
# Git für Maker, Teil 2

**In der Make-Ausgabe 1/25 haben wir erklärt, was genau Git ist, wie man es einrichtet und wie man die verschiedenen Funktionen des Programms für die lokale Versionsverwaltung nutzt. In dem zweiten Teil wird auf diesem Wissen aufgebaut, um weitere Funktionen von Git nutzen zu können: Projekte nicht nur lokal zu verwalten, sondern sie online auf der Plattform GitHub zu teilen und dort zu verwalten.**

von Daniel Schwabe

Die Vorgänge werden beispielhaft an GitHub gezeigt, weil es sich dabei um die größte Git-Plattform handelt. Ähnliche Services funktionieren grundlegend gleich. Befehle und Konzepte lassen sich auch auf andere Anbieter anwenden.

## Was kann GitHub denn?

Lädt man ein mit Git verwaltetes Projekt auf GitHub hoch, überträgt sich auch der gesamte Projektverlauf, wie er lokal abgespeichert wurde. Aber nicht nur der aktuelle Stand des Projekts ist online verfügbar, sondern auch alle früheren Versionen, die durch Commits versioniert wurden. Dadurch können andere Nutzer sämtliche Änderungen nachvollziehen und gezielt auf frühere Versionen zurückgreifen. Auch Branches werden online abgebildet.

Neben der reinen Bereitstellung bietet GitHub auch die Möglichkeit zur Zusammenarbeit. Andere Entwickler können nicht nur den Code einsehen, sondern auch aktiv mitwirken. Sie können z. B. eigene Commits vorlegen, die der Besitzer des Repositorys überprüft und entweder akzeptiert oder ablehnt. So bleibt die Kontrolle über das Projekt in der Hand des Besitzers, während gleichzeitig mehrere Personen gemeinsam daran arbeiten können.

## Bei GitHub registrieren

Um auf GitHub arbeiten zu können, braucht man als Erstes einen Account auf der Plattform. Dafür klickt man auf github.com oben rechts auf die Schaltfläche „Sign up“. Dort trägt man dann seine E-Mail-Adresse und ein Passwort ein. Was in dem Feld „Username“ eingetippt wird, verwendet GitHub später als öffentlichen Anzeigenamen, der von anderen Usern einsehbar ist. Ob man dort ein Alias oder den Klarnamen verwenden möchte, muss man selbst entscheiden.

Um alle Funktionen (Beitragen von Code, Ändern der Kontoeinstellungen etc.) von GitHub nutzen zu können, muss man für sein Konto die sogenannte Zwei-Faktor-Authentifizierung aktivieren. Was das ist und wie man sie nutzt, steht im Kasten „Zwei-Faktor-Authentifizierung“.

## Ein Onlinerepository anlegen

Im ersten Teil dieser Artikelreihe haben wir ein lokales Repository auf dem Computer ange-

## Kurzinfo

- Onlinerepository bei GitHub anlegen
- Commits hoch- und herunterladen
- In öffentlichen Repositorys mitarbeiten

## Mehr zum Thema

- Daniel Schwabe, Git für Maker, Make 1/25, S. 54
- Florian Schäffer, AVR-Programme debuggen, Teil 1, Make 4/24, S. 104

Alles zum Artikel im Web unter
*make-magazin.de/x96w*

## Zwei-Faktor-Authentifizierung

Unter einer Zwei-Faktor-Authentifizierung (meistens mit 2FA abgekürzt) versteht man, dass man sich bei der Anmeldung z. B. auf einer Website mit dem Passwort und einer zweiten Verifizierungsmethode, einem zweiten Faktor, anmelden muss.

Das kann ein immer wechselnder Code sein, den man bei jedem Anmeldeversuch per E-Mail oder SMS zugeschickt bekommt oder der von einer speziellen App generiert wird. Aber auch spezielle USB-Geräte können ein zweiter Faktor sein. Diese Geräte nennen sich „Passkeys“. Nutzt man so ein Gerät, wird auf der Website, bei der man sich mit diesem Passkey anmelden möchte, eine Datei gespeichert, mit der das USB-Gerät zweifelsfrei identifiziert werden kann.

Um 2FA auf GitHub einzurichten, klickt man nach der Anmeldung auf der Startseite oben rechts auf das Profilbild des Accounts und dann auf „Settings“. Im sich neu öffnenden Fenster befinden sich die entsprechenden Optionen unter „Password and authentication“. Hier kann man das Passwort ändern, einen Passkey oder 2FA über eine App oder SMS einrichten.

Wer einen Passkey besitzt, kennt sich schon mit dessen Einrichtung aus. Deshalb wird hier exemplarisch die Möglichkeit über eine Authenticator-App genutzt. Dafür klickt man unten auf der Seite auf den großen grünen „Enable two-factor authentication“-Button. Im neuen Fenster wird prominent ein QR-Code angezeigt.

Dieser QR-Code beinhaltet das sogenannte Geheimnis für diesen Account. Dabei handelt es sich um eine einmalige Zeichenfolge, mit der dann in Abhängigkeit von der Uhrzeit alle 30 Sekunden neue Sicherheitscodes (die dann der zweite Faktor sind) erzeugt werden. Dieses Geheimnis darf man niemals weitergeben.

Um mit diesem QR-Code Passwörter für den zweiten Faktor zu generieren, braucht man eine spezielle App wie den Google Authenticator oder den Microsoft Authenticator. Wer schon für eine andere Website so eine App nutzt, kann dieses Geheimnis dort auch einspeichern. Hier ist einmal der Vorgang im Google Authenticator erklärt.

Nach dem Öffnen der App tippt man auf das bunte Plus-Zeichen unten rechts in der Ecke. Dort wählt man dann „QR-Code scannen“ aus und richtet die Handykamera auf den QR-Code auf dem Bildschirm. Und das war es auch schon. Jetzt hat man einen Eintrag zu diesem GitHub-Account, der sich alle 30 Sekunden aktualisiert und einen Code generiert.

Um die 2FA auf der Website fertig einzurichten, muss man diesen unterhalb des QR-Codes eintragen und bestätigen. Hat man das getan, bekommt man 16 Sicherheitscodes angezeigt. Das sind feststehende Einmalpasswörter, die immer gelten und nicht von der Zeit abhängig sind. Sie sind hilfreich, wenn man keinen Zugriff mehr auf den zweiten Faktor hat (weil einem z. B. das Handy kaputtgegangen ist). Diese Codes muss man sicher irgendwo aufbewahren.

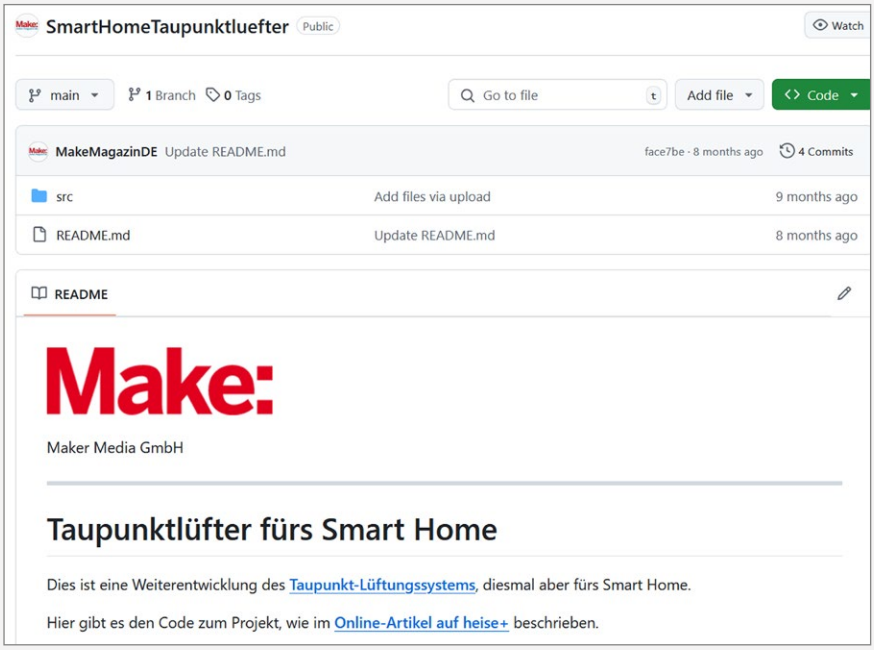
Wenn man sich jetzt bei GitHub anmeldet, gibt man zuerst die E-Mail und das Passwort ein und danach das aktuelle Einmalpasswort aus dem Authenticator.

## Readme

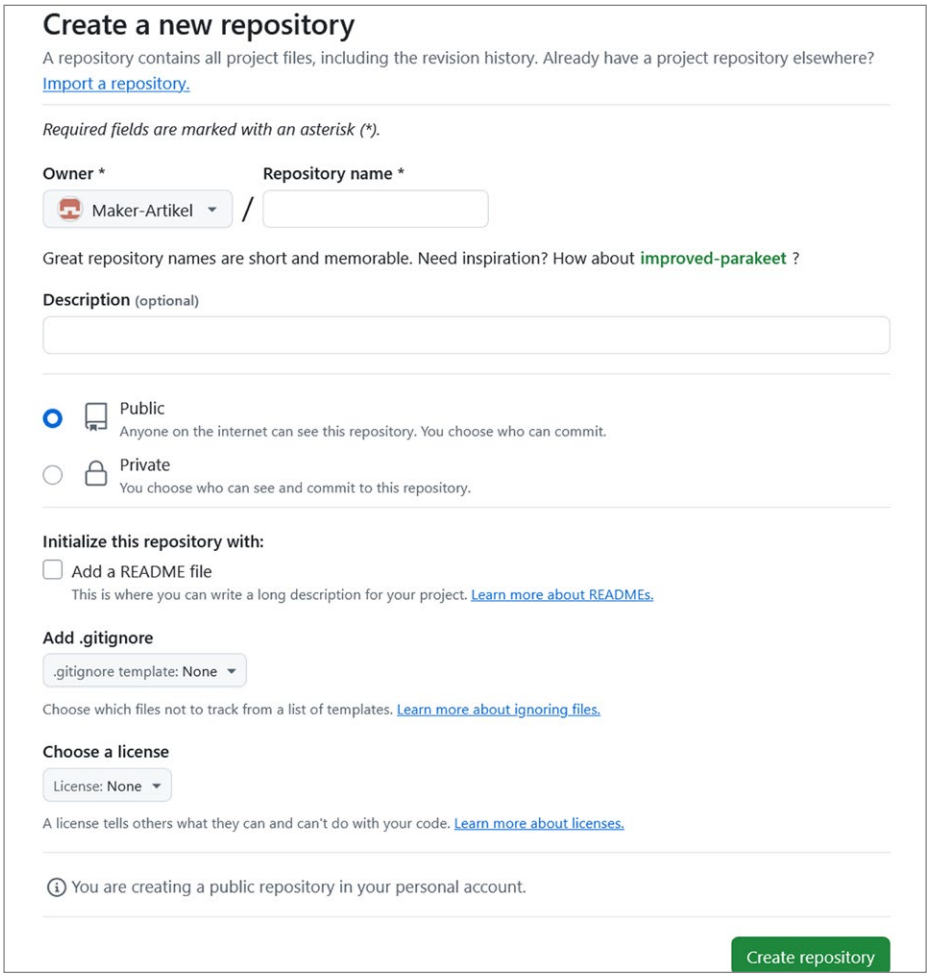
Aktiviert man bei der Erstellung eines Repositorys die Option „Add a README file“, wird das Repository nicht leer erzeugt, sondern mit einer bereits bestehenden Datei – der README.md.

Dabei handelt es sich um eine Textdatei im Markdown-Format, in der Informationen über das Projekt in Text- und Bildform oder Links bereitgestellt werden können. Der Inhalt der Readme wird auf der Hauptseite des Repositorys direkt angezeigt.

Damit der Inhalt dieser Datei automatisch angezeigt wird, muss sie im obersten Verzeichnis des Repositorys liegen. Wenn man keine README.md bei der Erstellung des Repositorys angelegt hat, kann man das nachträglich machen.




**Eine Readme kann nicht nur Text, sondern auch Bilder und Links beinhalten.**




**Die Optionen beim Erstellen eines Repositorys sind übersichtlich. Vor allem Name und Sichtbarkeit („Public“/„Private“) sind wichtig.**

legt. Um nicht nur lokal zu arbeiten, sondern den Inhalt des lokalen Repositorys online bereitzustellen, muss man auch auf GitHub ein Repository anlegen.

Dafür klickt man nach der Anmeldung oben rechts auf das große Plus-Symbol und dann auf „New Repository“.

In der überschaubaren Maske für die Erstellung trägt man als Erstes einen Repository-Namen ein. Dieser darf keine Leerzeichen beinhalten. Trägt man einen Namen mit Leerzeichen (oder Umlauten) ein, werden automatisch Bindestriche eingefügt. Als Beispiel nutzen wir hier mal das Projekt „Taupunktluefter“ aus Make 4/24.

Darunter ist ein Feld für eine Beschreibung des Projektes. Darin sollte stehen, worum es in dem Projekt geht, z. B.: „Taupunktlüftersystem, um Räume zu trocknen, basierend auf Wemos D1 Mini. Integration ins Smarthome über MQTT“.

Als Letztes legt man fest, ob das Repository „Public“ oder „Private“ sein soll. Ist ein Projekt „Public“, kann es von jedem im Internet gesehen und heruntergeladen werden. Beitragen können nur Leute, die man als Besitzer explizit festgelegt hat. Ist das Projekt „Private“, können nur zugelassene Accounts (einschließlich man selbst) das Projekt sehen und herunterladen. Diese Einstellung kann nachträglich geändert werden. Deshalb wird für dieses Beispiel-Repository erst mal „Private“ ausgewählt.

Die drei Optionen „Add a README file“, „Add .gitignore“ und „Choose license“ werden in den jeweiligen Kästen „Readme“, „Gitignore“

## Gitignore

Die Datei .gitignore in einem Repository gibt an, welche Dateien und Ordner von Git ignoriert werden sollen. Das ist z. B. nützlich für Log-Dateien, temporäre Dateien oder Builds.

Sind eine explizite Datei, ein Ordner oder generell Dateien mit einer bestimmten Endung in der .gitignore eingetragen, werden sie einem auch nicht mehr für Commits vorgeschlagen oder mit dem „git status"-Befehl als verändert oder nicht verwaltet angezeigt (siehe Teil 1 dieser Artikelreihe). In der Praxis wird eine .gitignore-Datei verwendet, um besonders große Dateien zu ignorieren. Auch sicherheitsrelevante Dateien (z. B. Konfigurationsdateien, die Zugangsdaten für Server etc. beinhalten) werden über die .gitignore von der Git-Verwaltung ausgeschlossen, damit diese nicht einsehbar im Internet landen.

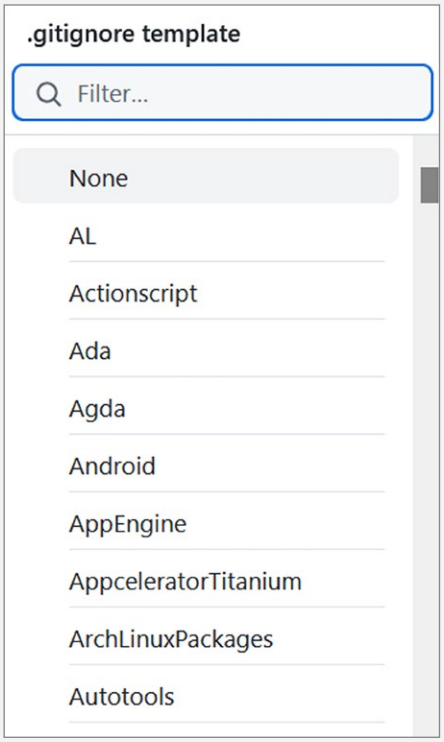
Eine .gitignore, die beispielsweise .temp-Dateien, Log-Dateien und Bilder ignoriert, würde so aussehen:

```
*.temp
Log-*.txt
*.jpg
*.png
```

Das * bedeutet „Hier kann alles stehen". *.temp stellt also ein, dass alle Dateien, die mit der Dateiendung temp gespeichert werden, ignoriert werden.

Bei der Erstellung eines Onlinerepositorys kann man über ein Drop-down-Menü aus einer Reihe vorgefertigter .gitignore-Dateien für verschiedene Programmiersprachen und Programmiertools auswählen.

Pro Branch eines Repositorys kann es nur eine .gitignore geben. Diese kann sich dann aber in unterschiedlichen Branches unterscheiden.




**GitHub stellt verschiedene .gitignore-Templates bereit. Für jedes Projekt gibt es ein passendes.**

und „Lizenz" erklärt. Mit einem Klick auf „Create repository" schließt man den Vorgang ab und wird auf die Seite des neuen, komplett leeren Repositorys geleitet.

## Lokales Repository mit Onlinerepository verbinden

Mit den Informationen aus dem letzten Artikel liegt in diesem Szenario schon ein lokales Repository vor. Es gibt mit Git verwaltete Dateien und auch schon einige Commits – allerdings nur lokal. Um dieses Projekt jetzt 1:1 (wie auf dem lokalen Computer) mit GitHub zu verbinden, geht man wie folgt vor.

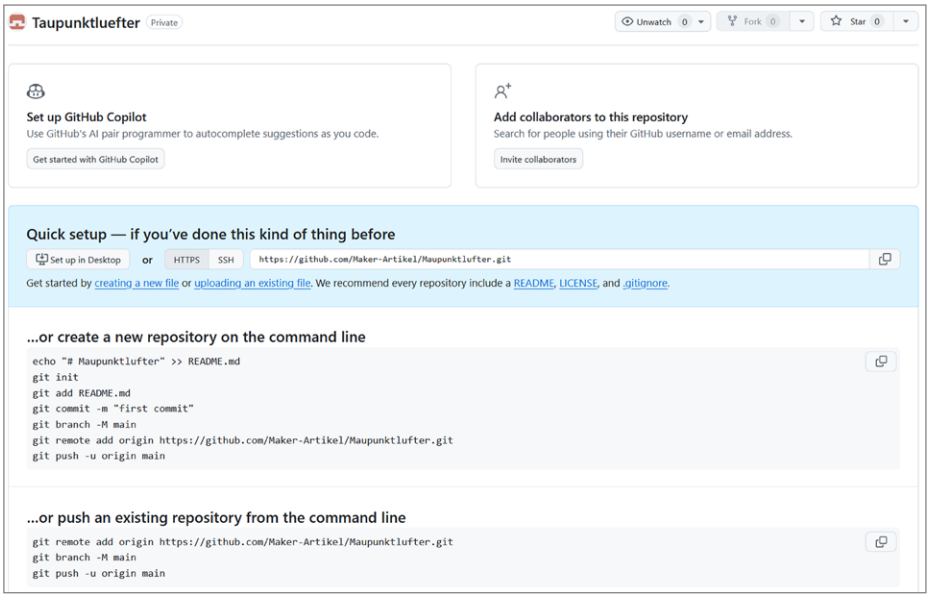
Nachdem man auf GitHub ein komplett leeres Repository angelegt hat – ohne Readme, Lizenz oder .gitignore –, landet man auf einer „Next Steps"-Seite.

Diese bietet drei Möglichkeiten, das Repository jetzt lokal auf den eigenen Computer zu synchronisieren; im vorliegenden Fall mit einem lokal existierenden Projekt ist die letzte Option „push an existing repository from the command line" die richtige, um weiterzumachen.

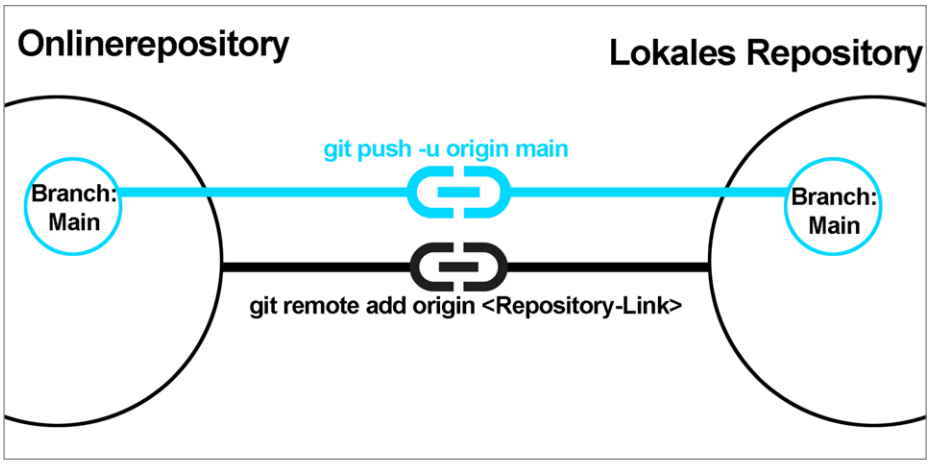
Dort sind drei git-Befehle aufgelistet, mit denen man die beiden Repositories verbindet. Diese Befehle müssen jetzt auf dem Computer in die Kommandozeile eingegeben werden, während man sich im Ordner befindet, in dem das lokale Repository gespeichert ist. Wie man das Terminal öffnet, ist dem ersten Teil dieser Artikelreihe zu entnehmen.

Der erste Befehl `git remote add origin <Repository-Link>` verbindet das lokale Repository mit dem online angelegten. Danach benennt `git branch -M main` den aktuellen Branch des lokalen Repositorys in „main" um. Das ist wichtig, weil viele Workflows auf GitHub davon ausgehen, dass der Haupt-Branch eines Repositorys „main" heißt. Selbst wenn man im Repository keinen neuen Branch erstellt hat, arbeitet man im main-Branch. Denn die Commits werden behandelt, als ob sie einem Branch zugeordnet sind. Eben dem main-Branch.

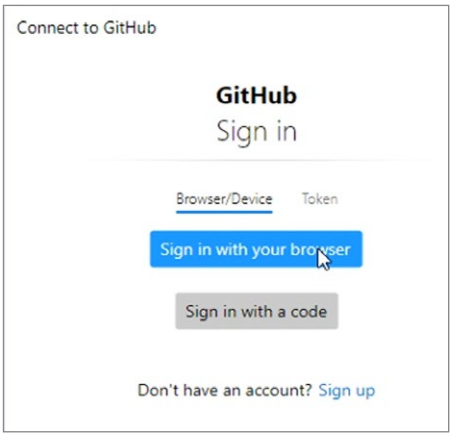
Als Letztes wird mit `git push -u origin main` festgelegt, dass der lokale main-Branch mit dem main-Branch von Origin verknüpft ist, den man mit dem ersten Befehl beim Einrichten des GitHub-Onlinerepositorys erstellt hat. Anschließend lädt man alles aus dem lo-



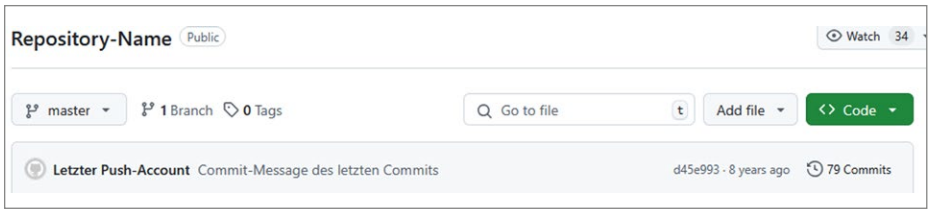

**GitHub zeigt bei einem neuen Repository genau an, wie man damit jetzt weiterarbeiten kann.**




Mit den Befehlen werden erst die Repositories und dann die main-Branches verbunden.




Wenn man das erste Mal über einen Git-Befehl mit GitHub interagiert, muss man sich anmelden.



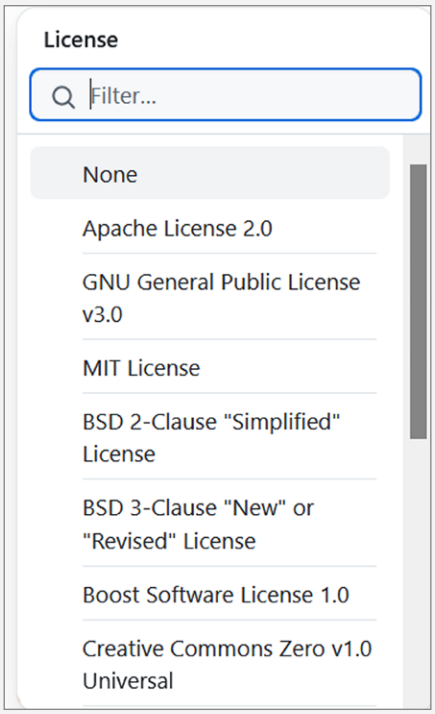

Ein Onlinerepository mit Namen, dem letzten Beitragenden und einer Anzahl an Commits.

## Lizenz

Wenn ein Repository öffentlich ist, kann jeder den darin enthaltenen Code herunterladen. Privat kann er dann damit machen, was er will. In der License-Datei wird allerdings festgelegt, was man damit in der Öffentlichkeit anstellen darf: ob der Code z. B. in anderen Projekten verwendet werden darf, ob dann der Name des Erstellers genannt werden muss (oder nicht) oder ob der Code gar nicht zur Weiterverwendung freigegeben ist.

Auch diese Datei kann man beim Erstellen des Onlinerepositorys aus einer Dropdown-Liste auswählen. Welche Lizenz was erlaubt, muss man individuell nachlesen.

Für Maker, die ihren Code der Community zur Verfügung stellen wollen, sind folgende Lizenzen das Mittel der Wahl: Die GNU General Public License, bei der Code zwar frei genutzt werden darf, das daraus resultierende Projekt allerdings auch unter GNU General Public License Open Source sein muss. Oder Apache 2.0 – auch hier darf Code frei verwendet werden, allerdings kann auf diesen verwendeten Code dann kein Patentanspruch von Dritten erhoben werden.




Mit einer Lizenz legt man den Umfang fest, in dem der eigene Code von Dritten in der Öffentlichkeit verwendet werden darf.

kalen Repository (Dateien und alle Commits aus der Vergangenheit) auf GitHub hoch.

Ist es das erste Mal, dass man mit GitHub auf diese Art und Weise interagiert, muss man sich erst anmelden, um den aktuellen PC als vertrauenswürdig auszuweisen und über die Kommandozeile Git-Interaktionen mit GitHub ausführen zu können.

Es öffnet sich ein „GitHub Sign in"-Fenster. Dort klickt man auf „Sign in with your browser" und meldet sich wie gewohnt auf GitHub an (über die Anmeldemethode mit 2FA). Danach bestätigt man, dass man den Computer authentifizieren will. Jetzt kann man mit dem Computer problemlos mit GitHub interagieren.

Danach reicht der kurze Befehl `git push` ohne weitere Attribute, um Commits hochzuladen.

## Upload und Download

Wenn man ein lokales Repository hat, das mit einem online geführten Repository auf GitHub verbunden ist, kann man lokale Änderungen hochladen und Änderungen aus dem Onlinerepository herunterladen, z. B. wenn man mit jemandem zusammenarbeitet. Siehe dazu Kapitel „Zusammenarbeiten".

Um lokale Änderungen hochzuladen, erstellt man zuerst einen Commit. Danach lädt man diesen Commit mit all seinen Änderungen mit dem Befehl `git push` hoch. Die neuen Änderungen sind sofort online sichtbar.

Ist online etwas verändert worden, lädt der Befehl `git pull` die neuen Änderungen herunter.

Diese Befehle funktionieren zwar ähnlich zu einfachem Hoch- und Herunterladen von Dateien, allerdings wird jeweils 1:1 der aktuelle Projektstand übertragen. Änderungen werden in die lokalen Dateien eingepflegt, neu erstellte Ordner werden ebenfalls angelegt. Es können Dateien verschoben oder gelöscht werden. Man synchronisiert mit diesen Befehlen Veränderungen im Projekt als Ganzes.

## Probleme bei Pull

Beim Ausführen des Befehls `git pull`, um Änderungen in ein lokales Projekt zu übernehmen, kann es passieren, dass noch ungespeicherte lokale Änderungen existieren. Da diese durch den Pull überschrieben würden, entsteht ein sogenannter Merge-Konflikt.

Diesen kann man lösen, indem man entweder einen lokalen Commit erstellt, den man aber nicht mit `git push` hochlädt, sondern dann direkt mit `git pull` die Online-Änderungen herunterlädt, oder indem man sie zwischenspeichert.

Mit dem Befehl `git stash` werden die Änderungen vorübergehend zwischengespeichert. Danach kann man wieder mit `git pull` Änderungen aus dem Internet laden und dann mit `git stash pop` die zwischengespeicherten Änderungen wiederherstellen.

Die Option, einen lokalen Commit anzulegen, ist die eleganteste Lösung. Dabei bleiben Änderungen auf jeden Fall bestehen und man kann jederzeit darauf zugreifen, um sie wieder ins Projekt einzubauen oder anderweitig zu verwenden.

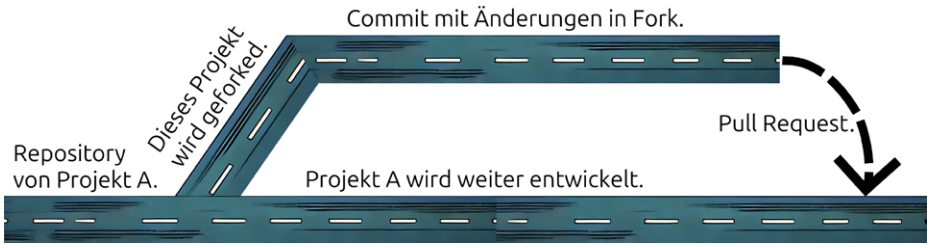
## Zusammenarbeiten

GitHub ist vor allem eine Plattform, die es Entwicklern ermöglicht, gemeinsam an Code zu arbeiten, Änderungen nachzuverfolgen und Feedback auszutauschen.

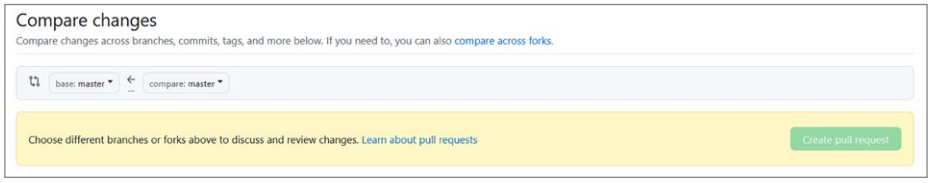
Bei der Zusammenarbeit auf GitHub gibt es zwei Szenarien:

1. Man arbeitet als fest umrissenes Team an einem Projekt und alle können darauf zugreifen und Änderungen vornehmen.
2. Jemand Projektfremdes möchte eine Änderung in das Projekt einbringen und legt seine Anpassungen vor, damit diese geprüft und eventuell in das Projekt aufgenommen werden.

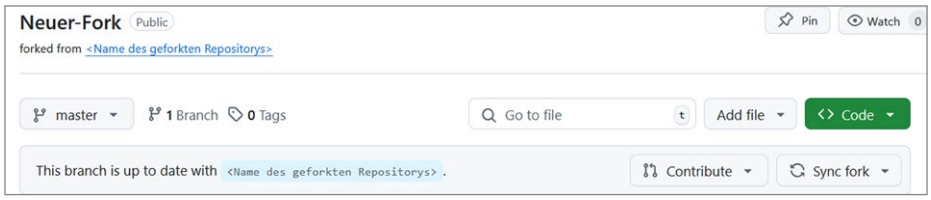
Im ersten Fall fügt man die Mitarbeiter explizit zu einem Repository hinzu, um ihnen bestimmte Rechte zu geben. Dafür öffnet man ein Repository auf GitHub, klickt über dem Repository-Namen ganz rechts auf „Settings“ und dort dann auf „Collaborators/Add




**Ein Fork legt eine Kopie eines Repositorys an. In diese können Änderungen gepusht werden.**

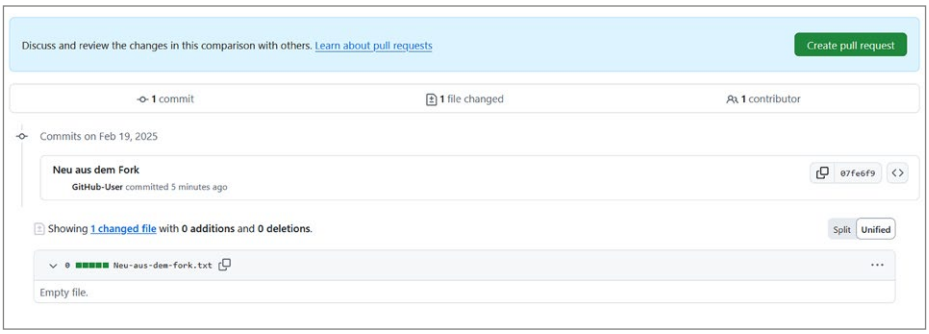



**Bei einem Pull Request kann man Branches und Forks auswählen, aus denen neuer Code in ein Projekt einfließen soll.**




**Das Repository eines Forks hat zusätzlich die Vermerke, von welchem originalen Repository es kopiert wurde und wie viele Änderungen es gab.**

1/3
Rechts




**Die Informationen über den Pull Request enthalten eine Liste der geänderten Dateien.**

People“. Danach öffnet sich ein Eingabefeld, in dem man Mitarbeiter über ihren GitHub-Usernamen oder ihre E-Mail hinzufügen kann.

Hat man Mitarbeiter eingeladen, bekommen diese eine E-Mail, in der sie ihre Teilnahme am Projekt bestätigen. Jetzt können sie ihre Änderungen am Projekt ganz normal mit `git push` hochladen. Zugriffe auf Repository-Optionen (wie beispielsweise eine Namensänderung oder Sichtbarkeitseinstellungen) hat der Account dann nicht.

Im zweiten Szenario möchte jemand, der nicht berechtigt ist, in das Repository zu pushen, etwas zu dem Projekt beitragen. Deshalb muss diese Person zuerst einen Fork (eine Abzweigung/Kopie) des Repositorys anlegen. Das geschieht, indem man auf der Hauptseite eines Repositorys ganz rechts neben dem Namen des Repositorys auf die Schaltfläche „Fork“ und dann auf „Create a new fork“ klickt.

Danach öffnet sich ein Fenster, in dem man den Namen des neuen Forks ändern kann. Unten rechts klickt man zum Bestätigen auf den Button „Create Fork“. Nachdem der Kopiervorgang abgeschlossen ist, erhält man ein neues Repository, in dem alle Commits des kopierten Projektes enthalten sind. Unter dem Namen des Repositorys und über dem Inhalt wird darauf hingewiesen, dass es sich um einen Fork handelt und wie sich dieser vom Original unterscheidet.

Dieses Repository kann man jetzt auf den Computer klonen, Änderungen vornehmen und diese dann in das eigene Repository pushen.

Hat man die Änderungen vorgenommen und in diesen Fork gepusht, geht man zurück auf das Originalrepository und klickt in der Menüleiste unter dem Namen auf „Pull requests“ und dort rechts auf den grünen Button „New Pull Request“. Da hier die Änderungen aus einem anderen Repository kommen, muss man noch auf „compare across forks“ unter der „Compare changes“-Überschrift klicken.

Jetzt wählt man auf der rechten Seite den Fork aus; es werden direkt die Unterschiede angezeigt.

Nach einem Klick auf „Create pull request“ kann man in einem Eingabefeld noch Informationen zu den Änderungen im Projekt geben und dann noch einmal mit „Create pull request“ bestätigen.

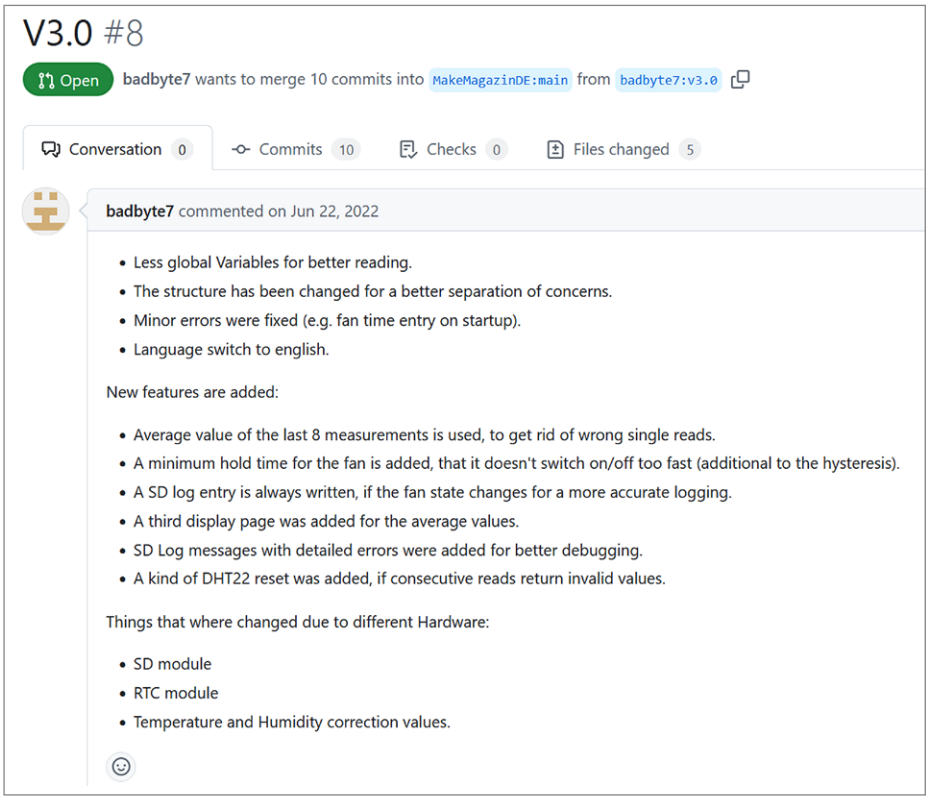
Jetzt ist der Besitzer des Repositorys an der Reihe, in das man mergen möchte. Der Besitzer des Repositorys bekommt eine Nachricht, dass neue Änderungen am Code vorgeschlagen wurden. Diese muss er jetzt überprüfen.

Dafür geht man als ursprünglicher Ersteller des Projekts auf GitHub bei dem Repository wieder auf die „Pull requests“-Seite, auf der alle Vorschläge aufgelistet sind. Klickt man auf einen, bekommt man eine neue Übersicht.

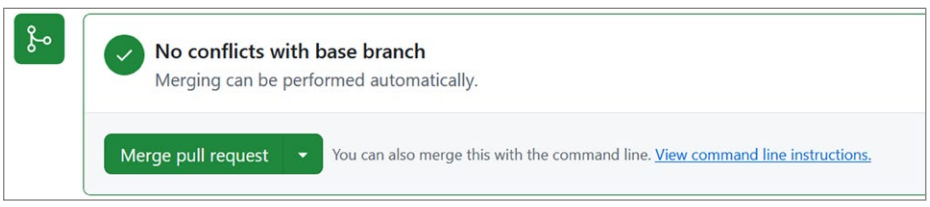
Im „Commit“-Tab werden auch die vorgenommenen Änderungen genau angezeigt. Wenn man in dieser Übersicht direkt erkennt, dass der vorgeschlagene Code einen Mehrwert hat, kann man diesen im Browser mit seinem mergen.

Über die Verlinkungen unter der Pull-Request-Überschrift kommt man auch zum Repository des Forks. Das ist wichtig, weil man dort das ganze Projekt herunterladen kann. Dann kann man überprüfen, ob die Änderungen funktionieren, sich das Projekt noch kompilieren lässt etc. Ist alles okay und man möchte die Änderungen in das Hauptprojekt übernehmen, klickt man im Pull Request auf „Merge pull request“.

Das erzeugt einen neuen Commit und kann dadurch auch wieder rückgängig gemacht




**Die Beschreibung eines Pull Requests sollte so detailliert wie möglich sein. Hier ein Beispiel.**




**Einen Pull Request kann man direkt im Browser in das Projekt einfügen.**

## Klonen

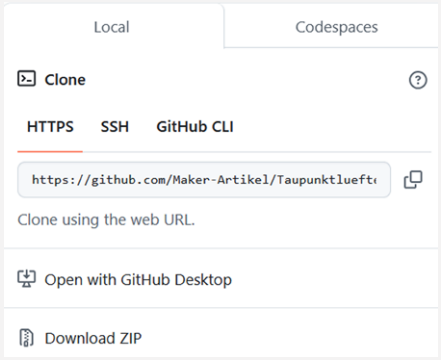
Um ein Repository lokal auf den Computer zu bekommen, muss man es klonen, also das Projekt inklusive aller Commits, Branches etc. und der Verbindung zum Onlinerepository herunterladen. Das passiert über den Konsolenbefehl `git clone <Onlinerepository-referenz>`. Die Referenz bekommt man auf der Repository-Seite über den grünen Button, auf dem „<> Code“ steht.

Dieser Button öffnet ein Menü, das uns drei Optionen zur Verfügung stellt: Klonen, mit der grafischen Oberfläche „GitHub Desktop“ öffnen oder Download als ZIP.

Für diesen Artikel wird wieder die Konsole genutzt. Für die Verwendung des grafischen GitHub-Clients gibt es online eine Anleitung. Diese ist in der Kurzinfo verlinkt.

Ein Repository lässt sich über drei Wege klonen: über HTTPS, SSH oder über GitHub CLI (Command Line Interface, Kommandozeile). Klonen über HTTPS und SSH ist bei allen Git-Services identisch nutzbar und basiert auf Git-Basisbefehlen. GitHub CLI ist ein spezielles Programm, das nur für GitHub funktioniert. Da es nicht standardisiert ist, liegt der Fokus in diesem Artikel auf HTTPS und SSH (siehe Kasten „SSH“).

Um über HTTPS zu klonen, kopiert man die URL, die unter HTTPS angezeigt wird, und fügt sie im Terminal hinter `git clone` ein. Wenn man jetzt mit Enter bestätigt, wird im aktuellen Ordner, in dem das Terminal geöffnet ist, ein neuer Ordner mit dem Namen des Repositorys angelegt und in diesen dann die Inhalte des Projektes heruntergeladen. Hat man aber schon einen Extraordner angelegt, kann man mit `git clone <Onlinerepository-referenz> .` (ein Leerzeichen und ein Punkt nach dem Befehl) direkt in den aktuellen Ordner speichern.




**Die verschiedenen Arten, Code herunterzuladen.**

Hat man ein volles Repository geklont, hat man jetzt alle Dateien, Commits und Branches, die online bereitgestellt wurden, lokal auf dem Computer. Das Projekt ist weiterhin mit der Online-Version verbunden. Wenn dort jetzt etwas Neues hochgeladen wird, kann man mit dem Befehl `git pull` diese Änderungen herunterladen.

Ein öffentliches Repository kann man immer klonen.

werden, sollten sich doch Probleme mit dem neuen Code ergeben.

Solche Pull Requests können auch gemacht werden, um Branches in den Haupt-Branch einzufügen. Das funktioniert genauso wie für Forks. Generell ist dieses Werkzeug dafür gedacht, dass parallel entwickelter Code in ein Hauptprojekt eingefügt wird, aber vorher noch vom Besitzer des Repositorys oder anderen Projektmitgliedern untersucht werden kann. Diese Überprüfung ist bei großen Open-Source-Projekten sehr wichtig. Dort gibt es immer wieder Versuche, schädlichen Code einzufügen. Aber sie bietet auch eine gute Möglichkeit, sich von anderen Programmierern den einen oder anderen Trick oder eine Herangehensweise abzuschauen.

Für kleine Pull Requests, in denen wirklich nur einzelne Codezeilen angepasst werden, ist das wirklich ein sehr einfacher Vorgang, bei dem man oft nicht einmal den Browser verlassen muss. Für große Änderungen sollte man sich aber immer das ganze geänderte Projekt anschauen und auf Funktionalität prüfen. Da kommt man um einen Download des Forks nicht herum.

Pull Requests werden über den Browser verwaltet, nicht über die Kommandozeile wie andere Git-Operationen.

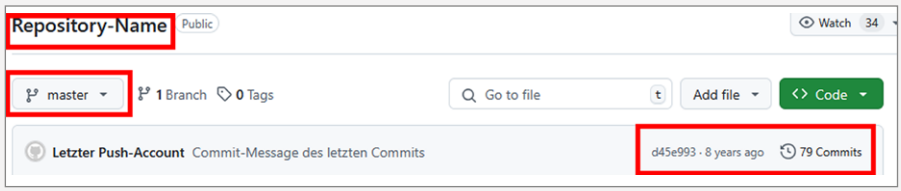
Mit diesem Wissen kann man mit Git nicht nur lokal seine Projektversionen verwalten, sondern auch auf ein GitHub-Repository hochladen. Dort stehen einem jetzt alle Möglichkeiten der Kollaboration offen. Andere können nun mit diesem Code ihre eigenen Projekte verwirklichen oder eigene Verbesserungen einbringen. Entweder als Teammitglied oder als dritte Partei, mit frischem Blick von außen.

Online gibt es einen Zusatzartikel, der die hier gezeigten Befehle anhand der grafischen GitHub-Desktop-Oberfläche zeigt. Er ist in der Kurzinfo verlinkt. *—das*

## Das Onlinerepository

Bei einem Repository auf GitHub steht oben links immer der vergebene Name. Darunter befindet sich ein Drop-down-Menü, mit dem man durch die verschiedenen vorhandenen Branches schalten kann.

Darunter befindet sich dann das Dateifenster, in dem alle Dateien im Projekt angezeigt werden. Dieses Fenster verfügt über eine Kopfzeile, in der links der Name des Accounts steht, der als Letztes etwas zum Projekt gepusht hat. Direkt daneben steht der Text der letzten Commit-Message. Ganz rechts findet man die Commit-ID.




**Von links oben nach rechts unten: Repository-Name, ausgewählter Branch, ID des aktuellen Commits und Commitanzahl.**

Wenn man auf diese klickt, wird einem angezeigt, was im Repository mit diesem Commit verändert wurde. Und daneben ist ein Zähler aller Commits. Klickt man dorthin, bekommt man eine Liste aller Beiträge und kann sich anschauen, was diese Commits hinzugefügt und geändert haben.

## SSH

SSH (Secure Shell) bei GitHub ist eine Möglichkeit, sicher mit Git-Repositories zu arbeiten, ohne jedes Mal Benutzernamen und Passwort eingeben zu müssen. Das Konzept dahinter basiert auf zwei Dateien, dem sogenannten Schlüsselpaar: einem privaten und einem öffentlichen Schlüssel. Der private Schlüssel bleibt sicher auf dem eigenen Computer gespeichert, der öffentliche Schlüssel wird bei GitHub hinterlegt. Wenn man sich per SSH verbindet, überprüft der Server, ob der öffentliche Schlüssel zum privaten Schlüssel passt. Dazu sendet der Server eine verschlüsselte Nachricht, die nur der passende private Schlüssel entschlüsseln kann. Gelingt das, weiß der Server, dass der richtige Benutzer zugreifen möchte, und gewährt den Zugang. Das Ganze passiert im Hintergrund, sodass man keine Passwörter mehr eingeben muss, sondern direkt Zugriff erhält.

Da der private Schlüssel niemals übertragen wird und die Authentifizierung ohne Passworteingabe (was sonst auch übertragen werden würde) erfolgt, schützt SSH vor Phishingangriffen und Brute-Force-Versuchen, weil der private Schlüssel nie den eigenen PC verlässt und dadurch nicht abgefangen werden kann.

### Klonen mit SSH

Um ein Repository über SSH zu klonen, braucht man als Erstes das Schlüsselpaar. Das erzeugt man im Terminal mit dem Befehl `ssh-keygen -t ed25519 -C <E-Mail>`.

Danach wird man gefragt, ob man den Speicherort der Schlüssel ändern möchte. Das lässt man auf der Standardeinstellung und bestätigt einfach mit Enter. Dann wird man nach einer Passphrase für die Schlüssel gefragt. Hier kann man entweder ein Passwort setzen oder auch einfach mit Enter bestätigen, um keins zu setzen.

Danach werden die Schlüssel generiert und unter C:\Users\<Nutzername>\.ssh\ gespeichert. Dort befinden sich jetzt zwei Dateien mit dem Namen id_ed25519. Eine PUB-Datei und eine ohne Endung. Die ohne Endung ist der sogenannte Private Key. Der bleibt immer auf dem Rechner. Die PUB-Datei ist der Public Key. Diese Datei bzw. ihren Inhalt hinterlegt man jetzt auf GitHub. Dafür öffnet man die PUB-Datei mit dem Windows-Editor und kopiert den Inhalt.

Auf GitHub klickt man oben rechts auf den Account-Avatar und dann auf „Settings/SSH and GPG keys“. Auf dieser Optionsseite gibt es oben rechts den Button „New SSH key“. Auf den klickt man und kann dem Schlüssel danach einen Namen geben. Das sollte der Name des Computers sein, mit dem man diesen Schlüssel nutzt. Im Feld „key“ fügt man jetzt den kopierten Schlüsselinhalt ein. Als Letztes klickt man auf „Add SSH key“.

Jetzt kann man beim Klonen auf „SSH“ klicken, den dortigen SSH-Link kopieren und für den git-clone-Befehl nutzen. Anmelden mit Passwort fällt jetzt weg.

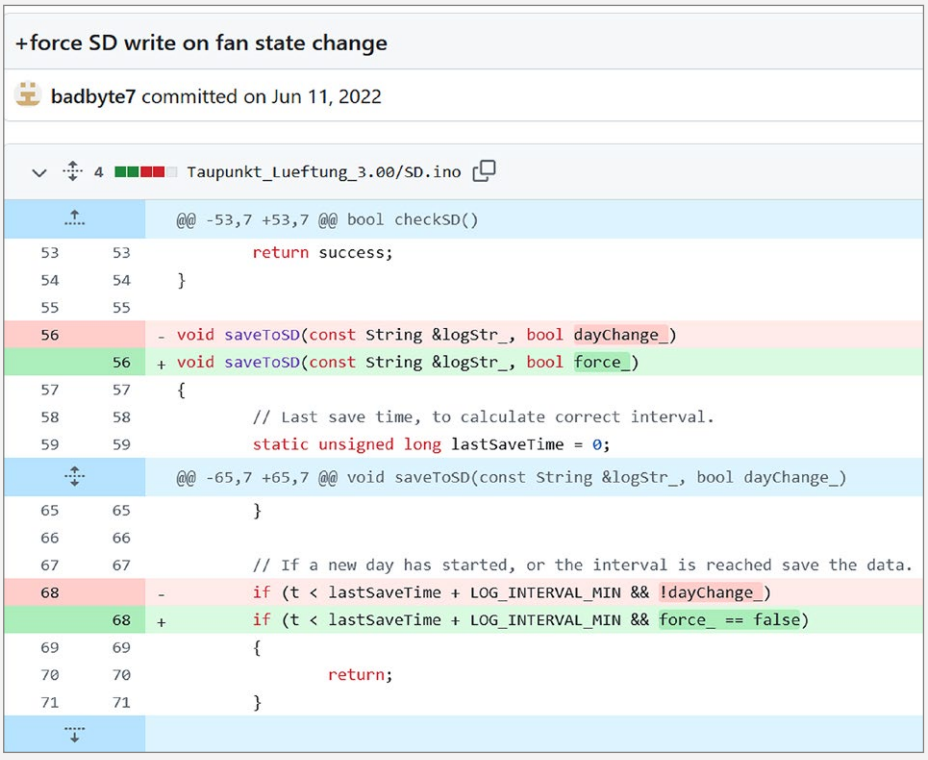
## Vergleich von Codeversionen

Im Text wird immer wieder davon gesprochen, dass man sich auf GitHub die Unterschiede im Code zwischen Commits ansehen kann. Das sieht wie folgt aus:

Wenn eine Datei verändert wird, werden die Stellen mit neuen Inhalten farblich codiert. Rote Codezeilen sind Stellen, in denen etwas Altes verändert wurde. Die Zeile wird gelöscht und ersetzt. Der neue Inhalt wird grün angezeigt.

Diese Ansicht ist nützlich, um bereits bestehende Commits in einem Repository zu vergleichen und herauszufinden, wo genau Änderungen stattgefunden haben. Zum Beispiel könnte seit dem letzten Commit bei der Ausführung des Codes ein Bug auftreten, der vorher nicht da war. Mit dieser Anzeige sieht man sofort, welche Stellen verändert wurden und wo der Bug verortet sein kann.

Auch bei Pull Requests ist diese Ansicht wichtig. Dadurch kann man direkt alle Stellen überprüfen, die verändert wurden – und einen Code sowohl auf Programmierfehler als auch auf eventuell eingefügten Schadcode kontrollieren.




**Vergleich aus einem Pull Request. Die geänderten Zeilen sind hervorgehoben.**