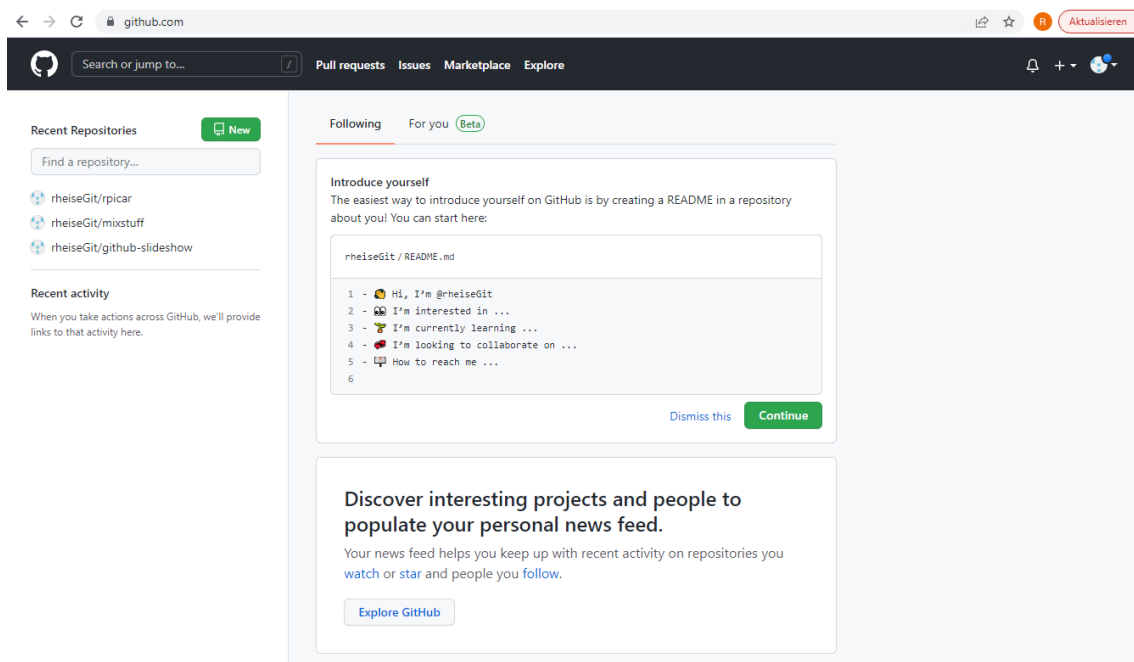
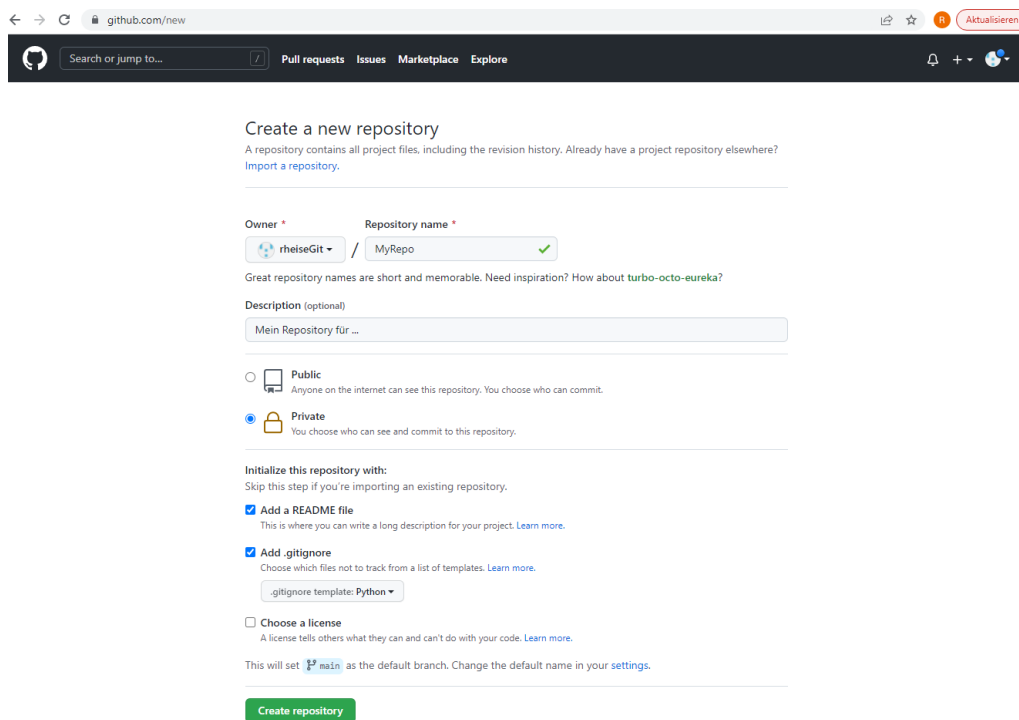


Abbildung 2.2: Beispielhafte Starseite nach dem Einloggen Die Schaltfläche ****New**** ermöglicht die Erstellung eines neuen Repositories.



The screenshot shows the 'Create a new repository' page on GitHub. The page has a dark header with the GitHub logo, a search bar, and navigation links: 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main content area is titled 'Create a new repository' and includes a description: 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this, there are two input fields: 'Owner' (set to 'rheiseGit') and 'Repository name' (set to 'MyRepo' with a green checkmark). A note says: 'Great repository names are short and memorable. Need inspiration? How about [turbo-octo-eureka?](#)'. There is a 'Description (optional)' text area with the placeholder 'Mein Repository für ...'. Below the description, there are two radio buttons for visibility: 'Public' (selected) and 'Private'. The 'Public' option has a subtext: 'Anyone on the internet can see this repository. You choose who can commit.' The 'Private' option has a subtext: 'You choose who can see and commit to this repository.' Below the visibility options, there is a section 'Initialize this repository with:' with a subtext: 'Skip this step if you're importing an existing repository.' There are three checkboxes: 'Add a README file' (checked), 'Add .gitignore' (checked), and 'Choose a license' (unchecked). The 'Add a README file' checkbox has a subtext: 'This is where you can write a long description for your project. [Learn more.](#)'. The 'Add .gitignore' checkbox has a subtext: 'Choose which files not to track from a list of templates. [Learn more.](#)' and a dropdown menu showing '.gitignore template: Python'. The 'Choose a license' checkbox has a subtext: 'A license tells others what they can and can't do with your code. [Learn more.](#)'. Below the checkboxes, there is a note: 'This will set `main` as the default branch. Change the default name in your [settings](#).' At the bottom, there is a green 'Create repository' button.

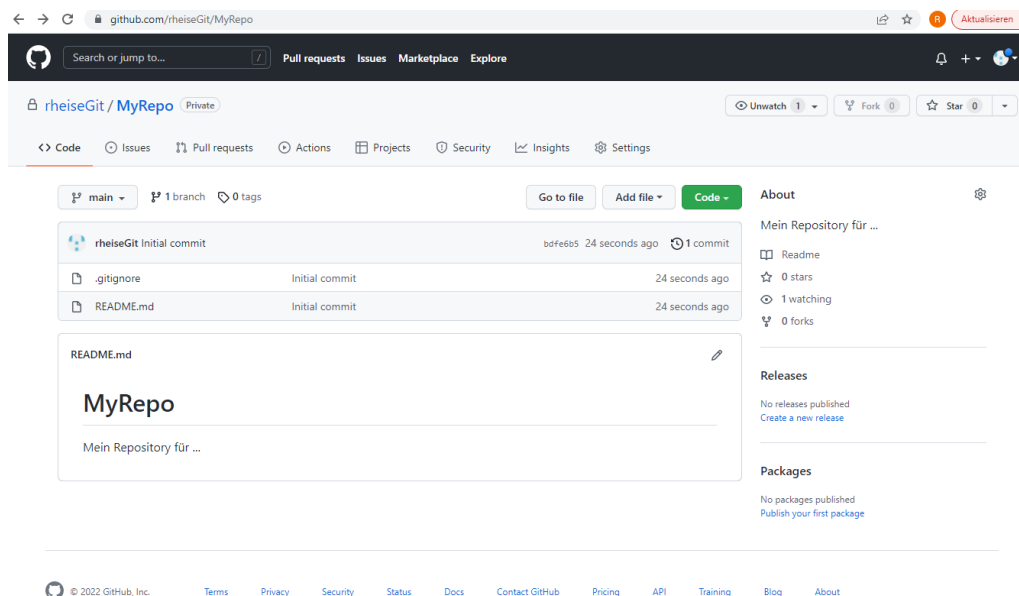
Abbildung 2.3: Anlegen eines Repositorys

Abbildung 2.4: Ansicht des leeres Repository Die Dateien README.md und .gitignore wurden optional erzeugt. Die URL, ersichtlich in der Adressleiste des Browsers (github.com/rheiseGit/MyRepo) stellt die URL dieses BeispielRepositorys dar.

2.2.3 Freigabe des Repositorys für alle Teammitglieder

Das angelegte Repository muss nun für alle Teammitglieder freigegeben werden. Dies ist ein Schritt, welcher spezifisch für das Repository ist! Klicken Sie dazu in der Ansicht des Repositorys auf “Settings” und dann im linken Menu auf “Collaborators” (siehe Abbildung 2.5). Als nächsten klicken Sie bitte unter “Manage access” auf den Button “Add people” (siehe Abbildung 2.6) und suchen nacheinander nach den GitHub-Nutzernamen Ihrer Teammitglieder, um diesen Zugang zum Repository zu gewähren. Die einzelnen Teammitglieder erhalten als Folge jeweils eine E-Mail an ihre in GitHub registriert E-Mail-Adresse. Jedes Teammitglied muss den Anweisung der E-Mail folgen, um die Einladung zur Kooperation anzunehmen um Zugang zu Repository zu erhalten. Nach der Annahme der Einladung kann jedes Teammitglied das Repository unter dessen Adresse erreichen und einsehen. Prüfen Sie dies!

Nun kann prinzipiell jedes Team-Mitglied auf das Repository in GitHub zugreifen. Um diesen Zugriff zu erleichtern und die Authentisierung zu erleichtern, soll diese Authentisierung per SSH-KEYs erfolgen.

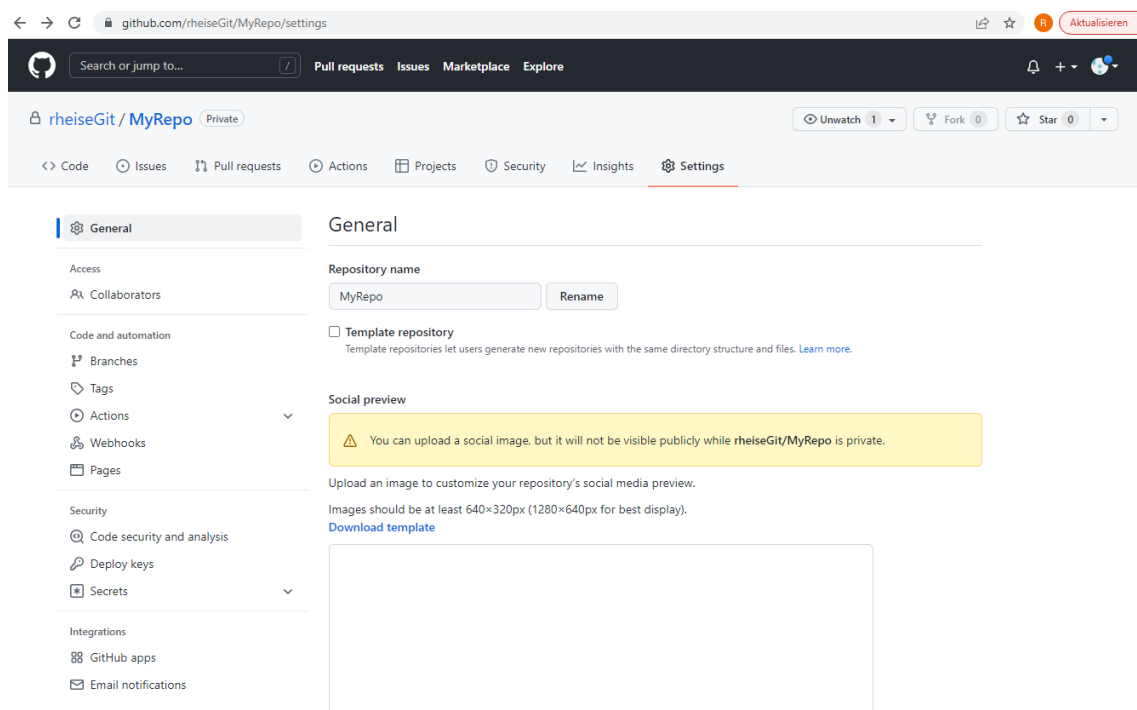


Abbildung 2.5: Settings des Repositorys

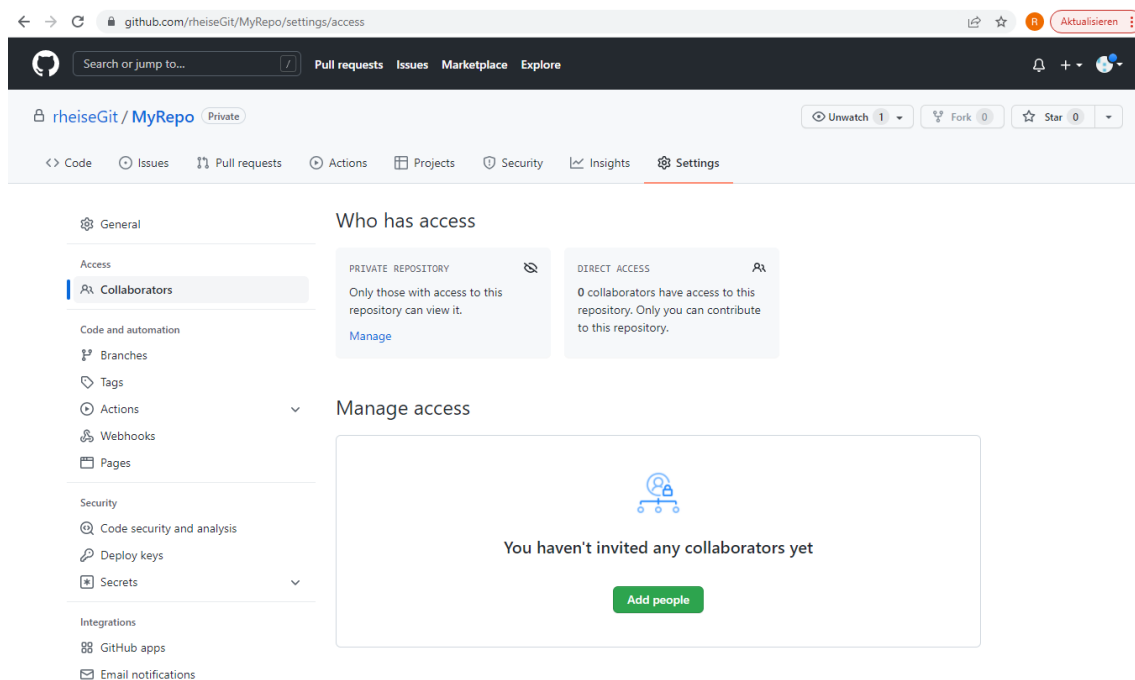


Abbildung 2.6: Zufügen von Teammitgliedern (Collaborators)

2.3 Erzeugen und Einrichten der SSH-KEYs für den Zugriff auf GitHub

Auf ein GitHub-Repository kann auf verschiedenen Wegen zugegriffen werden. In jedem Fall muss sich der Nutzer authentifiziert werden. In dieser beispielhaften Anleitung soll die Authentisierung per SSH-KEYs und damit automatisiert, während der späteren Verwendung Gits, erfolgen.

2.3.1 Erzeugen eines ssh-keys auf dem Raspberry Pi

Führen Sie das folgende Kommando im Terminal aus. Das Programm fragt nach einem Speicherort für die Keys. Behalten Sie die Voreinstellung (/home/pi/.ssh/id_rsa) bei indem Sie einfach ENTER drücken. Falls Sie eine Passwort (passphrase) verwenden wollen, merken Sie sich dieses gut. Sie müssen es bei jeder Interaktion mit dem Repository eingeben. Falls kein Passwort verwenden wollen drücken Sie ENTER. Diesen Fall entfällt eine Anfrage eines Passworts bei einer Interaktion mit dem Repository.

1 `ssh-keygen -o`

Durch das obige Kommando wurden nun u.a. ein Public-Key für Ihren RPi erzeugt und in der Datei `'/home/pi/.ssh/id_rsa.pub'` gespeichert. Mittels des folgenden Kommandos können sie diesen Public-Key ausgeben.

```
1 cat /home/pi/.ssh/id_rsa.pub
```

2.3.2 Registrierung des ssh-keys in GitHub

Im nächsten Schritt muss der Public-Key in GitHub registriert und damit freigegeben werden. Dies ist ein account-spezifischer Schritt!

- Loggen Sie sich dazu in GitHub ein, falls dies noch nicht geschehen ist, und begeben sich auf die Seite der Accounts-Settings (siehe Abbildung 2.8).
- Klicken Sie nun auf **SSH and GPG keys** (siehe Abbildung 2.8)
- Klicken Sie nun auf den Button **New SSH key** (siehe Abbildung 2.8)
- Geben Sie dem SSH-KEY einen Titel (siehe Abbildung 2.9). Dieser sollte Ihren verwendeten Rechner (z.B. RPi) beschreiben, da der Key von diesem Rechner verwendet wird. Kopieren Sie Ihren SSH-KEY mittels Copy&Paste in das Feld "Key". Sie erhalten ihn indem Sie in einem Terminal des RPis das Kommando `"cat /home/pi/.ssh/id_rsa.pub"` ausführen.
- Schließen Sie die Registrierung ab indem Sie auf den Button **Add SSH key** klicken.

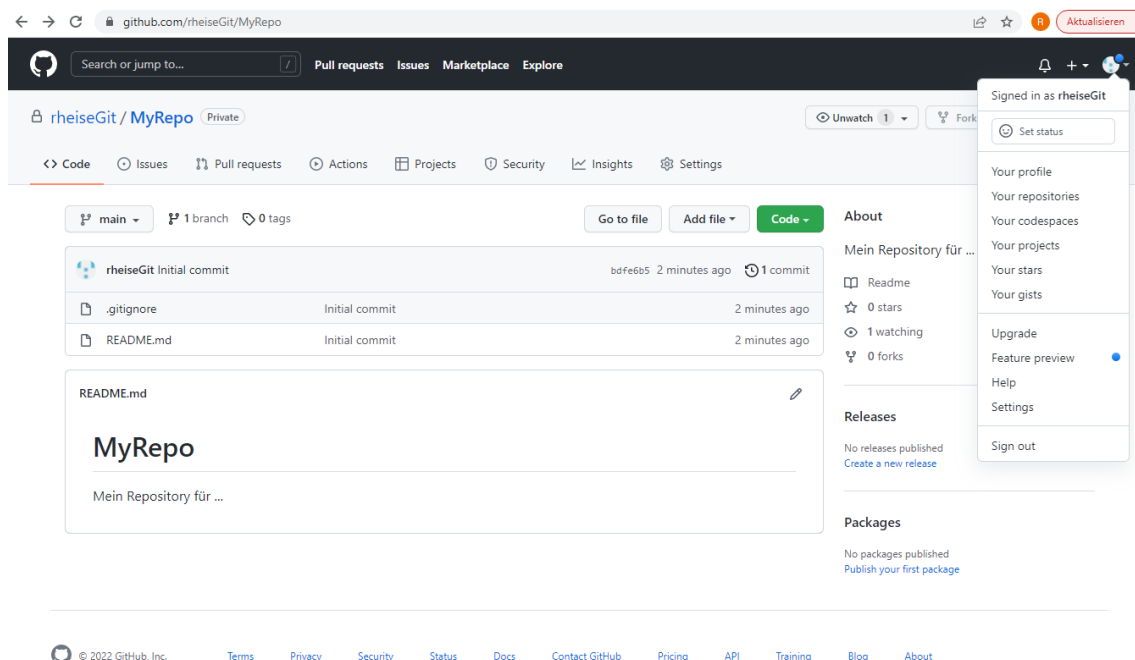


Abbildung 2.7: Auswahl Account Settings

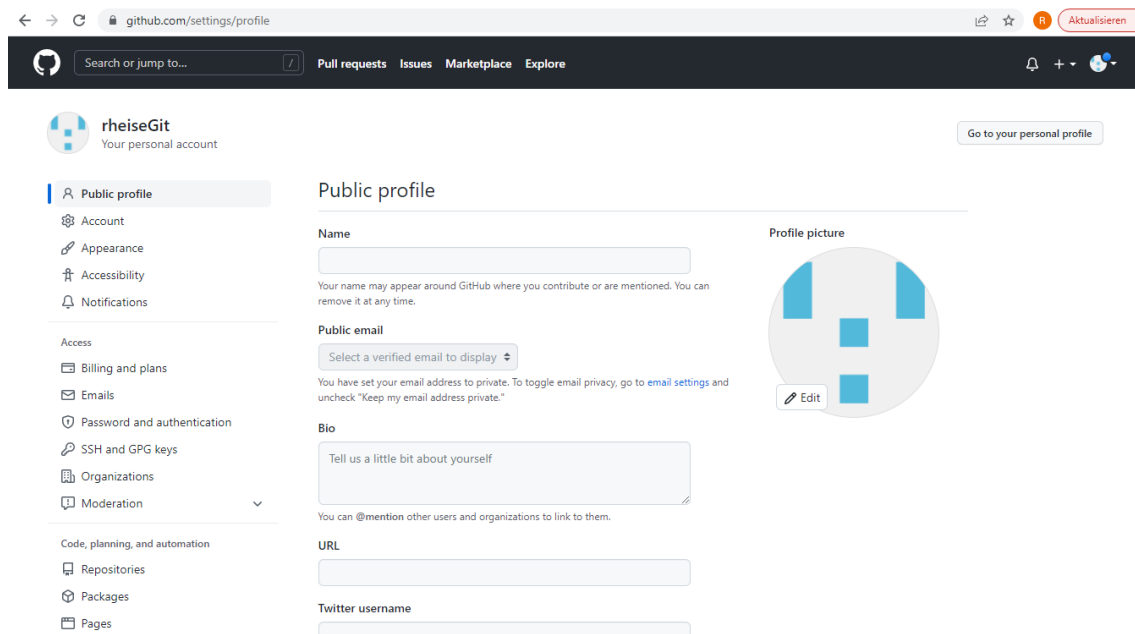


Abbildung 2.8: Account Settings Um einen SSH-KEY zu registrieren, klicken Sie auf SSH and GPG keys”

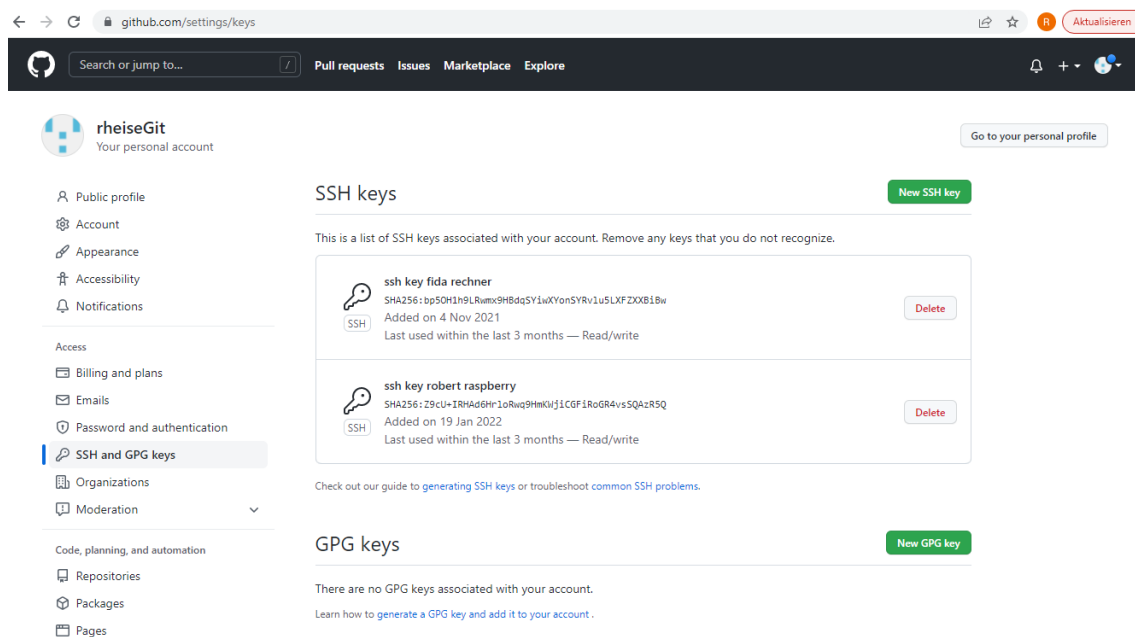


Abbildung 2.9: SSH und GPG Settings Klicken auf den Button ”New SSH key”, um ihren Public-Key zu registrieren.

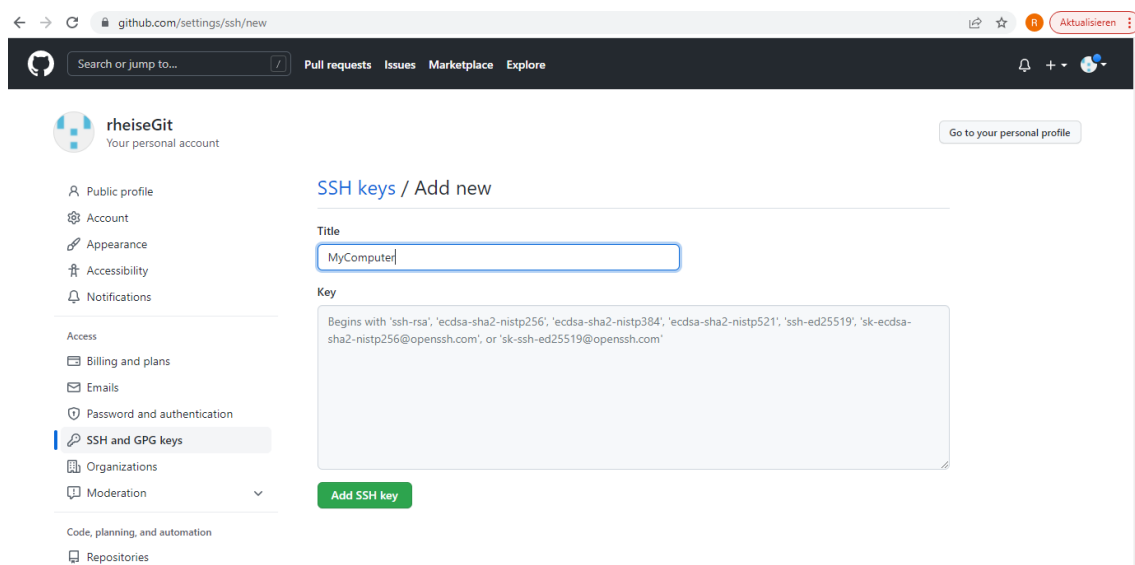


Abbildung 2.10: Registrieren eines neuen Public-Keys Im Feld Title sollten die Ihren Computer von welchem der Public-Key stammt beschreiben. In das Feld "Key" muss der Public-Key kopiert werden. "Title"

2.3.3 Erzeugen eines ssh-keys für Windows

Sie können per SSH-KEY auch von einem Windows-Rechner auf das GitHub-Repository zugreifen.

Siehe dazu: <https://pimylifeup.com/raspberry-pi-ssh-keys/>

2.4 Klonen des Repositorys mittels SSH

Rufen Sie in GitHub die Seite des zu klonenden Repositories auf. Unter dem Button **Code** wählen Sie nun **SSH** und kopieren den Identifikator des Repositorys (siehe Abbildung 2.11).

Führen Sie in einem Terminal auf dem RPi das folgende Kommando unter Verwendung des Identifikators aus. Als Resultat wird in dem Ordner der Ausführung des Kommandos ein neuer Ordner erscheinen, welche das "kopierte" Repository enthält (siehe auch Abbildung ??).

```
1 git clone git@github.com:USERNAME/REPOSITORYNAME.git
```

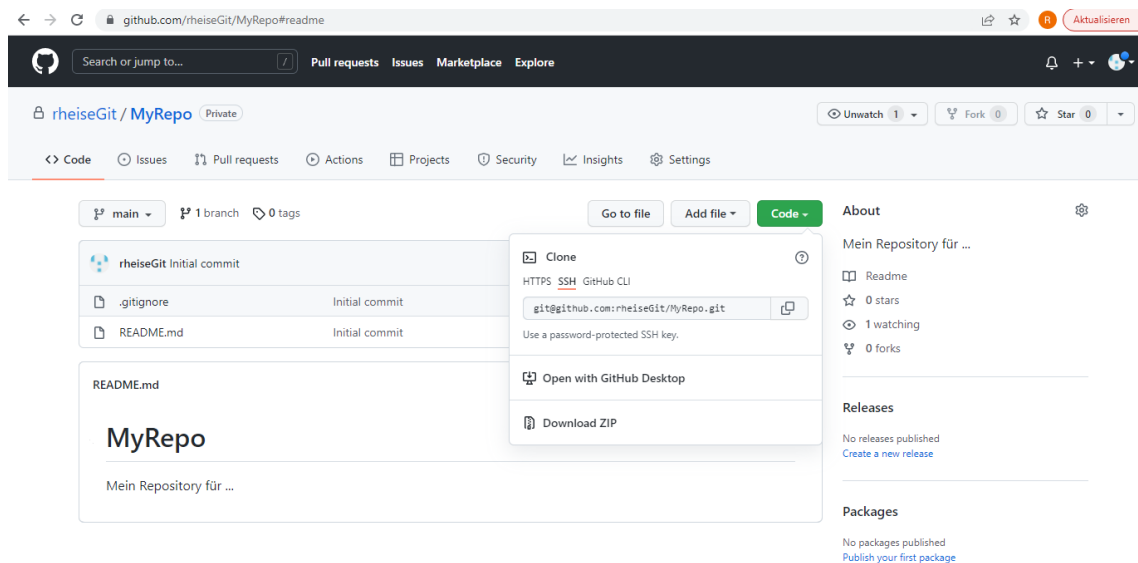


Abbildung 2.11: GitHub: Kopieren des Identifikators für SSH Der in diesem Beispiel zu kopierende Identifikator lautet "git@github.com:rheiseGit/MyRepo.git"

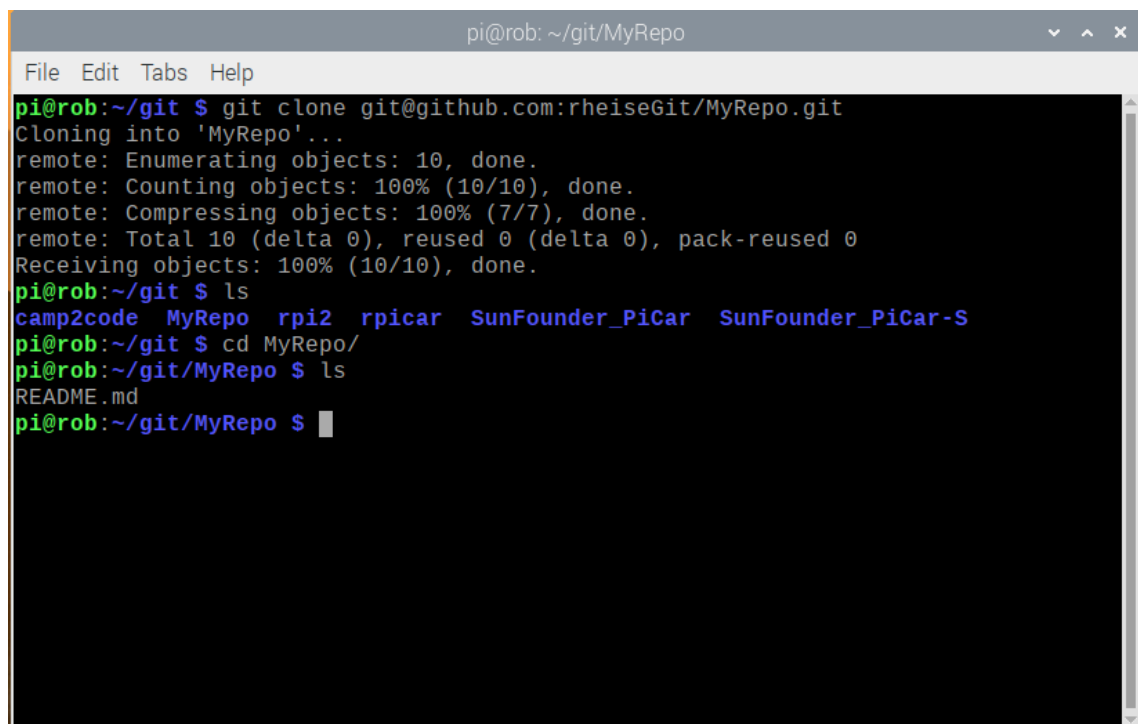


Abbildung 2.12: Terminal: Klonen des Repositories mittel SSH Ausführung des Kommandos "git clone git@github.com:rheiseGit/MyRepo.git" und Anzeige des neuen Ordners, welches die Arbeitskopie des Repositories enthält.

2.5 Konfiguration des Repositorys unter Git

In diesem Konfigurationsschritt werden der GitHub-Username und die verwendete E-Mail-Adresse in eine für das Repository spezifischen Konfigurationsdatei geschrieben. Wechseln Sie dazu zuerst den den Order des Repositorys.

```
1 cd REPOSITORYNAME
```

Führen Sie nun die folgenden zwei Git-Kommandos aus, um den Usernamen und die E-Mail-Adresse in die Datei ~/REPOSITORYNAME/.git/config aufzunehmen.

```
1 git config user.name "USERNAME"
```

```
1 git config user.email EMAIL
```

Mittels des folgenden Git-Kommandos können Sie die Existenz der gesetzten Werte in der Konfigurationsdatei überprüfen.

```
1 git config --list
```

3 Anwendung der Software Git

3.1 Auswahl grundlegender Git-Kommandos

Im Folgenden Abschnitt werden folgende Git-Kommandos in Form einer Aufgabe vorgestellt.

- **git add**
- **git commit**
- **git push**
- **git pull**
- **git status**

Dabei wird auf einige grundlegende Basiskonzept Gits eingegangen.

1. lokaler Arbeitsbereich (workspace, working tree)
2. lokaler Staging-Bereich (staging area)
3. lokale Versionierung mittels Commit (local branch)
4. zentrale Versionierung im zentralen Respository (remote branch)

Abbildung 3.1 illustriert diese grundlegenden Konzepte. Eine ausführliche Dokumentation Gits und eine Kommandoreferenz finden Sie unter <https://git-scm.com/>.

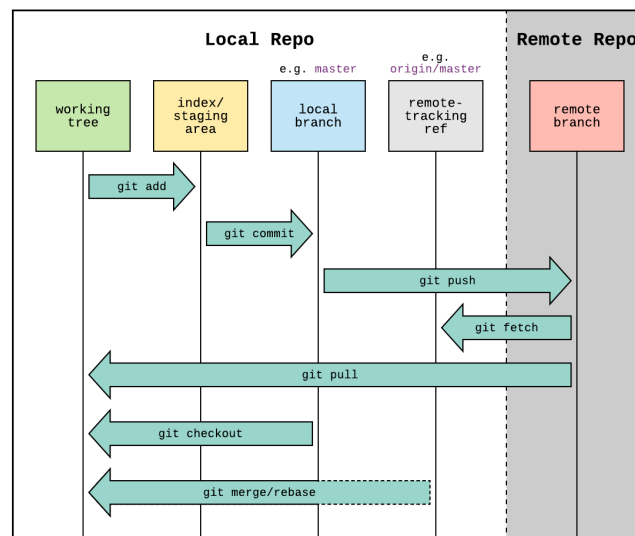


Abbildung 3.1: Git Workflow Mittels verschiedener Git-Kommandos werden Änderungen in verschiedene Bereiche des Repositorys (REPO) übertragen.

3.2 Demoaufgabe

Git stellt ein Vielzahl von Kommandos zur Verfügung, welche Sie im Terminal ausführen können. Dabei gilt, **Kommandos, welche im Ordner des Repository's ausgeführt werden, beziehen sich auf dieses Repository**. Mit dem Wechsel in dies Verzeichnis betreten Sie den Git-Arbeitsbereich des Repositorys. Öffnen Sie ein Terminal und gehen Sie begeben sich in den Ordner des Repository's.

```
1 cd REPOSITORYNAME
```

Um die Verwendung der Kommandos zu demonstrieren bzw. auszuprobieren, soll im Folgenden jedes Teammitglied eine Testdatei erzeugen und diese dem Repository zufügen. Final sollen alle auf diese Weise entstandenen Testdateien auf ihren lokale Arbeitsbereich übertragen.

Erzeugen Sie eine leere Testdatei mittels dem Bash-Kommando **touch** innerhalb des Ordners des Repositorys. Wählen Sie einen für Sie spezifischen Dateinamen, sodass es jeder Dateiname im Team nur einmal vorkommt z.B. durch das Einbeziehen Ihres Namens. Durch das Erzeugen der Datei fügen sie dies ihrem Git-Arbeitsbereich (workspace) hinzu.

```
1 touch TESTDATEINNAME.txt
```

Prüfen Sie mittels der Bash-Kommandos **ls**, ob die Datei existiert.

```
1 ls
```

Frage Sie mit dem Git-Kommando **git status** den Status von Git ab. Die Testdatei wird in der Ausgabe als “nicht verfolgt” (untracked) bezeichnet. Zusätzlich verweist Git auf das Kommando **git add**.

```
1 git status
```

Fügen sie mittels **git add** die Testdatei aus Git-Arbeitsbereich (workspace) dem Git-Staging-Bereich (staging area) zu. Die Datei wird dadurch nicht verändert oder kopiert. Sie wird nur durch Git anders behandelt. Mittels **git add** fügen Sie auch Änderungen einer Datei dem Staging-Bereich zu! Im Staging-Bereich werden Änderungen gesammelt, welche Sie später gemeinsam in die Versionierung aufnehmen können.

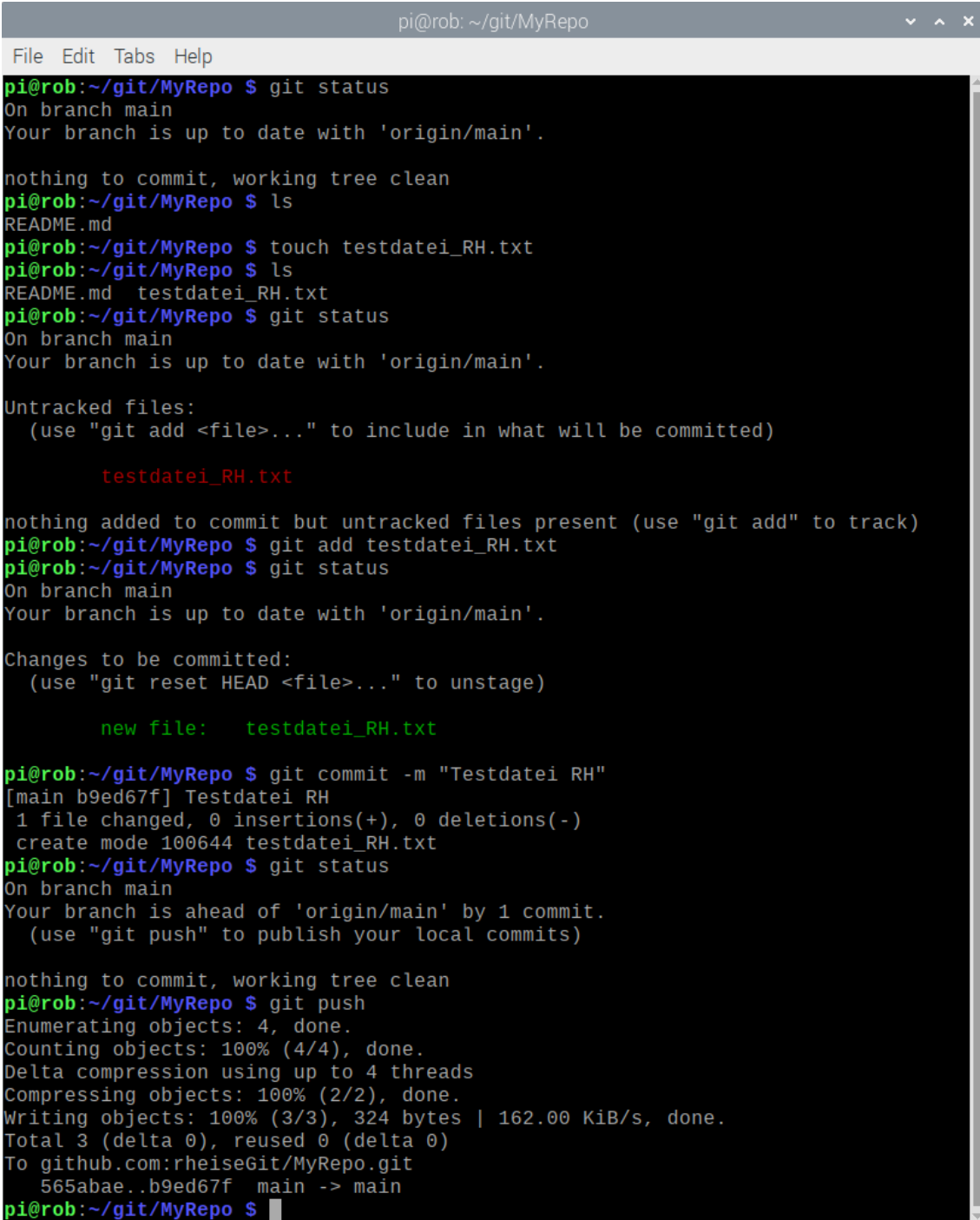
```
1 git add TESTDATEINAME.txt
```

Fragen Sie erneut mit **git status** den Status ab. Die Datei gilt nun als “staged”. Um sie nun in die eigentlich Versionskontrolle einzubinden, führen Sie das folgende Kommandos **git commit** aus. Verwenden Sie einen kurzen, aber informativen Kommentar. Einzelnen Commits stellen neue Versionen der Software dar.

```
1 git commit -m "EIN KOMMENTAR WELCHER DEN COMMIT BESCHREIBT"
```

Die erneute Abfrage des Status zeigt Ihnen nun an, dass der soeben erstellt “Commit” Ihrer Arbeitskopie noch nicht im zentralen Repository vorhanden ist. Um Ihre Änderungen ins das zentrale Repository zu übertragen, führen Sie das Kommandos **git push** aus. Abbildung 3.2 fasst die einzelnen Schritt beispielhaft zusammen.)

```
1 git push
```



```
pi@rob: ~/git/MyRepo
File Edit Tabs Help
pi@rob:~/git/MyRepo $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
pi@rob:~/git/MyRepo $ ls
README.md
pi@rob:~/git/MyRepo $ touch testdatei_RH.txt
pi@rob:~/git/MyRepo $ ls
README.md  testdatei_RH.txt
pi@rob:~/git/MyRepo $ git status
On branch main
Your branch is up to date with 'origin/main'.

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

        testdatei_RH.txt

nothing added to commit but untracked files present (use "git add" to track)
pi@rob:~/git/MyRepo $ git add testdatei_RH.txt
pi@rob:~/git/MyRepo $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   testdatei_RH.txt

pi@rob:~/git/MyRepo $ git commit -m "Testdatei RH"
[main b9ed67f] Testdatei RH
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 testdatei_RH.txt
pi@rob:~/git/MyRepo $ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

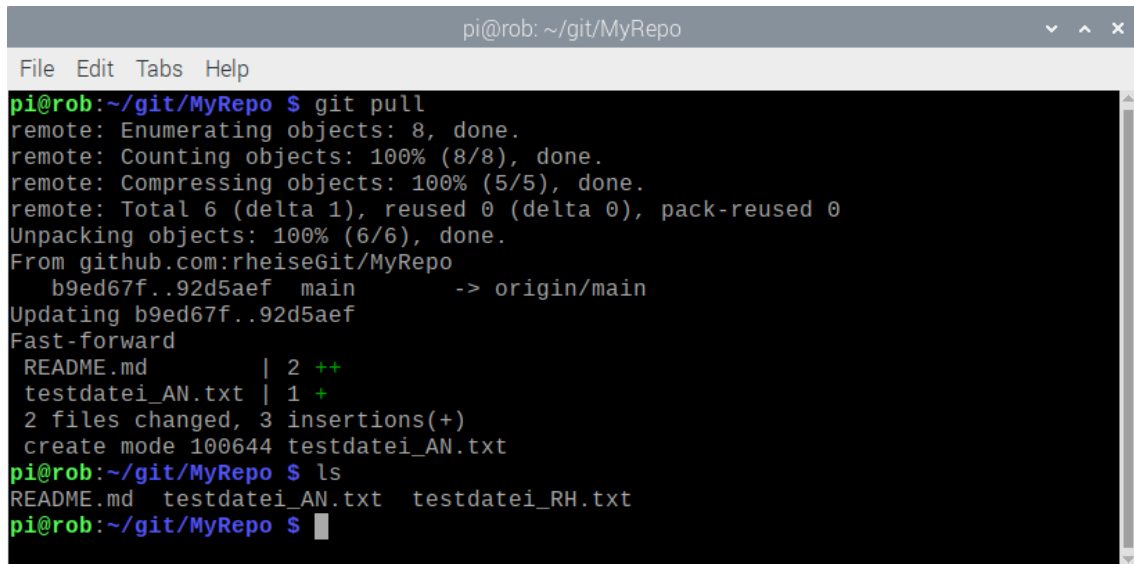
nothing to commit, working tree clean
pi@rob:~/git/MyRepo $ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 324 bytes | 162.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:rheiseGit/MyRepo.git
 565abae..b9ed67f  main -> main
pi@rob:~/git/MyRepo $
```

Abbildung 3.2: Terminal: Erstellen der Testdateien

Nachdem alle Teammitglieder ihre Testdateien in das zentrale Repository per git push übertragen haben, werden diese in GitHub sichtbar. Um diese Änderungen in auf Ihre Lokale Arbeitskopie zu übernehmen führen Sie **git pull** aus. Sie erhalten Information über die Änderungen

(siehe auch Abbildung 3.3).

```
1 git pull
```



```
pi@rob:~/git/MyRepo $ git pull
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
From github.com:rheiseGit/MyRepo
   b9ed67f..92d5aef  main      -> origin/main
Updating b9ed67f..92d5aef
Fast-forward
 README.md      | 2 ++
 testdatei_AN.txt | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 testdatei_AN.txt
pi@rob:~/git/MyRepo $ ls
README.md testdatei_AN.txt testdatei_RH.txt
pi@rob:~/git/MyRepo $
```

Abbildung 3.3: Terminal: Ziehen (pull) der Testdateien der Teammitglieder. Im Beispiel wurden Änderungen an README und eine neue Testdatei übertragen.

Sie können diese Aufgabe wiederholen und statt Dateien neu zu erstellen nur den Inhalt der Testdateien ändern.

3.3 Auswahl weitere Kommandos

Zufügen aller neuer Dateien und Änderungen bestehender Dateien in den Staging-Bereich

```
1 git add --all
```

Commit inkl. der vorlaufenden Übernahme alle Änderungen in den Staging-Bereich

```
1 git commit --am "KOMMENTAR"
```

Erweitern des letzten Commits

```
1 git commit --amend
```

Entfernen aller Änderungen des Staging-Bereichs

```
1 git reset
```

Rücknahme eines Commits unter Beibehalt der Änderungen des Staging Bereichs

```
1 git reset --soft HEAD~1
```

Rücknahme eines Commits unter Löschung der Änderungen des Staging Bereichs

```
1 git reset --hard HEAD~1
```

Liste der einzelnen Commits in umgekehrter chronologischer Reihenfolge

```
1 git log
```

Liste der verfolgten Dateien

```
1 git ls-files
```