KAPITEL 3

Die Basis: Das erste eigene GitHub-Projekt

Was brauchst du alles für den Einstieg mit GitHub? Einen funktionstüchtigen Rechner mit einem der Betriebssysteme Windows, Linux oder macOS, darüber hinaus eine stabile Internetanbindung und einen Internetbrowser. Ansonsten solltest du Folgendes tun:

- Dir einen kostenlosen GitHub-Account zulegen (wie, beschreibe ich im folgenden Abschnitt »Account anlegen« auf Seite 20).
- Falls du mit Git arbeiten willst: Git auf deinem Rechner installieren (eine entsprechende Anleitung findest du in Kapitel 7, Abschnitt »Git installieren und einrichten« auf Seite 141).

Solltest du bereits einen Account bei GitHub eingerichtet haben, kannst du dieses Kapitel gegebenenfalls überspringen. Vielleicht schaust du aber trotzdem kurz in die Abschnitte »Account schützen« auf Seite 22 und »Unsichtbar werden – die eigene Mailadresse schützen« auf Seite 23, falls du dich mit dem Schutz deines Accounts oder deiner Mailadresse noch nicht befasst hast.

Am Ende des Kapitels kannst du ...

- auf GitHub einen Account anlegen und diesen sowie deine Mailadresse schützen.
- Repositories anlegen und Änderungen an diesen vornehmen.
- Issues anlegen, klassifizieren, jemandem zuweisen und schließen.
- ein Repository löschen.
- ein lokales Projekt auf GitHub hochladen.

Falls du dir noch nicht sicher bist, ob du Git nutzen möchtest, kannst du die Installation auch später nachholen, sobald du im entsprechenden Kapitel angekommen bist. Für große Teile des Buchs werden wir Git nicht brauchen. Und wer weiß, vielleicht reichen dir ja auch die Funktionen von GitHub?

Account anlegen

Einen Account bei GitHub einzurichten, ist relativ einfach. Dafür klickst du im Menü von GitHub[1] auf den Button *Sign up* (deutsch »sich registrieren«) und füllst die geforderten Informationen entsprechend aus (siehe auch Abbildung 3-1).

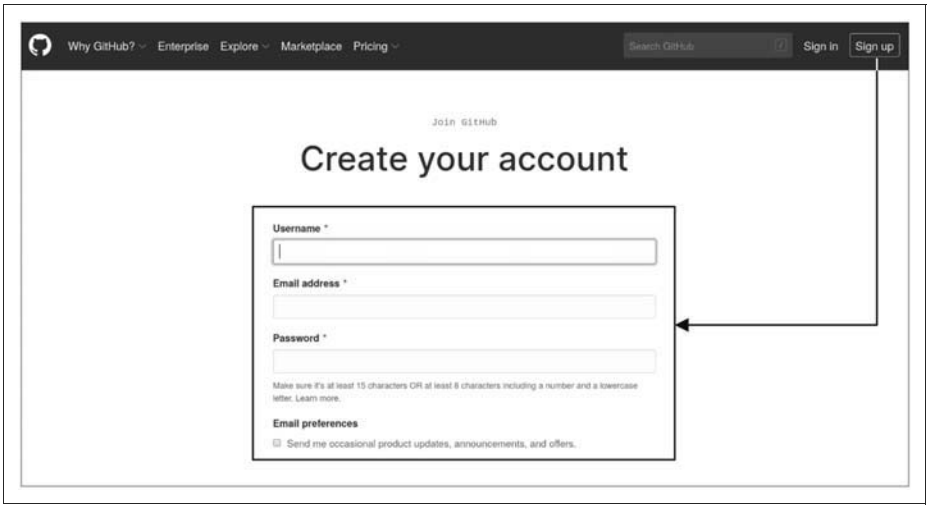

Abbildung 3-1: Sich bei GitHub zu registrieren, ist einfach: Username wählen, eigene Mailadresse eingeben und ein Passwort vergeben – fertig.

Sobald du eingeloggt bist, ändert sich das GitHub-Menü etwas (siehe Abbildung 3-2).

- Über ❶ kommst du immer zurück auf die Startseite von GitHub.
- In ❷ befindet sich die Suchfunktion, mit der du nach Repositories, aber auch innerhalb eines Repositorys suchen kannst.
- Die Punkte unter ❸ sind zum einen Schnellzugriffe auf die selbst erstellten *Pull requests* und *Issues* (die du in diesem und Kapitel 4 noch näher kennenlernst). Der Punkt *Marketplace* ist für das Erweitern des eigenen Repositorys gedacht (siehe Kapitel 9), und *Explore* hilft uns beim Finden anderer Projekte (siehe Kapitel 6).
- Bei ❹ steht der Schnellzugriff bereit, um beispielsweise ein neues Repository anzulegen.
- In ❺ findest du alles Relevante für deinen Account, z.B. deine Repositories oder die Möglichkeit, dich auszuloggen.

1 *https://github.com*

Abbildung 3-2: Nach dem Log-in verändert sich das GitHub-Menü etwas.

Standardprofilbild – Identicon

Vielleicht ist dir nach einem Log-in in GitHub ein merkwürdiges kleines pixeliges Bild oben rechts auf der Seite aufgefallen (in Abbildung 3-2 bei Punkt ❺), das so ähnlich aussieht wie eines der drei in Abbildung 3-3. Das ist ein sogenanntes *Identicon* – ein Kofferwort[2] aus »Identification« und »Icon« (zu Deutsch »Identifikation« und »Symbol«) –, und du siehst hier dein Profilbild.

Abbildung 3-3: Identicons sind das Standardprofilbild bei GitHub. (Bildquelle: https://github.blog/2013-08-14-identicons)

GitHub generiert diese Bilder automatisch in Abhängigkeit von deiner Benutzer-ID. Es gibt also keine Möglichkeit, die Farbe oder das Muster anzupassen. Du kannst das Bild aber jederzeit durch ein eigenes ersetzen (und auch wieder zum Identicon zurückkehren).

Tipp: Passwortsafe

An dieser Stelle möchte ich für den Einsatz eines Passwortsafes werben, der nicht nur alle Accounts und Passwörter sicher aufbewahrt, sondern häufig auch einen Passwortgenerator für sichere Passwörter mitbringt.

Ich persönlich nutze KeePassXC[3], das viele Erweiterungsmöglichkeiten bietet. Beispielsweise gibt es für den Browser eine Erweiterung, mit der ich Zugangsdaten aus dem Safe direkt in die Eingabemaske einer Webseite »beamen« kann. Und mit der Smartphone-App *Keepass2Android* habe ich meine Zugangsdaten auch mobil verfügbar, falls ich sie brauche.

2 Verschmelzung zweier Wörter zu einer neuen Bedeutung, z. B. breakfast + lunch = brunch.

3 *https://keepassxc.org/*

Account schützen

Eine Sache, die du einrichten kannst, um die Sicherheit deines Accounts zu erhöhen, ist die 2-Faktor-Authentifizierung. Du findest sie in den *Settings* deines Profils unter dem Punkt *Account security* (siehe auch Abbildung 3-4). Folge den Anweisungen dort, um diese Form der Authentifizierung einzurichten. Du hast die Möglichkeit, entweder eine Authenticator-App als zweiten Faktor zu wählen oder ein Verfahren über SMS. GitHub empfiehlt als Authenticator-App für das Smartphone *Authy*[4], *1Password*[5] oder *LastPass Authenticator*[6]. Jedes Mal, wenn du dich einloggst, wird GitHub dich auffordern, entweder via App oder SMS – je nachdem, was du eingerichtet hast – einen Code einzugeben, bevor du auf deinen Account zugreifen kannst.

Wiederherstellungsoptionen bei der 2-Faktor-Authentifizierung

Sobald die 2-Faktor-Authentifizierung eingerichtet ist, solltest du auf jeden Fall eine der von GitHub angebotenen Wiederherstellungsoptionen (*Recovery Options*) einrichten. Falls dein Smartphone verloren geht, hast du über diese Optionen die Möglichkeit, wieder Zugriff auf deinen GitHub-Account zu bekommen. GitHub bietet als Optionen an:

- **Recovery codes**: Eine Reihe an Codes, die bei Verlust des Smartphones zur Wiederherstellung genutzt werden können (am besten in einem Passwortsafe abspeichern).
- **Fallback SMS number**: Eine alternative Rufnummer, an die GitHub im Fall des Smartphone-Verlusts die Recovery Codes schickt.
- **Recovery tokens**: Die Möglichkeit, über Facebook wieder Zugriff auf den GitHub-Account zu bekommen.

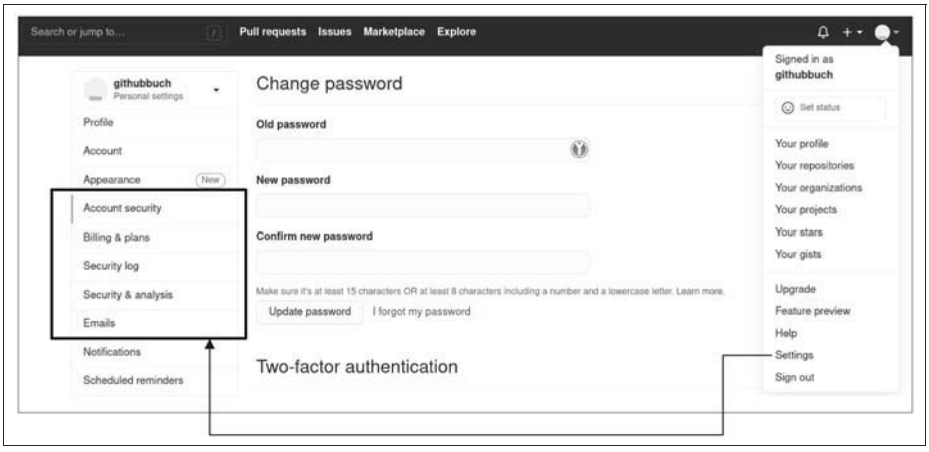

Abbildung 3-4: Im Profil kommst du über »Settings« zu den relevanten Einstellungsmöglichkeiten.

4 *https://authy.com/guides/github/*

5 *https://support.1password.com/one-time-passwords/*

6 *https://support.logmeininc.com/lastpass/help/lastpass-authenticator-lp030014*

2-Faktor-Authentifizierung

Die 2-Faktor-Authentifizierung – auch 2FA genannt – ist eine zusätzliche Sicherungsebene, um einen Account vor unbefugtem Zugriff zu schützen. Wenn ich Zugriff auf meinen Account haben will, benötige ich mindestens zwei verschiedene »Faktoren«, um mich überhaupt einloggen zu können. Faktoren können sein:

- Etwas, das ich kenne: Passwort, PIN-Code, spezielles Muster etc.
- Etwas, das ich habe: Smartphone, Kreditkarte, Hardwaretoken etc.
- Etwas, das ich bin: Fingerabdruck, Irisscan, Stimmerkennung etc.

Sofern ein Faktor abhandenkommt – beispielsweise wenn eine fremde Person mein Passwort herausgefunden hat –, erhält sie trotzdem noch keinen Zugriff auf meinen Account, solange sie nicht auch den zweiten Faktor kennt.

Manchmal liest man auch den Begriff »MFA«, was für *Multi-Faktor-Authentifizierung* steht. Das bedeutet, dass ich mehr als einen Faktor nutze. 2FA ist also eine Untermenge von MFA.

Unter dem Menüpunkt *Security log* werden alle für die Sicherheit deines Accounts relevanten Ereignisse gespeichert, beispielsweise wann du dich eingeloggt hast, von wo aus und mit welcher IP-Adresse[7]. Das *Security log* kannst du ebenfalls nutzen, um dir andere Ereignisse – nicht nur solche, die deinen Account betreffen – anzeigen zu lassen. Gibst du `operation:create` in das vorhandene Suchfeld ein, werden alle Ereignisse angezeigt, an denen du etwas Neues erstellt hast, etwa ein neues Repository. Mit `created:2020-02-18` kannst du dir alles anzeigen lassen, was du am 18. Februar 2020 erstellt hast.[8]

Unsichtbar werden – die eigene Mailadresse schützen

Eine Sache, die du beachten solltest, betrifft deine Mailadresse, die du beim Einrichten deines Accounts angibst. Falls du es nicht explizit änderst, wird deine Mailadresse auf deiner Profilseite angezeigt und auch für Aktivitäten innerhalb von GitHub verwendet.[9] Manche möchten das nicht, deswegen bietet GitHub dir die Möglichkeit, etwas anonymer unterwegs zu sein. In deinem Profil im Menü *Settings* unter dem Punkt *Emails* kannst du festlegen, dass GitHub deine Mailadresse privat behandelt (siehe Abbildung 3-4 und Abbildung 3-5).

7 Das ist die »Postanschrift« deines Rechners.

8 Eine ausführliche Auflistung dessen, was noch alles geht, findest du unter *https://help.github.com/en/github/authenticating-to-github/reviewing-your-security-log*.

9 Sogenannte webbasierte Git-Operationen, das wird verständlicher, sobald wir uns mit Git in Kapitel 7 beschäftigen.

Abbildung 3-5: GitHub bietet die Möglichkeit, deine Mailadresse geheim zu halten – meine anonyme Mailadresse habe ich hier noch einmal zusätzlich durch Verpixeln anonymisiert.[10]

Wie du auf dem Bild sehen kannst, bekommst du eine Art »Alternativ-E-Mail-Adresse« angezeigt, die aus einer siebenstelligen Identifikationsnummer (ID) und deinem Usernamen erzeugt wird: *ID+username@users.noreply.github.com* (beispielsweise 1234567+geheimniskraemer@users.noreply.github.com). Und wofür brauche ich diese Mailadresse? In »Git installieren und einrichten« auf Seite 141 zeige ich dir später einen Anwendungsfall, im Moment nehmen wir erst einmal hin, dass es diese anonyme Mailadresse gibt.

Des Weiteren siehst du auf dem Bild, dass du auch die Option hast, etwas zu blockieren. Diese Einstellung ist nur relevant, wenn du vorhast, mit Git später zu arbeiten.

Man kann bei GitHub auch noch eine Menge mehr einstellen, wie beispielsweise ein Profilbild hochladen oder Ähnliches. Da uns das aber nicht dabei hilft, die grundsätzlichen Funktionen von GitHub zu verstehen, gehe ich hier nicht näher darauf ein. Aber natürlich kannst du dir die weiteren Einstellungen investigativ erschließen.

Das erste eigene Repository anlegen

Da du jetzt startklar bist, werden wir ab diesem Abschnitt ein erstes eigenes Projekt anlegen und Veränderungen an den dort vorhandenen Dateien vornehmen. Wir lernen *Commit* und *Issues* genauer kennen, die die Basis für viele andere Aktivitäten bilden.

Für das Anlegen eines eigenen Projekts gibt es mehrere Möglichkeiten. Wir wenden uns mit dem Plussymbol oben rechts in der Ecke im GitHub-Menü zunächst der offensichtlichsten zu (siehe auch Abbildung 3-6) und wählen *New repository*.

10 Immer wenn du in den Screenshots irgendwelche verwischten Spuren siehst, ist das mein Accountname. Ich habe für die Screenshots in diesem Buch den Account so zugemüllt, dass er mir ein wenig peinlich ist und ich ihn hier daher nicht präsentieren möchte :).

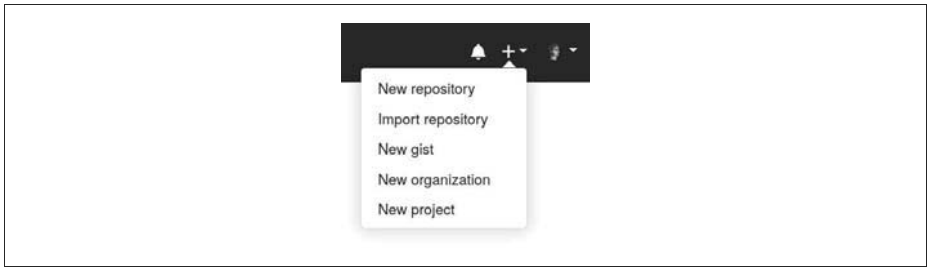

Abbildung 3-6: Ein neues Repository kann man unter anderem über das Plussymbol oben rechts erzeugen.

Tipp: Shortcut für neue Repositories

Eine weitere Möglichkeit, Repositories anzulegen, bietet die Webadresse *https://repo.new* im Browser. Dann bist du direkt auf der GitHub-Seite für neue Repositories. Das funktioniert natürlich nur, wenn du bereits eingeloggt bist, ansonsten landest du erst mal auf der Log-in-Seite.

Nun werden wir aufgefordert, eine Reihe von Angaben zu machen (siehe Abbildung 3-7). Gib deinem Projekt einen sprechenden Namen – das kann auch so etwas Originelles wie »testprojekt« sein – und, wenn du magst, eine entsprechende Beschreibung. Alles lässt sich später noch ändern.

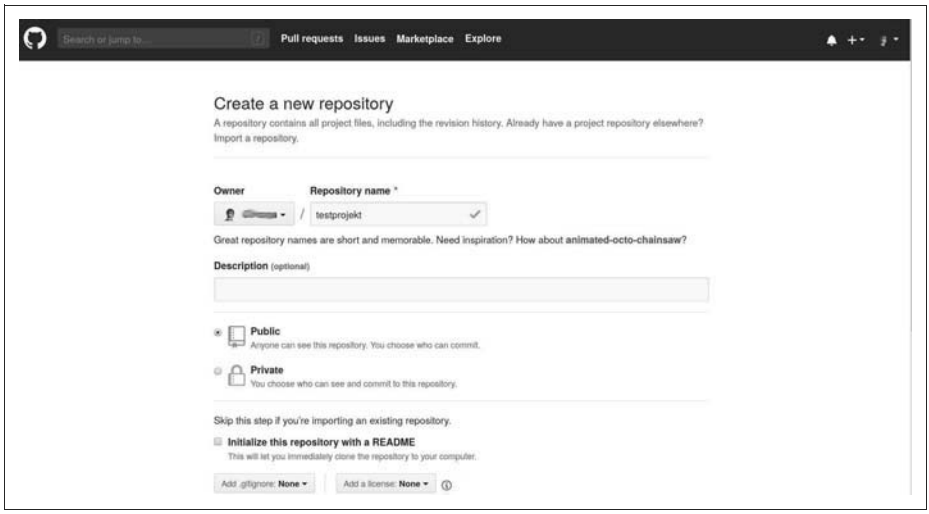

Abbildung 3-7: Ein neues Projekt/Repository ist mit wenigen Klicks erstellt.

Beim Erstellungsprozess wirst du auch gefragt, ob du ein öffentliches (*Public*) oder ein privates (*Private*) Repository anlegen möchtest. Öffentliche Repositories können später von jedem beliebigen Besucher angesehen, aber nur mit Erlaubnis verändert werden. Auf private Repositories hast dagegen ausschließlich du Zugriff. Wenn du für ein Repository einen *Collaborator* festlegst, kann dieser Veränderun-

gen vornehmen und natürlich auch alles sehen, selbst wenn das Repo als privat eingestellt ist. Wir sind hier mal mutig und wählen *Public* aus.

Tipp: Test-Repositories anlegen
Wenn du dich erst einmal nur mit GitHub und dessen Funktionen vertraut machen möchtest, empfiehlt es sich, ein oder mehrere Repositories zum Testen und Rumspielen anzulegen. Diese kannst du nach deinen Experimenten wieder löschen (siehe Abschnitt »Ein bestehendes Repository löschen« auf Seite 39 weiter unten in diesem Kapitel).

Zudem wirst du gefragt, ob GitHub für dich eine *README.md* anlegen soll (wir erinnern uns an Kapitel 2, Abschnitt »Informationen finden (interessierte Anwenderin)« auf Seite 13). Da das generell eine gute Idee ist, machen wir das natürlich. Alles andere kannst du erst einmal so lassen, wie es ist. Sobald du auf *Create Repository* geklickt hast, erstellt GitHub automatisch ein Repository mit einer noch recht leeren *README.md* und allen Möglichkeiten, die GitHub so bietet – beispielsweise dem Anlegen von Issues.

Wichtig zu wissen: Wir werden im Verlauf des Buchs ausschließlich auf public Repositories arbeiten. Sollte irgendeine Funktion, die ich dir vorstelle, nicht funktionieren, könnte es eventuell daran liegen, dass dein Repository privat ist. GitHub stellt viele Features für öffentliche Repos kostenlos zur Verfügung. Will man die gleichen Features in privaten Repos nutzen, funktioniert das nur nach Einwurf kleiner Münzen.

Eine inhaltliche Änderung am Projekt vornehmen

Als Erstes befüllen wir die *README.md*. Gehe dazu auf <> *Code*, falls du nicht schon dort bist. Auf der rechten Seite des Screenshots siehst du ein Stiftsymbol (siehe Abbildung 3-8). Das Symbol zeigt generell an, dass Dateien editiert werden können. Über das Stiftsymbol gelangst du in einen Texteditor, in dem du die *README.md* mit Inhalt füllen kannst.

Wie du in Abbildung 3-9 sehen kannst, bietet der Texteditor zwei Optionen an: *Edit file* (Datei editieren) und *Preview changes* (die Vorschau der Änderungen). Standardmäßig bist du im Modus *Edit file*, in dem du deine Änderungen machen kannst. Der Modus *Preview changes* bietet dir eine Vorschau der aktuellen Änderungen an, ohne dass du sie abspeichern musst. Dann lass uns die Datei mal mit Leben füllen. Wie bereits erwähnt, kannst du für die Formatierung der Datei das sogenannte *Markdown* verwenden. Du könntest zum Beispiel Folgendes eintippen:[11]

11 Nur der Vollständigkeit halber: Hierbei handelt es sich um »GitHub Flavored Markdown«, also Markdown, das durch GitHub etwas angereichert wurde. Beispielsweise gehört die Aufgabenliste dazu. Weitere Details findest du hier: *https://guides.github.com/features/mastering-markdown/*.

```
# Mein wunderbares Projekt
Dies hier ist mein **erstes Projekt**, um *GitHub* auszuprobieren.

## Meine To-dos:
- [x] README.md befüllen
- [ ] Andere Dinge ...
```

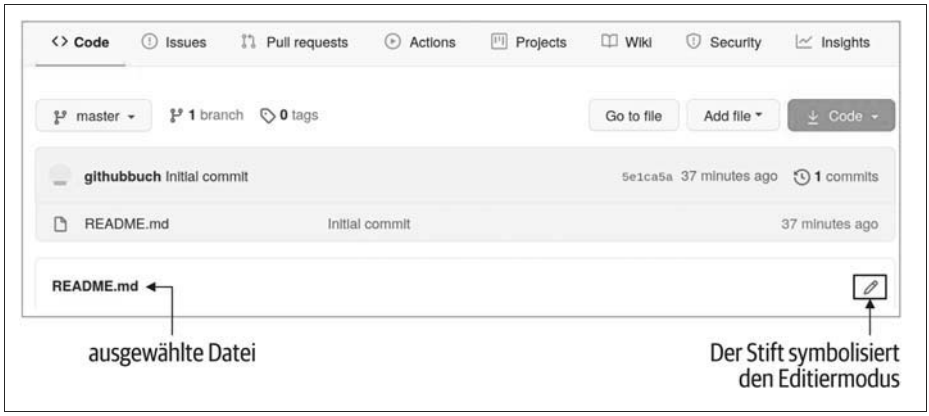

Abbildung 3-8: Durch Anwählen des Stiftsymbols kann man die Datei in einem Editor bearbeiten.

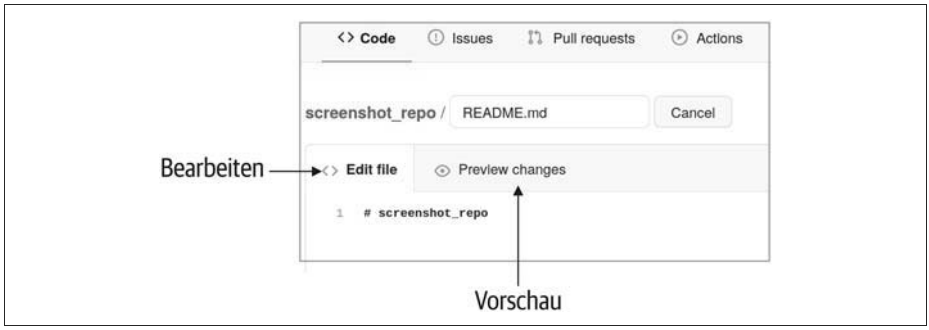

Abbildung 3-9: GitHub bietet einen Texteditor mit den Modi »Bearbeiten« und »Vorschau« an.

Spiel ruhig ein bisschen damit herum, Anregungen findest du in der Erklärbärbox »README.md« in Kapitel 2.

Über die Vorschaufunktion kannst du jederzeit überprüfen, ob das Ganze so aussieht, wie du es dir vorgestellt hast. Wenn du zufrieden bist mit deinem Werk, möchtest du es natürlich abspeichern. Die Möglichkeit dazu findest du, wenn du nach unten scrollst (siehe Abbildung 3-10). Aber hoppla, da steht ja gar nicht speichern, da steht irgendwas von »Commit« und »Branch«. Wir merken uns erst einmal: *Commit* ist so etwas wie *speichern* und fügt die Änderungen zum Projekt hinzu. Später, wenn wir mit Git arbeiten werden (siehe Kapitel 7), werden wir

sehen, dass das nicht ganz korrekt ist. Für den Augenblick reicht es aber, die beiden Begriffe gleichzusetzen.

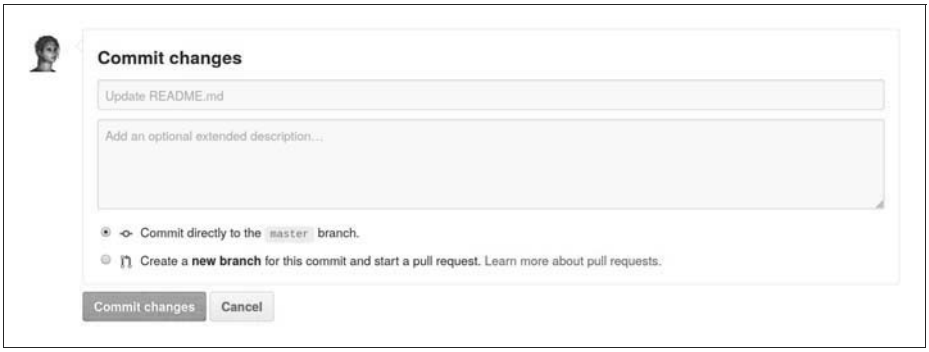

Abbildung 3-10: Abspeichern von Änderungen nennt man bei GitHub »Commit«.

Was den Branch anbelangt, lassen wir zunächst die Standardeinstellung stehen (*Commit directly to the master branch.,* zu Deutsch etwa »direkt auf dem Hauptzweig speichern«). Wir werden später sehen, was es damit auf sich hat (siehe Kapitel 4). Was du noch befüllen solltest, ist die Commit-Nachricht (die Textfelder). Netterweise macht uns GitHub bereits einen Vorschlag für den Titel der Commit-Nachricht, den du erst einmal so übernehmen kannst.

Commit-Nachrichten

Commit-Nachrichten bei GitHub bestehen aus zwei Teilen: dem Titel und einer (optionalen) detaillierteren Beschreibung. Den Titel solltest du immer aussagekräftig befüllen, damit jede und jeder (und auch du in ein paar Monaten) weiß, was du mit dem Commit bewirken wolltest. GitHub selbst gibt dazu folgende Empfehlungen, ergänzt um meine Tipps:

- Der Commit-Titel sollte nicht mit einem Punkt aufhören, da es sich um eine Überschrift handelt.
- Der Titel sollte nicht aus mehr als 50 Zeichen bestehen (Leerzeichen mitgezählt), siehe Abbildung 3-11.[12] Für detailliertere Informationen dient darunter das Feld *extended description* (»erweiterte Beschreibung«).
- Benutze aktive Sprache, keine passive, also »füge hinzu« anstatt »hinzugefügt« (im Englischen »add« anstatt »added«). Am einfachsten ist es, wenn man sich den Commit-Titel wie einen Befehl ans System denkt: »Ändere dies!«
- Versuche, mit dem Titel deine Absicht auszudrücken.
- Versuche, das Was und Warum zu beschreiben (z. B.: »Restrukturiere Modul A für bessere Lesbarkeit«).

12 Niemand reißt einem den Kopf ab, wenn es 51 Zeichen sind, es geht eher um das Prinzip »fasse dich kurz und präzise«.

- Schreibe die Nachricht so, dass auch du sie in fünf Jahren noch verstehen kannst.

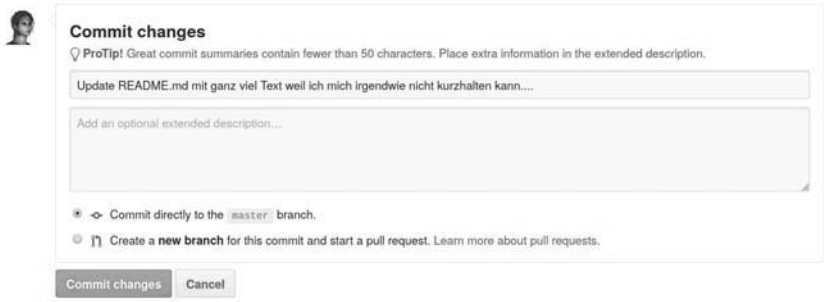

Abbildung 3-11: Wer mehr als 50 Zeichen eintippt, bekommt von GitHub einen entsprechenden Hinweis.

Ein schönes Zitat, das gut verdeutlicht, wieso präzise und klare Commit-Nachrichten so wichtig sind, ist Folgendes:

> Jedes Softwareprojekt ist ein gemeinschaftliches Projekt. Es hat mindestens zwei Entwickler, den ursprünglichen Entwickler und den ursprünglichen Entwickler ein paar Wochen oder Monate später, wenn der Gedankengang längst den Bahnhof verlassen hat.
>
> – *Who-T,* On commit messages, unter *https://who-t.blogspot.com/2009/12/on-commit-messages.html (Übersetzung von deepl.com)*

Nachdem du alle Änderungen dem Projekt hinzugefügt (bzw. committet) hast, gehe auf die Startseite des Projekts und bewundere das neue Erscheinungsbild. Genieße diesen Augenblick! Nichts ist so toll wie der erste Commit, der sofort auch optisch etwas verändert. Wenn du die *README.md* erneut anklickst, bekommst du noch ein paar Zusatzinformationen, wie beispielsweise die Anzahl der Mitwirkenden oder die Dateigröße (siehe Abbildung 3-12).

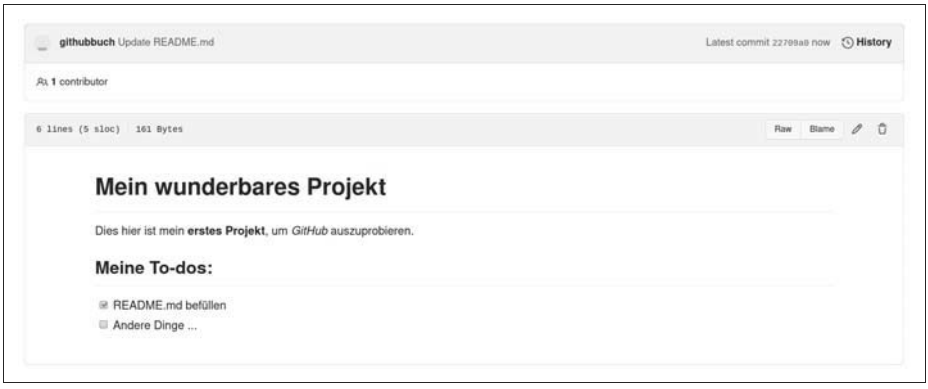

Abbildung 3-12: Durch Anklicken einer Datei erhält man weitere Zusatzinformationen.

SLOC

Vielleicht ist dir in Abbildung 3-12 die Zeile oben links aufgefallen:

```
6 lines (5 sloc) | 161 Bytes
```

Während die Angaben *6 lines* (Anzahl der Zeilen) und *161 Bytes* (Größe der Datei) vermutlich noch relativ eingängig sind, erscheinen die *5 sloc* doch erst einmal ominös, oder? SLOC ausgeschrieben bedeutet *Source Lines of Code* (»Quellcodezeilen« oder auch »Anzahl der Programmzeilen«) und ist eine Metrik, um Software zu vermessen. Wenn du dir unseren Markdown-Text oben noch mal anschaust, siehst du, dass wir sechs Zeilen benutzt haben, aber nur in fünf Zeilen auch wirklich Inhalte (Source Lines) stehen. Das sind unsere SLOC.

SLOC werden häufig genutzt, um ein Gefühl für die Größe und Komplexität einer Software zu bekommen. Beispielsweise hat Windows XP 40 Millionen SLOC, der Linux-Kernel über 25 Millionen.[13]

SLOC sind übrigens *kein* Maßstab für Qualität oder Produktivität, auch wenn sie in Softwareprojekten hin und wieder als Messung für den Fortschritt angewendet werden. Gute Softwareentwicklung zeichnet sich unter anderem auch dadurch aus, dass Codezeilen vereinfacht (»umgeräumt«) oder auch mal entfernt werden[14] und dass nicht nur immer mehr Code hinzugefügt wird. Das nachfolgende Zitat beschreibt das in meinen Augen ganz gut:

> »Die Verwendung von SLOC zur Messung des Softwarefortschritts ist wie die Verwendung von kg zur Messung des Fortschritts bei der Flugzeugherstellung.«
>
> *– Autor*in unbekannt, angeblich Bill Gates*

Wenn du dich genug an deinem Tun ergötzt hast, sollten wir einen Blick auf die Statistik für unser Projekt werfen (siehe Abbildung 3-13).

Vielleicht hast du schon bemerkt, dass sich die Anzahl der Commits auf deinem Projekt erhöht hat. Kein Wunder, du hast ja auch gerade deinen ersten gemacht. Eventuell wunderst du dich aber, dass da die Zahl 2 steht – schließlich war es ja gerade dein erster und nicht bereits der zweite Commit. Wir erinnern uns: Beim Anlegen des Repositorys haben wir GitHub gesagt, es möge für uns doch eine *README.md* anlegen – das war der erste Commit auf deinem Repo.[15]

13 Quelle: *https://de.wikipedia.org/wiki/Lines_of_Code*.

14 Das nennt man dann »Refactoring«.

15 Das gilt übrigens auch, wenn du beim Erstellen *.gitignore* oder *license* anwählst (siehe Abbildung 3-7).

Abbildung 3-13: Auf der Startseite eines Projekts ist die Gesamtanzahl aller Commits zu finden.

Herzlichen Glückwunsch, du hast dein erstes Repository angelegt und auch schon die erste Änderung vorgenommen. Jetzt wollen wir uns Issues und den Umgang damit anschauen.

Den ersten Ablauf üben – Issue anlegen und bearbeiten

Wir wissen aus Abschnitt »Informationen finden (interessierte Anwenderin)« auf Seite 13 in Kapitel 2 bereits, dass Issues Handlungsbedarf aufzeigen sollen (Fehler, Anfragen nach weiteren Funktionen – sogenannte Feature-Requests –, Anmerkungen etc.). Ich habe sie als eine Art offener Brief bezeichnet. Ein Issue ist aber viel mehr als ein einfacher Brief. Stell ihn dir als einen Zettel an einer Pinnwand vor – alle können diesen lesen und auch eigene Kommentare anbringen. Häufig entstehen rege Diskussionen in einem Issue, zum Beispiel über das Für und Wider einer neuen Funktion.

Der größte Vorteil ist, dass Issues als eine Art Dokumentation für alle (auch für zukünftige Teammitglieder) dienen können, da dort optimalerweise alle Informationen zu einem bestimmten Thema zusammenlaufen, z. B. wieso eine Entscheidung so getroffen wurde, wie sie getroffen wurde.

Issues können von dir selbst angelegt werden, in vielen Projekten werden sie aber auch häufig von anderen Menschen (Contributoren) angelegt. Einen Issue schließen können die Projekteignerin, die Maintainerin oder der Contributor, der den Issue erstellt hat.

Um den Umgang damit zu üben, legen wir mal einen Issue an, in dem wir aufschreiben, dass uns eine wichtige Information auf der Startseite fehlt. Es kann dir durchaus passieren, dass andere Menschen auf dein Projekt stoßen und ihnen irgendwelche Informationen fehlen. In der Regel werden sie dies über einen Issue kundtun. Aber auch wenn du selbst Themen findest, die du noch bearbeiten und nicht vergessen möchtest, ist das Anlegen eines Issues eine gute Sache (in Kapitel 10, Abschnitt »Eigene Projektboards – mit Projects den Überblick behalten« auf Seite 236, werden wir sehen, dass es dafür auch andere Wege gibt). Zum einen vergisst du die Dinge nicht, und zum anderen können Besucherinnen deiner Seite

sehen, dass du dir des Themas durchaus bewusst bist und dort in (hoffentlich) naher Zukunft Abhilfe schaffst.[16] Im Folgenden werden wir:

1. einen Issue anlegen,
2. den Issue bearbeiten und
3. den Issue schließen.

Einen Issue anlegen

Gehe zum Register *Issues*, wähle *New Issue* und gib dem Issue einen aussagekräftigen Namen und einen Inhalt, z. B. den aus Abbildung 3-14.[17]

Speichere deinen neuen Issue, indem du auf den Button *Submit new issue* (deutsch etwa »neuen Issue einreichen«) klickst. Wenn du danach im Projektmenü *Issues* auswählst, erscheint jetzt der eben von dir neu angelegte Issue.

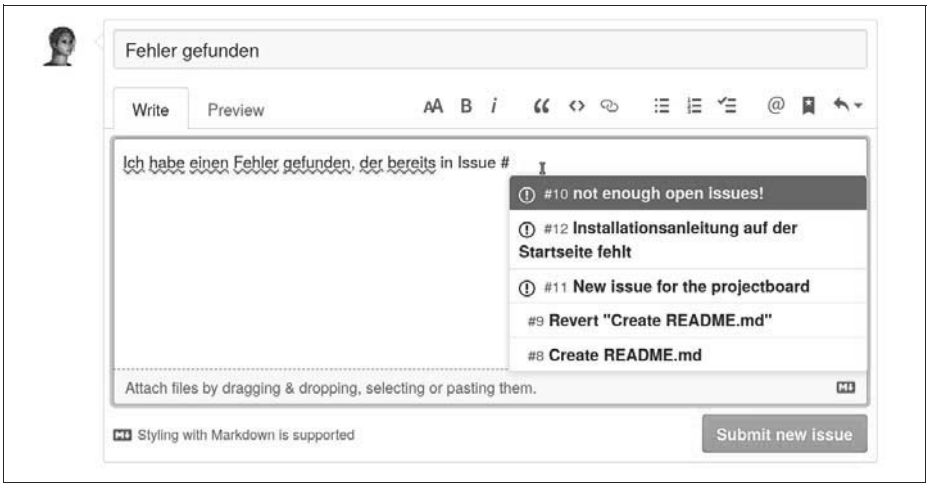

Abbildung 3-14: Das Erstellen eines Issues ist handwerklich ziemlich einfach. Die größere Herausforderung besteht darin, einen gescheiten Titel und eine gute Beschreibung zu finden.

Auf dem Bild habe ich noch eine Besonderheit eingebaut. Innerhalb eines Issues gibt es die Möglichkeit, auf andere Issues oder Pull-Requests zu verlinken. Das ist vor allem dann nützlich, wenn man zeigen möchte, dass ein anderer Issue ein ähnliches Problem adressiert oder ein bestimmter Pull-Request das Problem lösen könnte (Pull-Requests lernst du in Kapitel 4 noch kennen). Diese Verlinkung funktioniert über das Eingeben des Rautesymbols *#*, und GitHub unterstützt uns mit einem Drop-down und einer Auswahl an Möglichkeiten. Das funktioniert allerdings nur, wenn bereits ein Issue oder Pull-Request vorhanden ist.

16 Es kann sogar passieren, dass jemand anderes um die Ecke kommt und den Issue für dich löst ...

17 Wie du in der Abbildung gut sehen kannst, unterkringelt GitHub einige Wörter. Das ist die Rechtschreibprüfung, die leider kein Deutsch spricht.

Es gibt auch noch die Möglichkeit, mit sogenannten *@mentions*[18] Menschen direkt zu adressieren, die beispielsweise vielleicht einen Blick auf den Issue werfen sollten. Diese bekommen dann eine Benachrichtigung, dass sie erwähnt wurden. Abbildung 3-15 ist ein realer Beitrag in einem Issue aus dem *moby*-Projekt[19], das wir in Kapitel 2 bereits kurz kennenlernen durften.

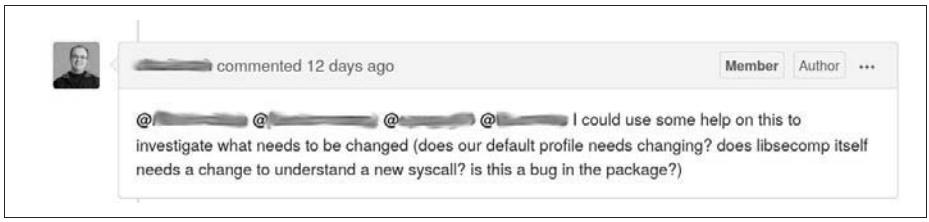

Abbildung 3-15: Über @mentions kann man Menschen direkt ansprechen. Diese erhalten eine entsprechende Benachrichtigung.

Sofern du Projekteignerin oder Maintainerin bist, kannst du Issues bestimmten Personen zuordnen, die dann für den entsprechenden Issue verantwortlich sind (englisch *Asignee*). Diese Verantwortung kann im Laufe der Zeit wechseln, z. B. wenn der Datenbankspezialist bei der Suche feststellt, dass der Fehler an einer ganz anderen Stelle ist. Wenn er den Fehler deswegen nicht lösen kann, gibt er den Issue vielleicht weiter an die Hauptentwicklerin. Da dein Projekt noch relativ frisch und neu ist, gibt es bisher nur eine Person, der du die Verantwortung übertragen kannst.[20]

Um Issues besser wiederfinden zu können oder auch um die Dringlichkeit oder Wichtigkeit eines Issues klarzumachen, kann man sie mit sogenannten *Labels* klassifizieren. GitHub bietet von Haus aus eine Reihe vorgefertigter Labels an (siehe Abbildung 3-16). Es ist aber auch möglich, eigene Labels zu erstellen, wenn es für das jeweilige Projekt sinnvoll erscheint. Labels werden uns später noch helfen, andere Projekte zu finden, bei denen wir unterstützen können (siehe Abschnitt »Fremdes Projekt suchen« auf Seite 117 in Kapitel 6).

Wir wollen zu Übungszwecken dem Issue ein Label geben und dir zuweisen. Es gibt dafür zwei Wege.

- **Weg 1**: Du wählst den entsprechenden Issue aus und bekommst auf der rechten Seite ein entsprechendes Menü (siehe Abbildung 3-17).
- **Weg 2**: Du gehst auf die Issue-Übersichtsseite (siehe Abbildung 3-18) und wählst über die Checkbox den entsprechenden Issue an ❶. Danach kannst du über die Drop-down-Felder die Aktionen durchführen ❷ + ❸. Dieser Weg hat den großen Vorteil, dass du mehrere Issues gleichzeitig bearbeiten kannst.

18 Das @-Symbol wird mitgesprochen.

19 *https://github.com/moby/moby*

20 Wenig überraschend: dir selbst!

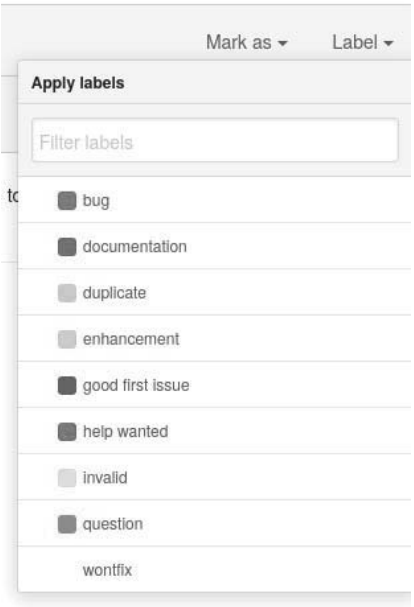

Abbildung 3-16: GitHub bietet einige Labels von Haus aus bereits an. Jedes Label kann eine unterschiedliche Farbe haben.

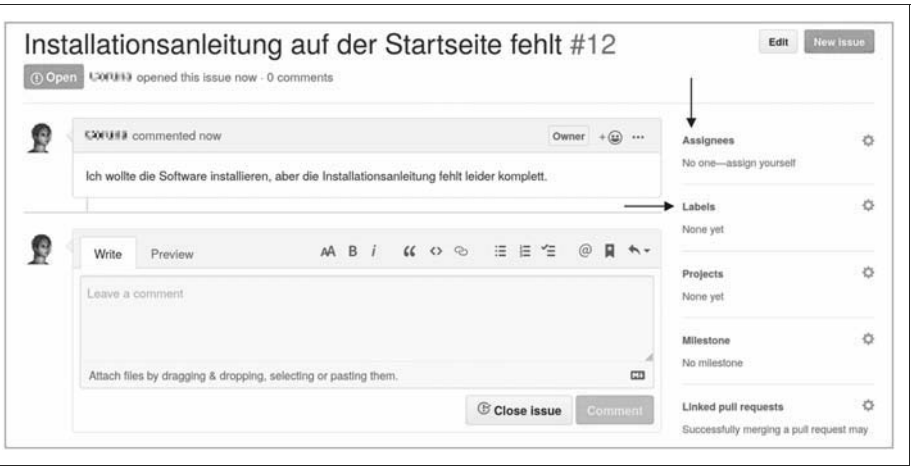

Abbildung 3-17: Issue labeln und zuweisen, Weg 1: direkt im Issue auf der rechten Seite

Abbildung 3-18: Issue labeln und zuweisen, Weg 2: in der Issue-Übersicht nach Anklicken des/der jeweiligen Issues

Wähle ein Label deiner Wahl (z.B. *enhancement*) und wähle dich als *Asignee* aus. Das Ganze könnte dann so aussehen wie in Abbildung 3-19. Das Zuweisen eines Labels oder Asignees erfolgt durch einen einfachen Klick und muss nicht noch gesondert gespeichert oder bestätigt werden. Das kann leicht dazu führen, dass man aus Versehen etwas falsch zuweist. Wie man in der Abbildung sieht, habe ich zunächst das falsche Label *question* ausgewählt und bin danach erst zum richtigen Label *enhancement* gegangen. Hier sieht man sehr gut, dass GitHub jede Aktivität in einem Issue auch dokumentiert und dadurch für jeden und jede nachvollziehbar macht – selbst wenn es sich um einen Fehler handeln sollte.

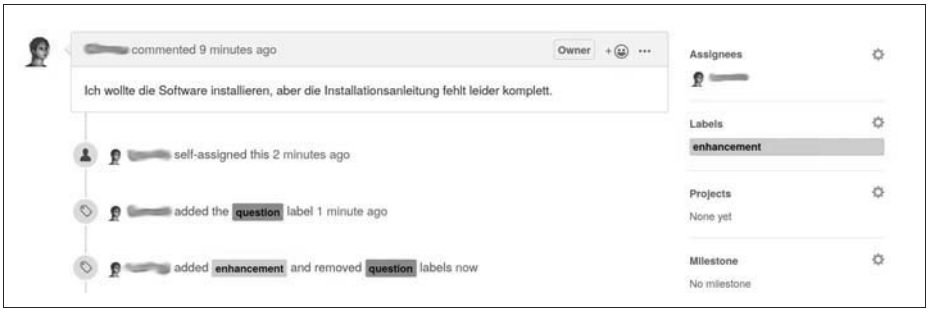

Abbildung 3-19: Issues können Personen (»Asignees«) zugeordnet werden und können Etiketten (»Labels«) haben, um sie klassifizieren zu können.

Den Issue bearbeiten und schließen

Jetzt wollen wir das Problem lösen, das in dem Issue beschrieben wird, indem wir die Startseite entsprechend editieren (man nennt das häufig auch »den Issue lösen«). Gehe zur *README.md*, wechsle in den Editormodus und führe ein paar Änderungen durch. Bevor du speicherst, tippst du in die detaillierte Beschreibung `fixes #1` ein. Sofern du nicht schon andere Dinge innerhalb deines Projekts gemacht hast, adressiert das unseren eben erstellten Issue (*1*). GitHub gibt dir dazu via Tool-

tipp[21] einen Hinweis dazu, ob dem so ist (siehe Abbildung 3-20). Solltest du schon etwas »rumgespielt« haben, musst du eventuell eine andere Identifikationsnummer (ID) angeben, wie man bei mir mit der Nummer *12* gut sieht. Die ID deines Issues findest du in der Issue-Übersicht (siehe Abbildung 3-21).

Abbildung 3-20: GitHub unterstützt dich mit nützlichen Tooltipps.

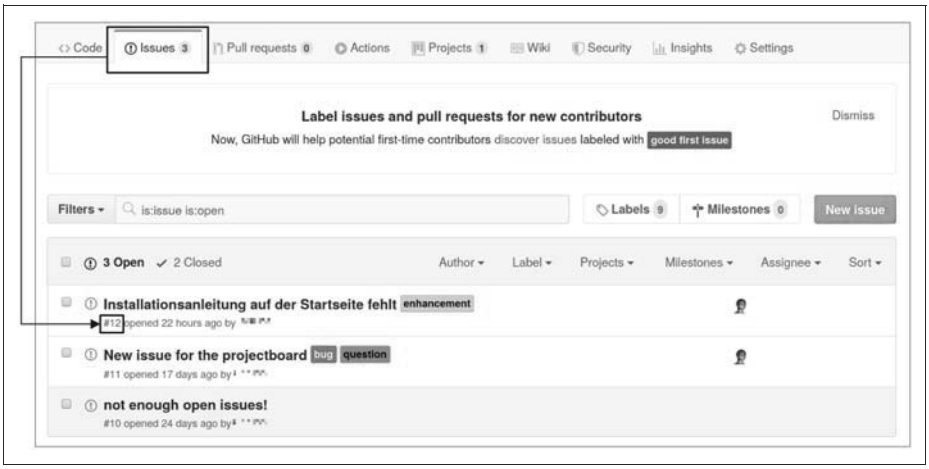

Abbildung 3-21: Die Identifikationsnummer (ID) eines Issues ist in der Issue-Übersicht zu finden.

Sobald du die richtige ID angegeben hast, speicherst du wie vorher auch. Jetzt ist aber noch etwas Bemerkenswertes passiert. Wenn du in die Issue-Übersicht wechselst, siehst du, dass der Issue weg ist?! Ja und nein, der Issue ist nicht weg, sondern wurde automatisch geschlossen. Der Standardfilter sorgt dafür, dass nur offene Issues angezeigt werden, daher kannst du ihn nicht mehr sehen (erinnere dich an

21 Ein kleines Fenster, das sich in Programmen oder Webseiten öffnet und in der Regel eine kurze Beschreibung oder Hilfestellung gibt.

Kapitel 2, Abschnitt »Informationen finden (interessierte Anwenderin)« auf Seite 13, hier hatte ich die beiden Standardfilter *is:issue is:open* bereits erklärt). Wenn du den Filter veränderst, um dir die geschlossenen Issues anzeigen zu lassen, siehst du den bearbeiteten Issue wieder. Klickst du nun den entsprechenden Issue noch mal an, bekommst du ein paar weitere Informationen, beispielsweise dürfte dort so etwas Ähnliches stehen wie:

```
DeinUsername closed this in 7a0eeb2 2 minutes ago
```

Dabei ist die komische kryptische Zeichenkette (in meinem Fall `7a0eeb2`) mit einem Link versehen. Diese Zeichenkette ist die sogenannte *Commit-ID*. Jeder Commit, den du machst (oder andere machen), hat eine eigene ID, um die einzelnen Commits voneinander unterscheiden zu können. Später im Kapitel zu Git (siehe Kapitel 7) werden wir der ID erneut begegnen.

Commit-ID

Die Commit-ID bei Git/GitHub besteht aus einem sogenannten SHA-1-Hash. Es handelt sich dabei um einen Algorithmus, der aus einer Eingabemenge eine 40 Zeichen lange Ausgabemenge produziert. Das heißt, hinten kommen immer 40 Zeichen raus – egal wie groß die Eingabemenge ist.

Das entscheidende Feature dabei ist, dass die exakt gleiche Eingabe auch immer die exakt gleichen 40 Zeichen erzeugt. Das hat den Vorteil, dass man dadurch relativ schnell sehen kann, ob zwei Commits identisch sind oder nicht. Git/GitHub bildet einfach mit zwei Commit-Eingabemengen (Inhalt, Datum etc.) den Hash-Wert, und wenn dieser gleich ist, sind beide Commits identisch.

Da 40 Zeichen ziemlich lang sind, um sie ständig anzuzeigen oder auch noch ständig irgendwo einzutippen, verwendet man in der Regel nur die ersten sieben bis zwölf Zeichen. Diese sind meist eindeutig genug.

Du kannst Issues auch manuell schließen, indem du in dem Issue ganz nach unten scrollst und den Button *Close issue* (deutsch »Issue schließen«) anwählst (siehe Abbildung 3-22). Das ist natürlich nicht ganz so toll wie das automatische Schließen, aber manchmal gibt es Issues, die einfach nicht gelöst werden – beispielsweise weil irgendeine Besucherin ein völlig abgefahrenes Feature über einen Issue bei dir eingekippt hat, das du nicht in dein Projekt aufnehmen möchtest. Sollte so etwas der Fall sein, schreibe in den Kommentar, warum du den Issue nicht umsetzen wirst, und zwar nicht nur, weil es höflicher ist, es erhöht auch die Chance, dass die Menschen weiterhin Issues anlegen (so du das denn möchtest). Nichts ist frustrierender, als ohne Begründung »abserviert« zu werden.

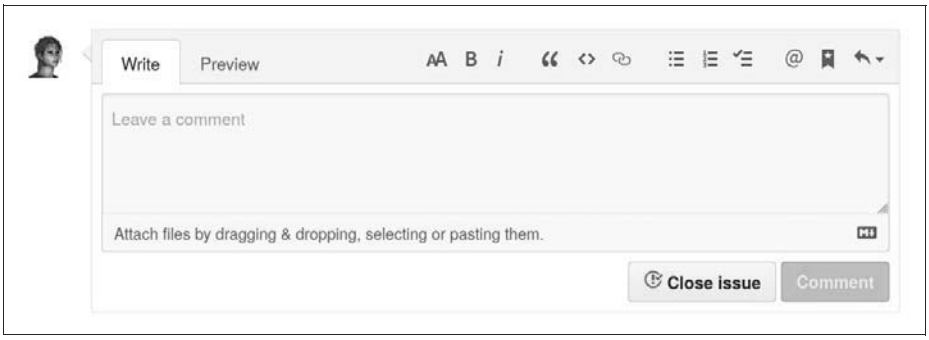

Abbildung 3-22: Issues kann man auch manuell schließen.

Gelöste Issues schließen

Es ist gute Praxis, gelöste Issues zu schließen. Du behältst dadurch die Übersicht, und potenzielle Contributoren versuchen sich nicht an einer Lösung und verbraten Zeit.

Issues mithilfe von Schlüsselwörtern automatisch schließen

GitHub bietet die Möglichkeit, Issues über Schlüsselwörter zu schließen. Die Schlüsselwörter sehen dabei wie folgt aus:

- *close*
- *closes*
- *closed*
- *fix*
- *fixes*
- *fixed*
- *resolve*
- *resolves*
- *resolved*

Sobald in einem Commit eines dieser Schlüsselwörter mit einer Referenz auf den Issue eingegeben wird (*#ISSUE_NR*), wird dieser Issue geschlossen. Man kann auch mehrere gleichzeitig schließen, dafür muss aber vor jeder Referenz eines der Schlüsselwörter stehen, also etwa *fixes #1, fixes #2*.[22]

Man kann auch Issues in anderen Repositories schließen, vorausgesetzt, man hat die entsprechenden Berechtigungen. Das geht mit einem der Schlüsselwörter und als Referenz *USERNAME/REPONAME#ISSUE_NR*.

22 Das Ganze gilt übrigens nur für den Standard-Branch (master), erfolgt ein Commit auf einen anderen Branch, wird der Issue nicht automatisch geschlossen. Wir waren bisher nur auf dem Standard-Branch unterwegs, daher sei diese Info nur der Vollständigkeit halber ergänzt. Keine Sorge, das Thema Branches wird im nächsten Kapitel noch klarer.

Du weißt nun, wie man Issues anlegt, sie labelt und jemandem zuweist. Als cooles neues Zusatzwissen kannst du jetzt auch Issues automatisch schließen lassen, wenn du mit bestimmten Schlüsselwörtern bei einem Commit arbeitest.

Eventuell hast du bereits ein oder mehrere Repositories zum Testen und Rumspielen angelegt. Um deinen Account wieder etwas aufzuräumen, zeige ich dir im nächsten Abschnitt, wie du ein Repository löschen kannst.

Ein bestehendes Repository löschen

Das Löschen eines bestehenden Repositorys ist relativ einfach, aber auf den ersten Blick etwas furchteinflößend, wir müssen nämlich dafür in die *Danger Zone* (also in den »Gefahrenbereich«), siehe Abbildung 3-23. Gehe auf dein zu löschendes Repository und wähle *Settings* aus. Um in die *Danger Zone* zu kommen, musst du ganz nach unten scrollen. Achte darauf, dass im linken Menü *Options* ausgewählt ist (sollte beim Anklicken von *Settings* standardmäßig der Fall sein).

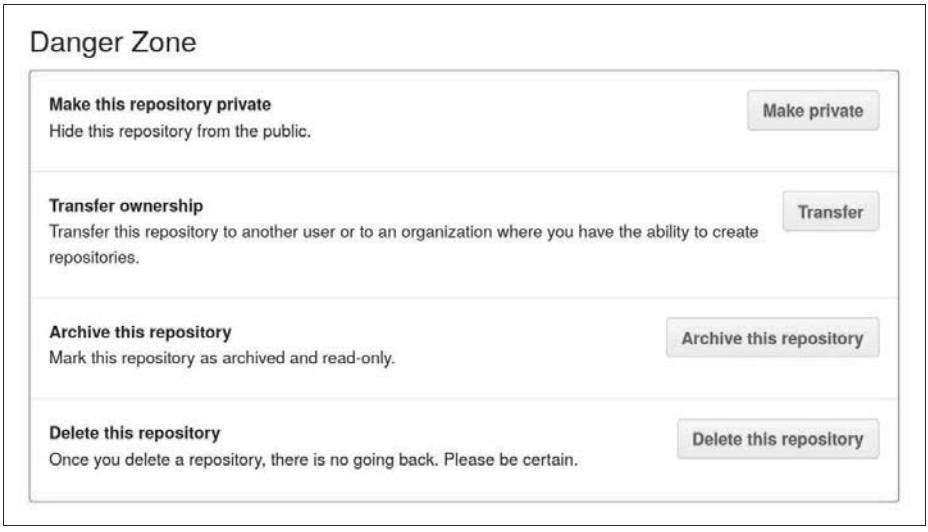

Abbildung 3-23: In der Danger Zone werden alle gefährlichen Aktionen eines Repositorys gebündelt.

In der *Danger Zone* wähle den Button *Delete this repository* (deutsch »Lösche dieses Repository«) aus. Es erscheint eine Warnmeldung, die dich dazu auffordert, den Namen deines Repositorys nach dem Muster USERNAME/REPONAME einzutippen, bevor du die Löschung mit Klick auf den Button *I understand the consequences, delete this repository* (deutsch etwa »Ich verstehe die Konsequenzen, lösche dieses Repository«) bestätigen kannst (siehe Abbildung 3-24). Das ist ein Sicherheitsmechanismus, damit du dein Repository mit all seinem Inhalt (Dateien, Issues, Kommentare, Wiki etc.) nicht aus Versehen löscht.

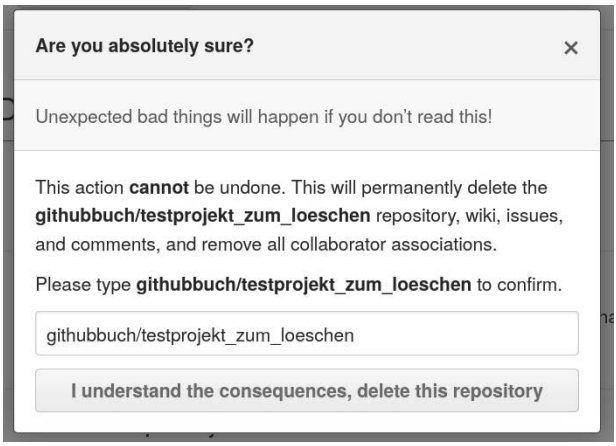

Abbildung 3-24: Als Sicherheitsmechanismus ist zur Löschung der Name des Repositorys vorher einzugeben.

Jetzt weißt du, wie du überflüssige und eventuell auch verbastelte Repositories löschen kannst. Als Nächstes (und Letztes für dieses Kapitel) möchte ich mit dir einmal durchgehen, wie du ein bereits bestehendes eigenes Projekt, das aktuell irgendwo auf der Festplatte schlummert, auf GitHub bekommst.

Ein bestehendes Projekt hochladen

Ein bestehendes Projekt hochzuladen, ist simpel. Lege ein neues Repository für dein Projekt an. Klicke auf den Button *Add file* und dann auf *Upload files* (siehe auch Abbildung 3-25). Danach hast du die Möglichkeit, per Drag-and-drop oder durch Anklicken des Links *choose your files* die gewünschten Dateien und/oder Ordner auszuwählen und hochzuladen (siehe auch Abbildung 3-26).

Abbildung 3-25: Dateien und Ordner hochladen geht über Add file und Upload files.

Am Ende muss nur noch ein Commit durchgeführt werden, und die Dateien und/oder Ordner sind auf dein Repository hochgeladen. Bitte beachte: Die Dateien dürfen die Größe von 25 MByte pro Datei nicht überschreiten! Solltest du größere Dateien hochladen wollen, musst du auf die Konsole ausweichen (das lernst du in Kapitel 8 kennen).

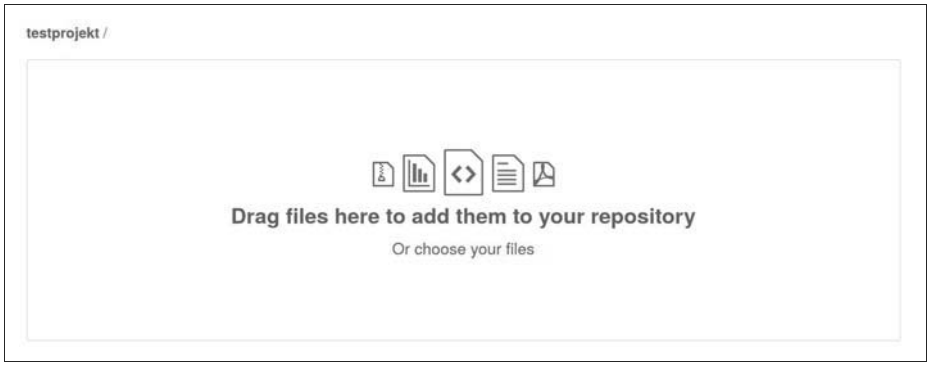

Abbildung 3-26: Via Drag-and-drop lassen sich Dateien und Ordner ganz leicht hinzufügen.

Damit hast du jetzt die Basis, um mit GitHub die ersten Schritte unfallfrei zu gehen. Um aber wirklich produktiv mit anderen zusammenzuarbeiten, fehlen noch ein paar Dinge. Da bei GitHub die Kollaboration im Vordergrund steht (wir erinnern uns an Kapitel 1, Abschnitt »Einsatzgebiete von GitHub« auf Seite 3), werden wir uns im nächsten Kapitel damit beschäftigen, was es zu beachten gibt, wenn plötzlich andere an unserem Projekt mitarbeiten.