CHAPTER 3

The basis: The first own GitHub project

What do you need to get started with GitHub? A functional computer with one of the operating systems Windows, Linux or macOS, as well as a stable Internet connection and an Internet browser. Otherwise , you should test the following :

- Get a free GitHub account (how, I describe in the following section »Create account« on page 20).
- If you want to work with Git: Install Git on your computer (for instructions, see Chapter 7, section "Installing and setting up Git" on page 141).

If you've already set up an account on GitHub, you can skip this chapter if necessary. Maybe you take a quick look at the sections »Protect account« on page 22 and »Become invisible – protect your own e-mail address« on page 23 if you have not yet dealt with the protection of your account or e-mail address.

At the end of the chapter you can ...

- create an account on GitHub and protect it and your e-mail address.
- Create repositories and make changes to them.
- Create, classify, assign and close issues to someone.
- Delete a repository .
- Upload a local project to GitHub.

If you are not yet sure whether you want to use Git, you can also catch up on the installation later as soon as you have arrived in the corresponding chapter. For large parts of the book we will not need Git. And who knows, maybe the functions of GitHub are enough for you?

Create an account

Setting up an account on GitHub is relatively easy. To do this, click on the *Sign up* button in the menu of GitHub[1] (german »register«) and fill in the required information accordingly (see also Figure 3-1).

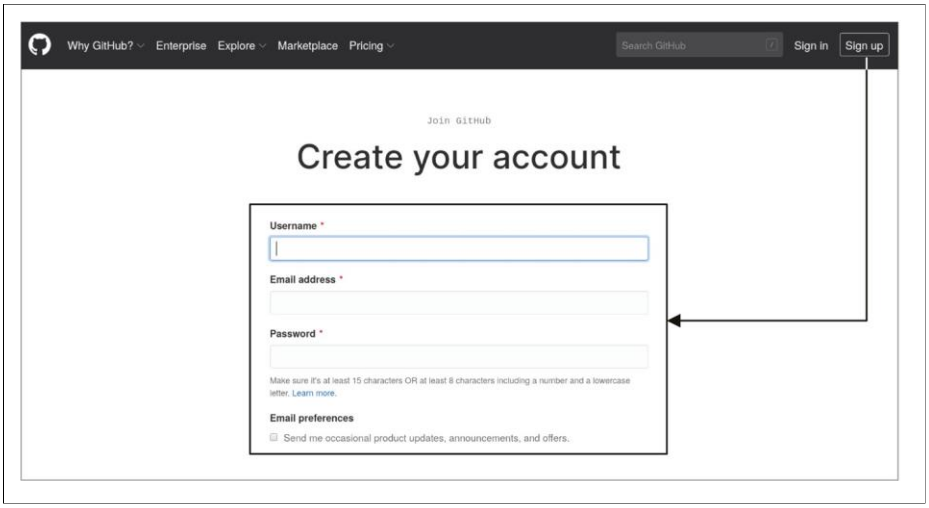

Why GitHub?
Enterprise
Explore
Marketplace
Pricing
Search GitHub
Sign in
Sign up
Join GitHub
Create your account
Username *
Email address *
Password *
Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.
Email preferences
Send me occasional product updates, announcements, and offers.

Figure *3-1: Registering with GitHub is easy:* Choose *your username, enter your own e-mail address and assign a password – done.*

Once you're logged in, the GitHub menu changes a bit (see Figure 3-2).

- 1 will always get you back to the GitHub homepage.
- In 2 is the search function, with which you can search for repositories, but also within a repository.
- The points under 3 are, on the one hand, quick access to the self-created *pull requests* and *issues* (which you will get to know even more closely in this and Chapter 4). The *Marketplace* point is intended for expanding your own repository (see Chapter 9), and *Explore* helps us find other projects (see Chapter 6).
- At 4, quick access isavailable, for example, to create a new repository.
- In 5 you will find everything relevant to your account, e.g. your repositories or the possibility to log out.

1 *https://github.com*

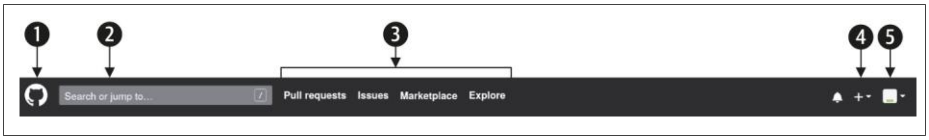


Figure 3-2: After logging in, the GitHub menu changes slightly.

Standardprofilbild – Identicon

After logging in to GitHub, you may have noticed a strange little pixelated image at the top right of the page (in Figure 3-2 at point 5) that looks similar to one of the three in Figure 3-3. This is a so-called *iden- ticon* – a portmanteau word[2] from »Identification« and »Icon« (to German »Identification« and »Symbol«) – and you can see your profile picture here.

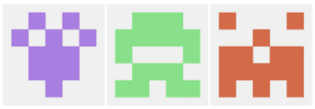


Figure 3-3: Identicons uses the default profile picture on GitHub. (Image source: https://github.blog/2013-08-14-identicons)

GitHub automatically generates these images depending on your user ID. So there is no way to adjust the color or pattern. However, you can replace the image with your own at any time (and also return to the Identicon).

Tip: Password safe

At this point, I would like to promote the use of a password safe, which not only keeps all accounts and passwords secure, but often also comes with a password generator for secure passwords.

Personally, I use KeePassXC[3], which offers many expansion options . For example, there is an extension for the browser with which I can "beam" access data from the safe directly into the input mask of a website. And with the smartphone app *Keepass2Android*, I also have my access data available on mobile devices if I need them.

2 Merging two words into a new meaning, e.g. breakfast + lunch = brunch.

3 *https://keepassxc.org/*

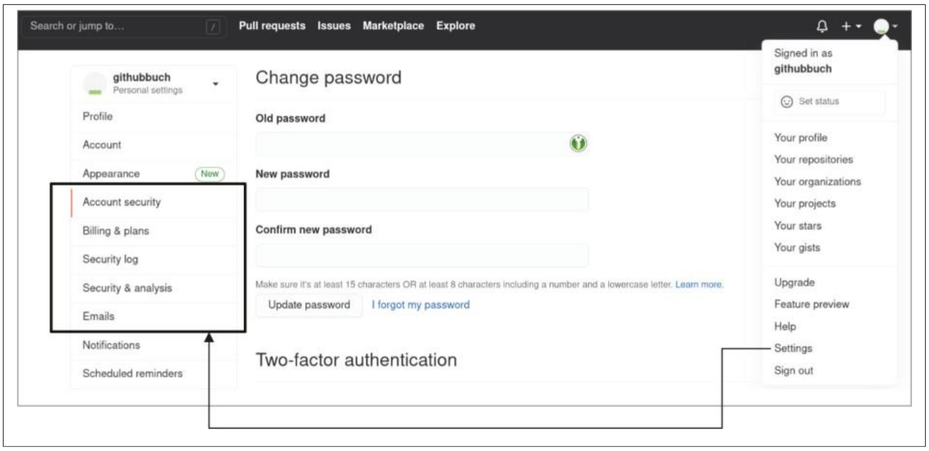
Protect your account

One thing you can set up to increase the security of your account is 2-factor authentication. You can find them in the *settings* of your profile under *Account security* (see also Figure 3-4). Follow the instructions there to set up this form of authentication. You have the option to choose either an authenticator app as a second factor or a procedure via SMS. GitHub recommends *Authy*[4], *1Password*[5], or *LastPass Authenticator*[6] as an authenticator app for your smartphone. Every time you log in, GitHub will ask you to enter a code either via app or SMS , depending on what you've set up . you can access your account .

2-factor authentication recovery options

Once 2-factor authentication is set up, you should set up one of the *recovery* options offered by GitHub. If your smartphone is lost, these options give you the opportunity to regain access to your Git Hub account. GitHub offers as options:

- **Recovery codes**: A series of codes that can be used for recovery in the event of loss of the smartphone (preferably stored in a password safe).
- **Fallback SMS number**: An alternative phone number to which Git Hub sends the recovery codes in the event of a smartphone loss.
- **Recovery tokens**: The ability to regain access to the GitHub account via Facebook .



Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

Figure 3-4: In the profile, you can access the relevant settings via »Settings« .

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-factor authentication

2-factor authentication – also known as 2FA – is an additional layer of security to protect an account from unauthorized access. If I want to have access to my account, I need at least two different
»Factors« to be able to log in at all. Factors can be:

- Something I know: password, PIN code, special pattern , etc.
- Something I have: smartphone credit card, hardware token, etc.
- Something I on the: fingerprint, iris scan, voice recognition etc.

If a factor is lost – for example, if a stranger has found out my password – they will still not have access to my account unless they also know the second factor.

Sometimes you also read the term "MFA", which stands for *multi-factor authentication*. This means that I use more than one factor. So 2FA is a subset of MFA.

All events relevant to the security of your account are stored under the *security point Security log* , such as when you logged in, from where and with which IP address[7]. You can also use the *security log* to display other events – not just those that affect your account. If you enter operation:create in the existing search field, all events where you have created something new will be displayed, such as a new repositoryy. With created:2020-02-18 you can display everything you have on February 18, 2020 . [8]

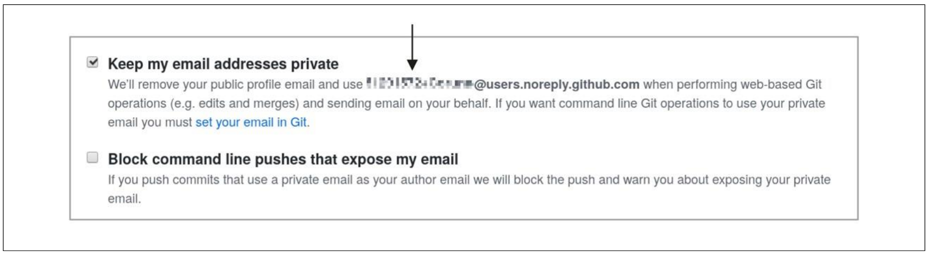
Become invisible - protect your own e-mail address

One thing you should keep in mind is your email address, which you provide when setting up your account. If you don't explicitly change it, your email address will appear on your profile page and will also be used for activities within GitHub. [9] Some people don't want that, so GitHub gives you the opportunity to be a little more anonymous. In your profile in the *Settings* menu under *Emails*, you can specify that GitHub *treats* your email address privately (see Figure 3-4 and Figure 3-5).

7 This is the "mailing address" of your computer.

8 A detailed list of what else is possible can be found at *https://help.github.com/en/github/authenticating-to-github/*reviewing-your-security-log.

9 So-called web-based Git operations, which becomes more understandable as soon as we deal with Git in Chapter 7.



☑ **Keep my email addresses private**
We'll remove your public profile email and use [redacted]@users.noreply.github.com when performing web-based Git operations (e.g. edits and merges) and sending email on your behalf. If you want command line Git operations to use your private email you must set your email in Git.

☐ **Block command line pushes that expose my email**
If you push commits that use a private email as your author email we will block the push and warn you about exposing your private email.

Figure 3-5: GitHub offers the possibility to keep your e-mail address secret – I have anonymized my *anonymous e-mail address here again by pixelation.* [10th]

As you can see in the picture, you will be shown a kind of "alternative e-mail address", which is generated from a seven-digit identification number (ID) and your username: *ID+username@users.noreply.github.com* (for example, 1234567+geheimniskraemer@users.noreply.github.com). And why do I need this e-mail address? In »Install and set up Git« on page 141 I will show you a use case later, at the moment we accept that there is this anonymous e-mail address.

Furthermore, you can see on the picture that you also have the option to bleak something. This setting is only relevant if you plan to work with Git later.

You can also set a lot more on GitHub, such as uploading a profile picture or something similar. However, since this does not help us to understand the basic functions of GitHub, I will not go into more detail here . But of course you can explore the other settings .

Create your first own repository

Now that you are ready to go, we will create a first own project from thissection and make changes to the files available there. We get to know *commit* and *issues* in more detail, which form the basis for many other activities .

There are several possibilities for creating your own project. We use the plus icon in the upper right corner of the GitHub menu to get to the most obvious one (see also Figure 3-6) and choose *New repository*.

10 Whenever you see any blurred traces in the screenshots, that's my account name. I have cluttered the account for the screenshots in this book so much that it is a little embarrassing for me and I therefore do not want to present it here :).

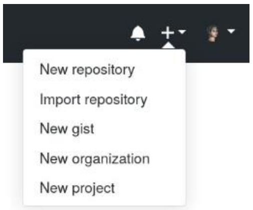
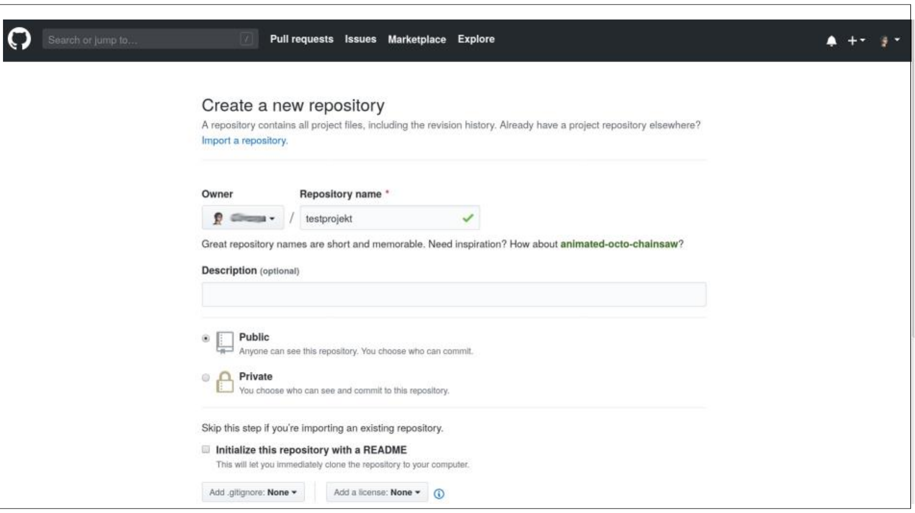


Figure 3-6: A new repository can be created *using the plus symbol at the top right.*

Tip: Shortcut for new repositories
Another way to create repositories is the web address *https://repo.new* in the browser. Then you're right on the Git hub page for new repositories. Of course, this only works if you are already logged in, otherwise you will first land on the log-in page.

We are now asked to provide a number of details (see Figure 3-7). Give your project a meaningful name – this can also be something as original as "test project" – and, if you like, a corresponding description. Everything can be changed later .


Search or jump to... Pull requests Issues Marketplace Explore

Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.

Owner: [redacted] / Repository name *: testprojekt ✓

Great repository names are short and memorable. Need inspiration? How about **animated-octo-chainsaw**?

Description (optional)

● **Public**
Anyone can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾ | Add a license: **None** ▾ ⓘ

Figure 3-7: A new project/repository is created *with just a few clicks.*

During the creation process, you will also be asked if you want to create a public or a private repository. Public repositories can later be viewed by any visitor, but can only be modified with permission . On the other hand, only you have access to private repositories. If you set a *collaborator* for a repository, it can change and of course see everything, even if the repo is set as private . We are brave here and choose *Public*.

Tip: Create test repositories

Once you just want to familiarize yourself with GitHub and its features, it's a good idea to create one or more repositories for testing and playing around. You can delete them after your experiments (see section "Deleting an existing repository" on page 39 below in this chapter).

You will also be asked if GitHub should create a *README.md* for you (we remember Chapter 2, section "Finding information (interested users)" on page 13). Since this is generally a good idea, we do this of course. You can leave everything else as it is. As soon as you have clicked on *Create Repository* , GitHub automatically creates a repository with a still quite empty *README.md* and all the possibilities that GitHub offers – for example, the creation of issues.

Important to know: During the course of the book, we will work exclusively on public repositories. If any feature I'm introducing to you doesn't work, it might be because your repository is private. GitHub provides many features for public repos for free. If you want to use the same features in private repos, this only works after inserting small coins.

Make a change in the content of the project

First, we fill the *README.md*. To do this, go to <> *code* if you're not already there. On the right side of the screenshot, you'll see a pencil icon (see Figure 3-8). The icon generally indicates that files can be edited . The pencil icon takes you to a text editor where you can fill the *README.md* with content.

As you can see in Figure 3-9, the text editor offers two options: *Edit file* and *Preview changes*. By default, you are in *Edit file* mode, where you can make your changes. The *Preview changes* mode offers you a preview of the current changes without having to save them. Then let's fill the file with life. As already mentioned, you can use the so-called *markdown* to format the file . For example, you could type the following:[11]

11 Just for the sake of completeness: This is »GitHub Flavored Markdown«, i.e. Markdown, which has been somewhat enriched by GitHub. For example, the task list is one of them. More details can be found here: *https://guides.github.com/features/mastering-markdown/*.

```
# My wonderful project
 This is my **first project** to  try  *GitHub*.

## Meine To-dos:
- [x]  Fill README.md
- [ ] Other things ...
```

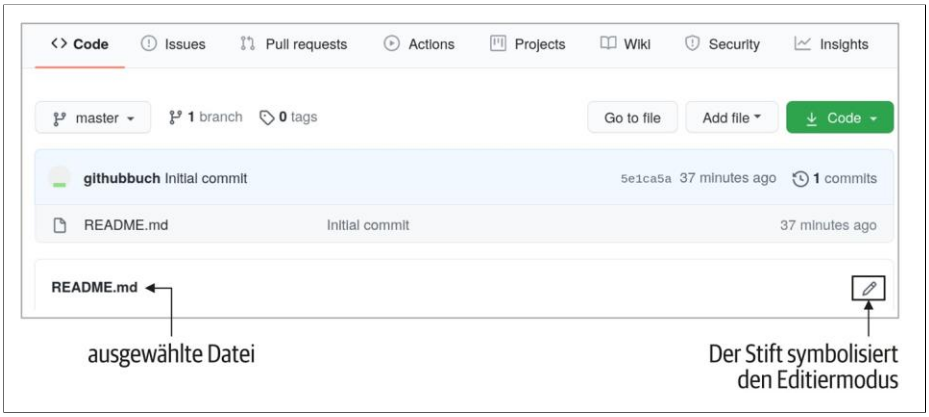


Figure 3-8: By selecting the pencil symbol you can edit the file in an editor .

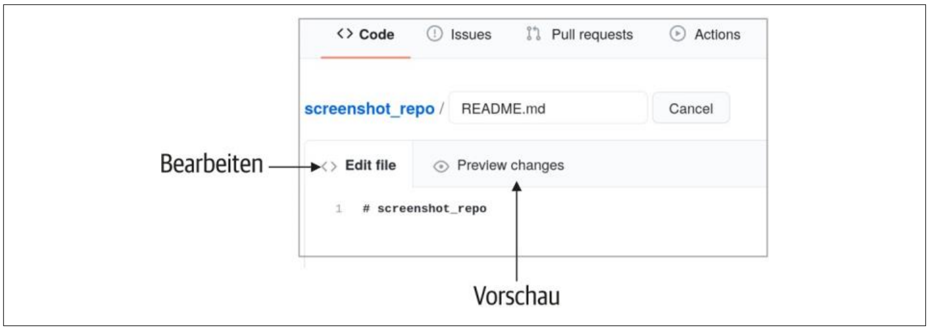
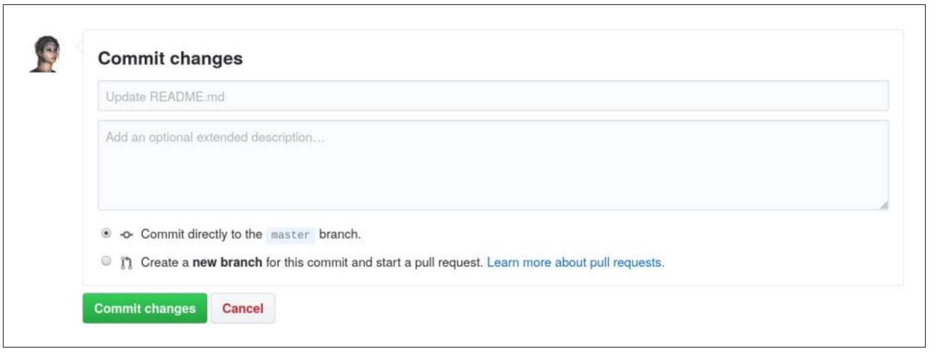


Figure 3-9: GitHub offers a text editor with Edit and Preview modes.

 Play around with it a bit, you can find suggestions in the explainer bear box "README.md" in Chapter 2.

Via the preview function, you can check at any time whether the whole thing looks as you imagined it. If you are satisfied with your work, you naturally want to save it. You can find this by scrolling down (see Figure 3-10). But oops, it doesn't say anything like "commit" and "branch". We remember once: *Commit* is something like *save* and adds the changes to the project. Later, when we will work with Git (see Chapter 7), we will see that this is not entirely correct. For the moment, however, it is enough to equate the respective terms.


Commit changes

Update README.md

Add an optional extended description…

● Commit directly to the `master` branch.

○ Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

Figure 3-10: Saving changes is called a "commit" on *GitHub*.

As far as the branch is concerned, we first leave the default setting (*Commit directly to the master branch.,* to German about "save directly to the main branch"). We'll see what it's all about later (see chapter 4). What you should noch fill in is the commit message (the text boxes). Nicely, GitHub already makes us a suggestion for the title of the commit message, which you can take over for the time being.

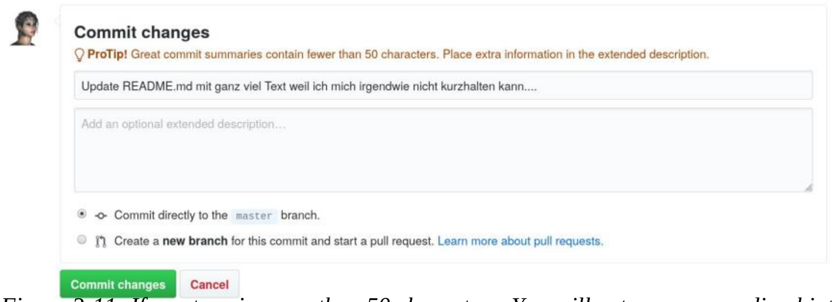
Commit messages

Commit messages on GitHub consist of two parts: the title and a (optional) more detailed description. You should always fill in the title so that everyone (and you in a few months) knows what you wanted to achieve with the commit. GitHub itself gives the following recommendations, supplemented by my tips:

- The commit title should not stop with a period because it is a heading .
- The title should not be longer than 50 characters (spaces count), see Figure 3-11. [12] For more detailed information, the *extended description* field is used below.
- Use active language, not passive, as"add" instead of "added". The easiest way is to think of the commit title like a command to the system: "Change this!"
- Try using the title to express your intention.
- Try to describe the what and why (e.g.: "Restructure module A for better readability").

12 Nobody tears your head off when it's 51 characters, it's more about the principle "keep yourself short and precise".

- Write the message so that you too can still understand it in five years.



Commit changes

ProTip! Great commit summaries contain fewer than 50 characters. Place extra information in the extended description.

Update README.md mit ganz viel Text weil ich mich irgendwie nicht kurzhalten kann....

Add an optional extended description...

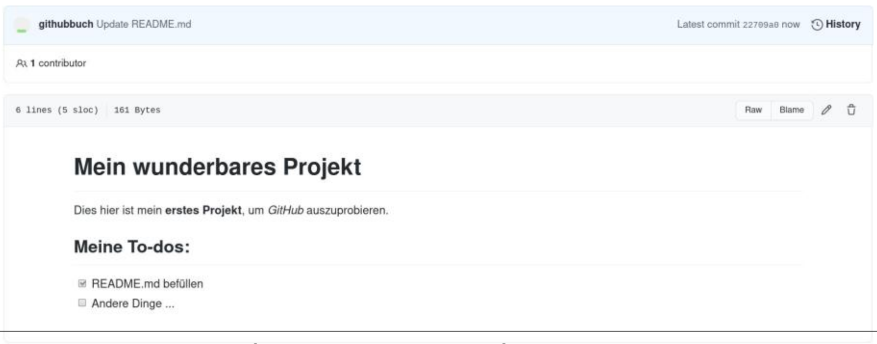
- [✓] Commit directly to the `master` branch.
- Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

Figure 3-11: If you type in more than 50 characters, You will get a corresponding hint from GitHub.

A nice quote that illustrates why precise and clear commit messages are so important is the following:

> Every software project is a collaborative project. It has at least two developers, the original developer and the original developer a few weeks or months later, when the train of thought has long since left the station.
>
> – *Who-T,* On commit messages, unter *https://who-t.blogspot.com/2009/12/on-commit-messages.html (Übersetzung von deepl.com)*

After you've added (or committed) all the changes to the project, go to the project's home page and admire the new look. Enjoy this moment! Nothing is as great as the first commit, which immediately changes something visually. If you click on the *README.md* again, you will get some additional information, such as the number of participants or the file size (see Figure 3-12).



githubbuch Update README.md — Latest commit 22709a0 now — History

1 contributor

`6 lines (5 sloc) | 161 Bytes`

Mein wunderbares Projekt

Dies hier ist mein **erstes Projekt**, um *GitHub* auszuprobieren.

Meine To-dos:

- [x] README.md befüllen
- [ ] Andere Dinge ...

Figure 3-12: Clicking on a file provides additional information.

SLOC

You may have noticed the line in the upper left corner of Figure 3-12:

6 lines (5 sloc) | 161 Bytes

While the specifications *6 lines* (number of lines) and *161 bytes* (size of the file) are probably still relatively catchy, the *5 sloc* seem omi- nös, right? SLOC means *source lines of code* (»source code lines« or »number of program lines«) andis a metric for measuring software . If you look at our Markdown text above, you will see that we have used six lines, but only in five lines there are really contents (source lines). These are our SLOC.

SLOC are often used to get a feel for the size and complexity of a software. For example, Windows XP has 40 million SLOC, the Linux kernel over 25 million. [13]

By the way , *SLOTs are not a* measure of quality or productivity, even if they are occasionally used in software projects as a measure of progress. Good software development is also characterized by the fact that lines of code are simplified (»rearranged«) or sometimes removed [14] and that not only more and more code is added. The following quote describes this quite well in my eyes:

> "Using SLOC to measure software progress is like using kg to measure progress in aircraft manufacturing."
>
> – *Author unknown, allegedly Bill Gates*

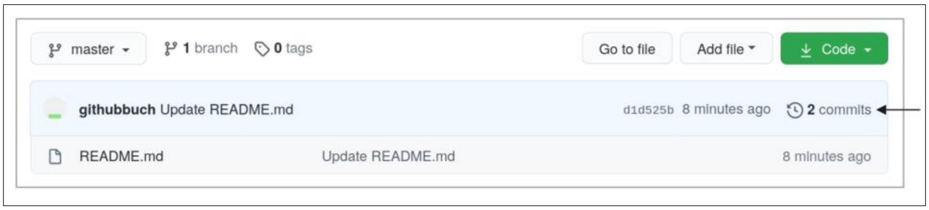
If you've enjoyed your actions enough, we should take a look at the statistics for our project (see Figure 3-13).

You may have noticed that the number of commits on your project has increased. No wonder, you just made your first one. But you may be surprised that there is the number *2* – after all, it was just your first and not already the second commit. We remember: When we created the repository, we told GitHub to create a README.md for us – that was the first commit on your repo. [15]

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

14 This is called »refactoring«.

15 By the way, this also applies if you create . *gitignore* or *license* (see Figure 3-7).



githubbuch Update README.md d1d525b 8 minutes ago 2 commits

README.md Update README.md 8 minutes ago

Figure 3-13: The total number *of commits can be found* on *the home page of a project*.

Congratulations, you have created your first repository and already made the first change. Now we want to look at issues and how to deal with them.

Practice the first process - create and edit issues

We already know from the section »Finding information (interested user)« on page 13 in Chapter 2 that issues should indicate the need for action (errors, requests for further functions – so-called feature requests –, comments, etc.). I described it as a kind of open letter. But an issue is much more than a simple letter. Think of it as a piece of paper on a bulletin board – everyone can read it and make their own comments. Often there are lively discussions in an issue, for example about the pros and cons of a new function.

The biggest advantage is that issues can serve as a kind of documentation for everyone (including future team members), as this is where all information on a specific topic converges, e.g. why a decision was made the way it was made.

Issues can be created by yourself, but in many projects they are also often created by other people (contributors). An issue can be concluded by the project owner, the maintainer or the contributor who created the issue.

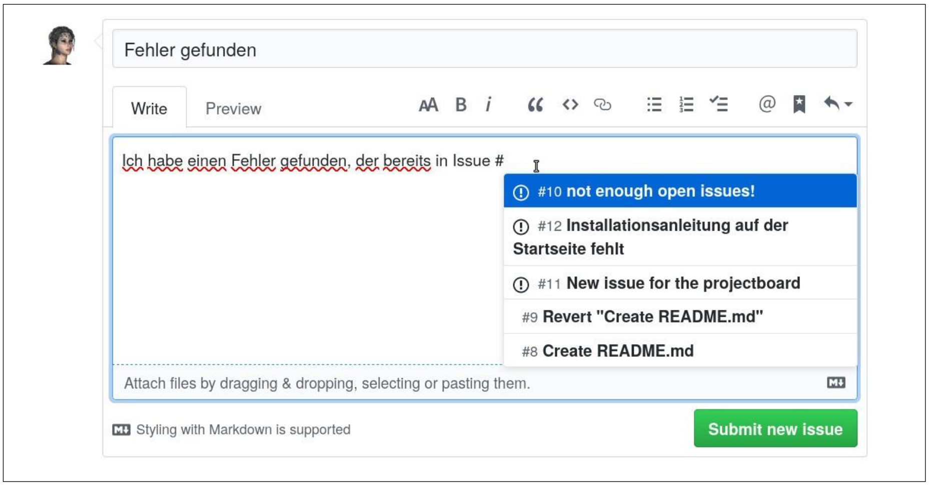
In order to practice how to deal with it, we create an issue in which we write down that we are missing an important piece of information on the start page. It can happen to you that other people come across your project and they lack any information. As a rule, they will announce this via an issue. But even if you find topics yourself that you still want to work on and notforget, creating an issue is a good thing (in Chapter 10, section "Own project boards – keeping track with Projects" on page 236, we will see that there are other ways to do this). On the one hand, you don't mess things up, and on the other hand , visitors to your site can see that you are well aware of the topic and will remedy it in the (hopefully) near future. [16] We shall:

1. create an issue ,
2. edit the issue and
3. close the issue .

Create an issue

Go to the *Issues* tab, select *New Issue* and give the issue a meaningful name and content, e.g. the one shown in Figure 3-14. [17]

Save your new issue by clicking on the *Submit new* german issue button . If you then select Issues in the project menu , the new issue you have just created will now appear.


Fehler gefunden

Ich habe einen Fehler gefunden, der bereits in Issue #

#10 not enough open issues!
#12 Installationsanleitung auf der Startseite fehlt
#11 New issue for the projectboard
#9 Revert "Create README.md"
#8 Create README.md

Write
Preview
Attach files by dragging & dropping, selecting or pasting them.
Styling with Markdown is supported
Submit new issue

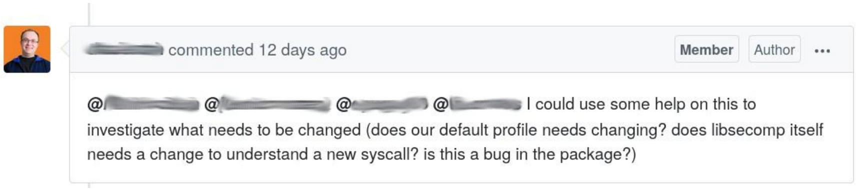
Figure 3-14: Creating an issue is quite simple by hand. The bigger challenge is to find a clever title and a good description.

On the picture I have added another special feature. Within an issue, it is possible to link to other issues or pull requests. This is especially useful if you want to show that another issue addresses a similar problem or a certain pull request could solve the problem (you can get to know pull-R equests in chapter 4). This link works by typing the pound symbol #, and GitHub supports us with a drop-down and a selection of options. However, this only works if an issue or pull request already exists.

16 It can even happen that someone else comes around the corner and solves the issue for you...

17 As you can see in the illustration, GitHub includes a few words. This is the spelling checker, which unfortunately does not speak a German.

There is also the possibility to address [18] people directly with so-called *@mentions*, who should, for example, perhaps take a look at the issue. They will then receive a notification that they have been mentioned. Illustration 3-15 is a real contribution in an issue from the *moby* project[19], which we were able to get to know briefly in chapter 2.


[redacted] commented 12 days ago

Member Author ···

@[redacted] @[redacted] @[redacted] @[redacted] I could use some help on this to investigate what needs to be changed (does our default profile needs changing? does libsecomp itself needs a change to understand a new syscall? is this a bug in the package?)

Figure 3-15: @mentions can be used to address *people directly. They will receive a corresponding notification.*

If you are a project owner or maintainer, you can assign issues to specific persons who are then responsible for the corresponding issue (English *asignee*). This responsibility can change over time, e.g. if the database specialist discovers during the search that the error is in a completely different place. If he cannot solve the error because of this, he may pass the issue on to the Main developer. Since your project is still relatively fresh and new, there is only one person to whom you can delegate the responsibility. [20]

In order to be able to find issues better or to make the urgency or importance of an issue clear, they can be classified with so-called *labels*. GitHub offers a number of ready-made labels by default (see Figure 3-16). However, it is also possible to create your own labels if it makes sense for the respective project. Labels will later help us to find other projects that we can support (see section »Looking for a foreign project« on page 117 in chapter 6).

We want to give the issue a label for practice purposes and assign it to you. There are two ways to do this.

- **Way 1**: You select the corresponding issue and get a corresponding menu on the right page (see Figure 3-17).
- **Way 2**: You go to the issue overview page (see Figure 3-18) and select the corresponding issue to 1 via the checkbox. After that, you can use the drop-down fields to perform the actions 2 + 3. This way has the great advantage that you can edit several issues at the same time.

18 The @ symbol is also spoken.

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

20 Not surprisingly: yourself !

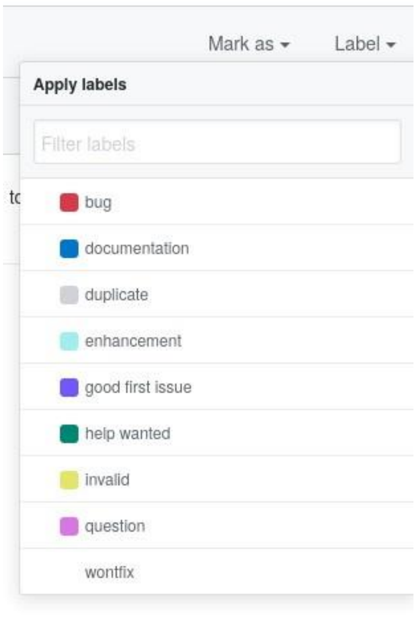
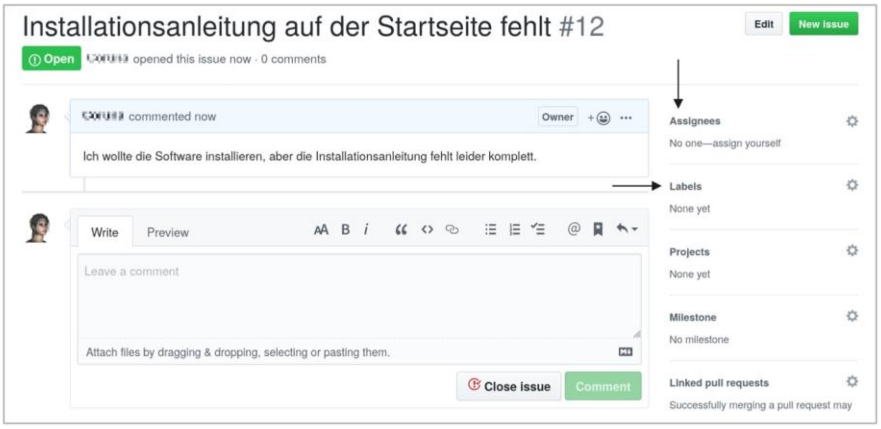


Figure 3-16: GitHub already offers some labels by default . Each label can have *a different color.*

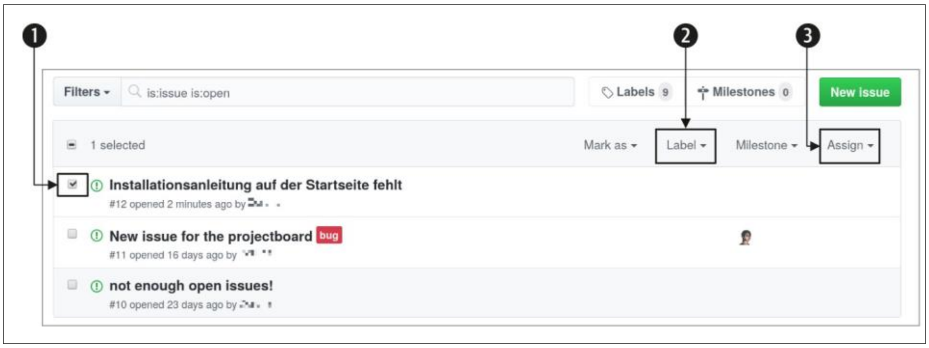

Installationsanleitung auf der Startseite fehlt #12

[redacted] opened this issue now · 0 comments

[redacted] commented now

Ich wollte die Software installieren, aber die Installationsanleitung fehlt leider komplett.

Figure 3-17: Label and assign issue，way 1: directly in the issue on the right side



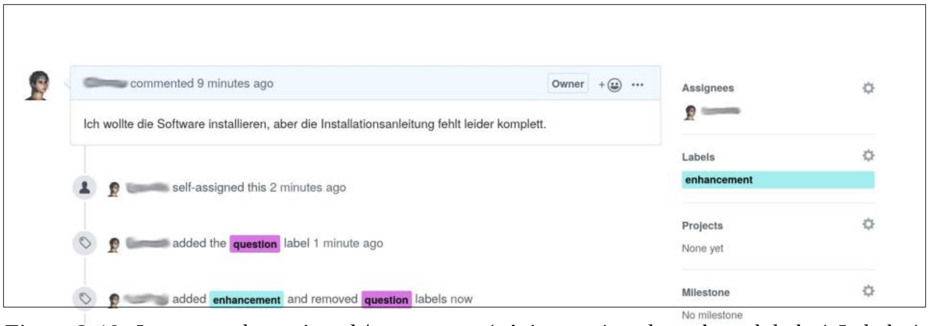
Installationsanleitung auf der Startseite fehlt
#12 opened 2 minutes ago by [redacted]

New issue for the projectboard bug
#11 opened 16 days ago by [redacted]

not enough open issues!
#10 opened 23 days ago by [redacted]

Figure 3-18: Label *and assign issues*, way *2:* in *the issue overview after clicking on the respective issue(s)*

Choose a label of your choice (e.g. *enhancement*) and choose yourself as *Anignee*. The whole thing could then look like figure 3-19. The assignment of a label or asignees is done by a simple click and does not have to be saved or confirmed separately. This can easily lead to accidentally assigning something falsch. As you can see in the picture, I first selected the wrong label *question* and then went to the right label *enhancement*. Here you can see very well that GitHub also documents every activity in an issue and makes it comprehensible for everyone – even if it should be an error.


[redacted] commented 9 minutes ago

Ich wollte die Software installieren, aber die Installationsanleitung fehlt leider komplett.

[redacted] self-assigned this 2 minutes ago

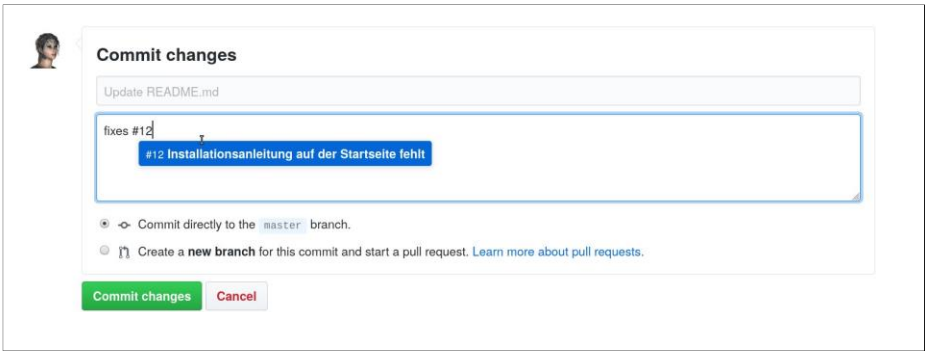
[redacted] added the question label 1 minute ago

[redacted] added enhancement and removed question labels now

Figure 3-19: Issues can be assigned to *persons (»Asignees«) and can have labels (»Labels«) in order to* be able *to classify them.*

Edit and close the issue

Now we want to solve the problem described in the issue by editing the start page accordingly (this is often called " solving the issue"). Go to the *README.md*, switch to editor mode and make a few changes. Before you save, type fixes #1 into the detailed description . Unless you have already done other things within your project, this addresses our just created issue (*1*). GitHub gives you via tool tip[21] a hint as to whether this is the case (see Figure 3-20). If you have already "played around" something, you may have to enter a different identification number (ID), as you can see with me with the number *12*. The ID of your issue can be found in the issue overview (see Figure 3-21).


Commit changes

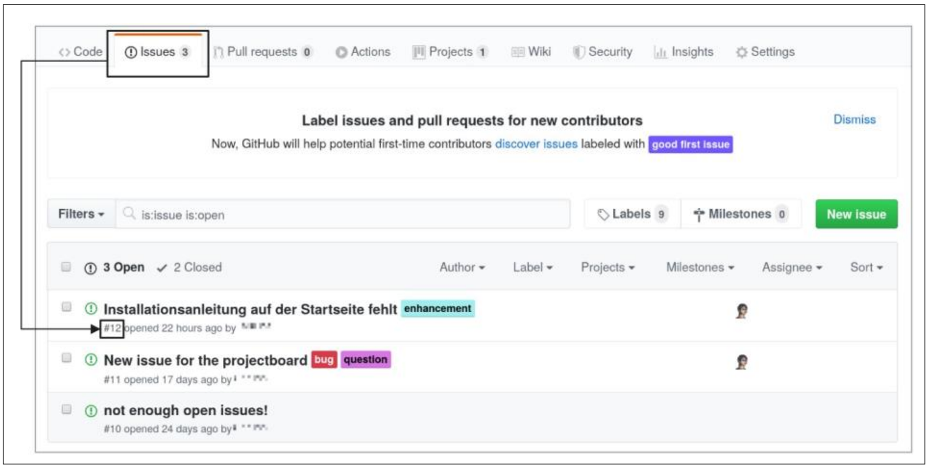
Update README.md

fixes #12

#12 Installationsanleitung auf der Startseite fehlt

Commit directly to the master branch.
Create a new branch for this commit and start a pull request. Learn more about pull requests.
Commit changes
Cancel

Figure 3-20: GitHub supports you with useful tooltips.



Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with good first issue

Installationsanleitung auf der Startseite fehlt enhancement
#12 opened 22 hours ago by

New issue for the projectboard bug question
#11 opened 17 days ago by

not enough open issues!
#10 opened 24 days ago by

Figure 3-21: The identification number (ID) of an issue can be found in the issue overview.

Once you have entered the correct ID, you save as before. But now something remarkable has happened. If you switch to the issue overview , you will see that the issue is gone?! Yes and no, the issue is not gone, but was automatically closed. The default filter ensures that only open issues are displayed, so you can no longer see it (remember

21 A small window that opens in programs or web pages and usually gives a short description or assistance.

Chapter 2, section »Finding information (interested user)« on page 13, here I had already explained the two standard filters *is:issue is:open*). If you change the filter to see the closed issues, you will see the edited issue again. If you now click on the corresponding issue again, you will get some more information, for example, there should be something similarto :

DeinUsername closed this in 7a0eeb2 2 minutes ago

The funny cryptic string (in my case 7a0eeb2) is provided with a link. This string is the so-called *commit ID*. Each commit you make (or others do) has its own ID to distinguish the individual com- mits from each other . Later in the chapter on Git (see Chapter 7) we will encounter the ID again.

Commit-ID

The commit ID on Git/GitHub consists of a so-called SHA-1 hash. It is an algorithm that produces a 40-line output quantity from an input set. This means that 40 lines always come out at the back – no matter how large the input quantity is.

The decisive feature here is that the exact same input always generates the exact same 40 characters. This has the advantage that you can see relatively quickly whether two commits are identical or not. Git/GitHub simply forms the hash value with two commit input sets (content, date, etc.), and if it is the same, both commits are identical.

Since 40 characters are quite long to display them constantly or to type them in somewhere, you usually only use the first seven to twelve characters. These are usually clear enough.

You can also close issues manually by scrolling to the bottom of the issue and selecting the *Close issue* button (german see Figure 3-22). Of course, this is not quite as great as the automatic closing, but sometimes there are issues that simply are not solved – for example, because some visitor has a completely lost feature about has tipped an issue that you don't want to include in your project. If this is the case, write in the comment why you will not implement the issue, not only because it is more polite, it also increases the chance that people will continue to create issues (if you want to). Nothing is frustrating than being "served" without justification.

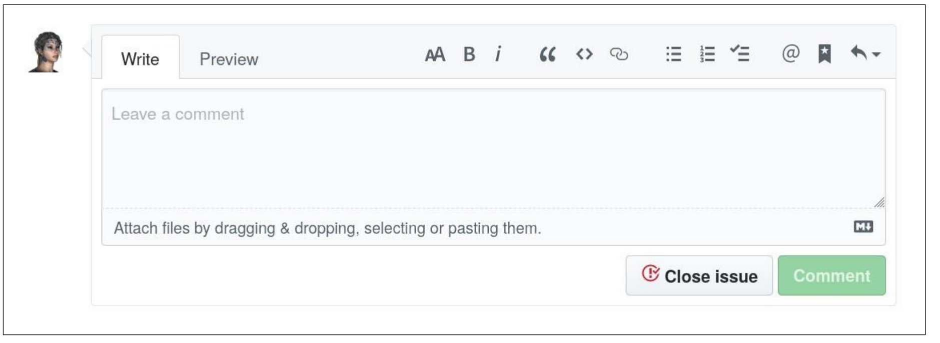
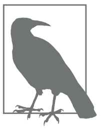

Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close issue
Comment

Illustration 3-22: Issues can also be closed *manually.*



Close resolved issues

It is good practice to close resolved issues . You keep track of things, and potential contributors don't try to find a solution and waste time.

Automatically close issues using keywords

GitHub offers the possibility to close issues via keywords . The keywords are as follows:

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

As soon as one of these keywords with a reference to the issue is entered in a commit (*#ISSUE_NR*), this issue is closed. You can also close several at the same time, but you have to have one of the keywords in front of each reference , such as *fixed #1*, *fixed #2*. [22]

You can also close issues in other repositories, provided you have the appropriaten permissions. This can be done with one of the keywords and as a reference *USERNAME/REPONAME#ISSUE_NR*.

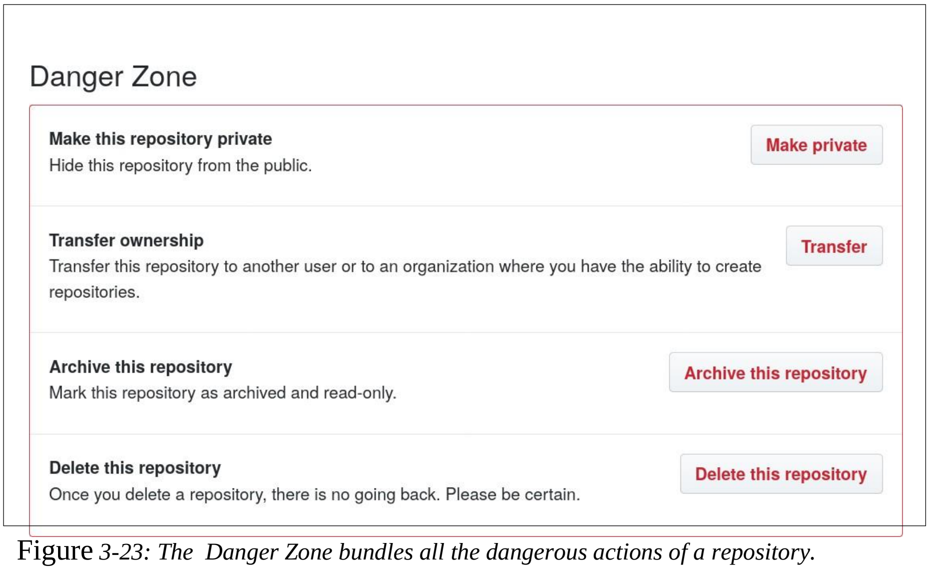
22 By the way, the whole thing only applies to the standard branch (master), if a commit to another branch takes place, the issue is not automatically closed. So far we have only been on the standard branch, so this info is only for completeness halber added. Don't worry, the topic of branches will become even clearer in the next chapter.

You now know how to create issues, label them and assign them to someone. As a cool new knowledge, you can now also close issues automatically when you work with certain keywords during a commit.

You may have already created one or more repositories for testing and playing around. To clean up your account a bit, I'll show you in the next section how to delete a repository.

Delete an existing repository

Deleting an existing repository is relatively easy, but at first glance a bit scary, because we have to go to the *Danger Zone*, see Figure 3-23. Go to your repository to be deleted and select *Settings*. To get into the *Danger Zone*, you have to scroll all the way down. Make sure that *Options* is selected in the left menu (should be the default when clicking Settings).



Danger Zone

Make this repository private
Hide this repository from the public.

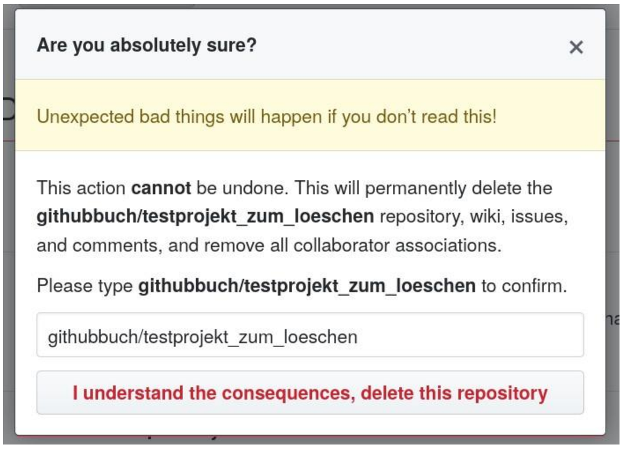
Transfer ownership
Transfer this repository to another user or to an organization where you have the ability to create repositories.

Archive this repository
Mark this repository as archived and read-only.

Delete this repository
Once you delete a repository, there is no going back. Please be certain.

Figure *3-23: The Danger Zone bundles all the dangerous actions of a repository.*

In the *Danger Zone*, select the *Delete this repository* button german. A warning message will appear asking you to type in the name of your repository according to the username/REPONAME pattern before deleting it by clicking on the button *I understand the conse- quences, delete this repository* (german such as "I understand the consequences, delete this repository") (see Figure 3-24). This is a security mechanism so that you can use your repository with all its contents(files, issues, comments, wiki, etc.) not accidentally deleted.



Are you absolutely sure?

Unexpected bad things will happen if you don't read this!

This action **cannot** be undone. This will permanently delete the **githubbuch/testprojekt_zum_loeschen** repository, wiki, issues, and comments, and remove all collaborator associations.

Please type **githubbuch/testprojekt_zum_loeschen** to confirm.

githubbuch/testprojekt_zum_loeschen

Figure 3-24: As a security mechanism, the *name of the repository must be entered beforehand for deletion*.

Now you know how to solve superfluous and possibly also tinkered repositories. Next (and last for this chapter) I would like to go through with you how to get an already existing own project, which is currently slumbering somewhere on the hard disk, on GitHub.

Upload an existing project

Uploading an existing project is simple. Create a new repository for your project. Click the *Add file* button and then click *Upload files (*see also Figure 3-25). Then you have the option to drag and drop or click on the *choose your files* link to select and upload the desired files and/or folders (see also Figure 3-26).

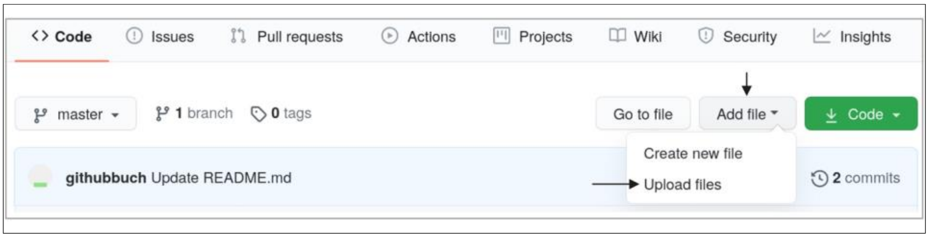


Figure 3-25: Uploading files and folders is done via Add file and Upload files.

In the end, all you have to do is commit and the files and/or folders are uploaded to your repository. Please note: The files must not exceed the size of 25 MByte per file! If you want to upload larger files, you have to switch to the console (you will learn this in chapter 8).

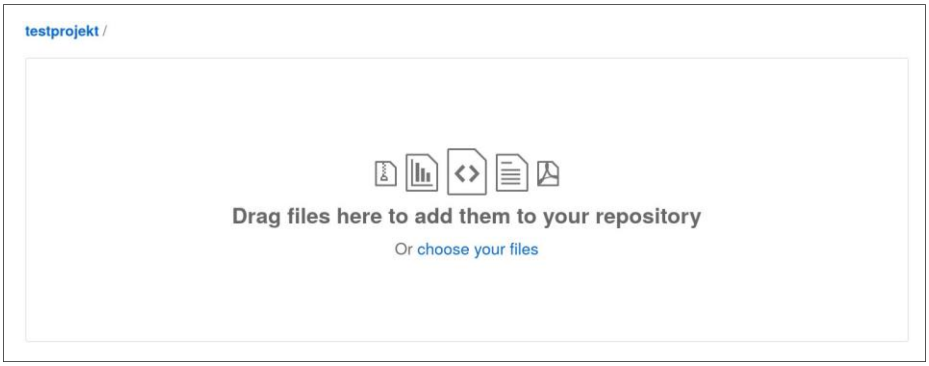


Figure 3-26: Drag-and-drop makes it easy to add *files and folders.*

Now you have the basis to take the first steps accident-free with GitHub. But in order to work really productively with others, a few things are still missing . Since GitHub focuses on collaboration (we remember chapter 1, section »Areas of application of GitHub« on page 3), we will deal in the next chapter with what to doif suddenly others participate in our Contribute to the project.