
GitHub for Technical Writing

发布 1.0.0

Zhang Yan

2021 年 01 月 21 日

目录:

1	Github 简介	1
1.1	楔子	1
1.2	Github 是什么	1
1.3	Github 在哪里	1
2	Github Desktop 安装和使用	3
2.1	为什么使用 Github Desktop	3
2.2	如何使用 Github Desktop	3
2.3	如果你不喜欢 Github Desktop	6
3	创建第一个仓库	7
3.1	什么是仓库	7
3.2	创建本地仓库并推送到远程	7
3.3	创建远程仓库并从本地拉取	10
4	GitHub 基本操作	13
4.1	提交到本地分支	13
4.2	提交到远程分支	15
4.3	什么是分支	15
5	GitHub 多人协作	17
5.1	邀请你的队友	17
5.2	再谈分支	18
5.3	克隆仓库并新建分支	18
5.4	合并分支	20
6	Chapter6 Issue 和 Wiki 的使用	23
6.1	Issue	23
6.2	Wiki	25
7	Github 文档发布	27
7.1	Github.io	27
7.2	发布	27

1.1 楔子

本教程意图可以让没有技术背景的同学也能快速准确地使用 `github` 这个方便高效的工具进行文档代码化开发。故而对于 `Git` 工具和 `Github` 这一仓库托管网站不再进行概念上的区分。

我们一般会在自己的本地电脑上进行文档写作，然后保存在本地。这种传统的写作方式有自己的弊端。考虑这样一个问题，假如今天我出门在外，但是有一个突发的任务需要我修改文档并提交，手里没有保存文档的那个电脑。或者这样一个问题，当我修改完一章之后，发现写作任务被修改了，需要改回当初写作的那个版本。或许我们会很容易的想到解决方法，把这个文档每次写完后存到网盘，为每一次修改都保存一个版本。你可能觉得这样会很麻烦，但是如果有一个网盘工具能帮助你记录这些东西，你只需要动动手指就可以实现上面提到的功能，是不是很酷！`Github` 以及围绕它的一些软件，就是这样一个东西，而且，它远远不止这些东西！

1.2 Github 是什么

初学者可以把 `Github` 理解成一个云端的网盘。每次你在本地完成了写作，把文件上传到远程的网盘里，它都会为你忠实的记录你的每一次修改，方便你看到最新的工作进程。同时他还能将你的工作恢复到任何有提交记录的版本。这些特质，让 `Github` 在多人协作过程中，也能展示它强大的能力。

1.3 Github 在哪里

直接在浏览器中输入 `www.github.com` 或者使用搜索引擎直接搜索 `github` 进入官方页面。



Github 向我们详细展示了 Github 应用场景和基本功能。我们需要做的第一步就是加入 Github。通过 Sign Up 按钮，注册 Github 账号。注册完毕后登录就可以开始使用 Github 了。

Github Desktop 安装和使用

2.1 为什么使用 Github Desktop

有技术背景的程序员在使用 Github 的时候非常不愿意使用 Github Desktop，他们对于黑框命令行有着偏爱。对于技术文档写作人来说，大可不必大费周章的去学习 Git 命令，对着黑框敲击。使用 Desktop 就像使用一个软件一样使用 Github。就像它的主页打出的口号：Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

2.2 如何使用 Github Desktop

访问 <https://desktop.github.com/> 下载软件。遇到被墙的问题请自行解决。

下载安装后，使用之前注册的账号登录 Github Desktop 并授权。

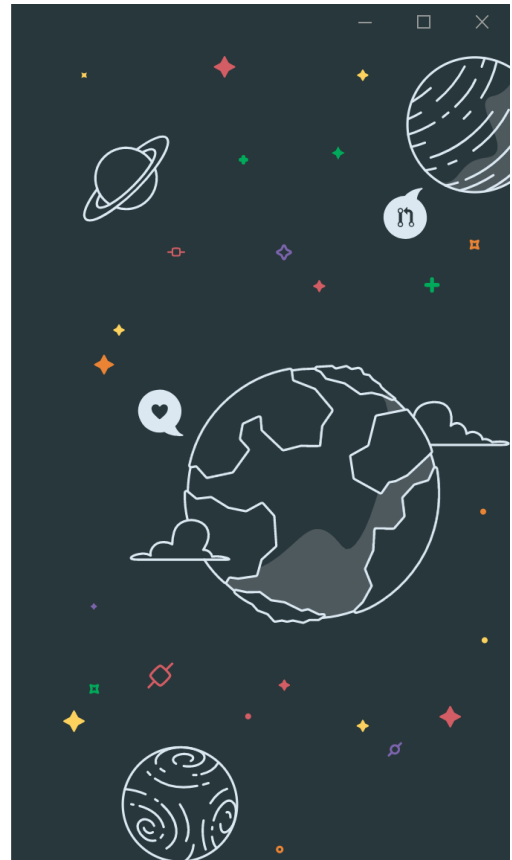
Welcome to GitHub Desktop

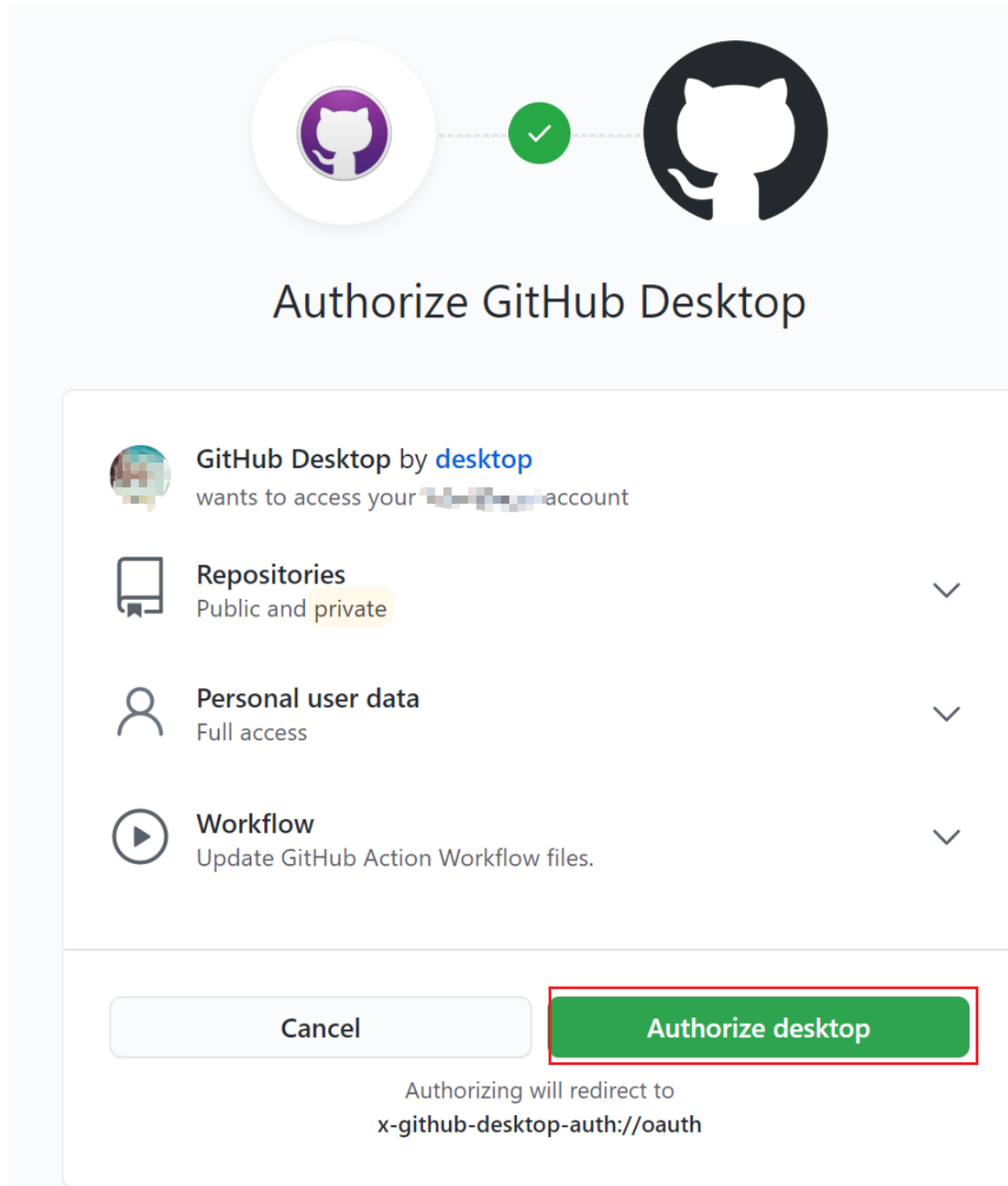
Your browser will redirect you back to GitHub Desktop once you've signed in. If your browser asks for your permission to launch GitHub Desktop please allow it to.

[Sign in to GitHub.com](#)

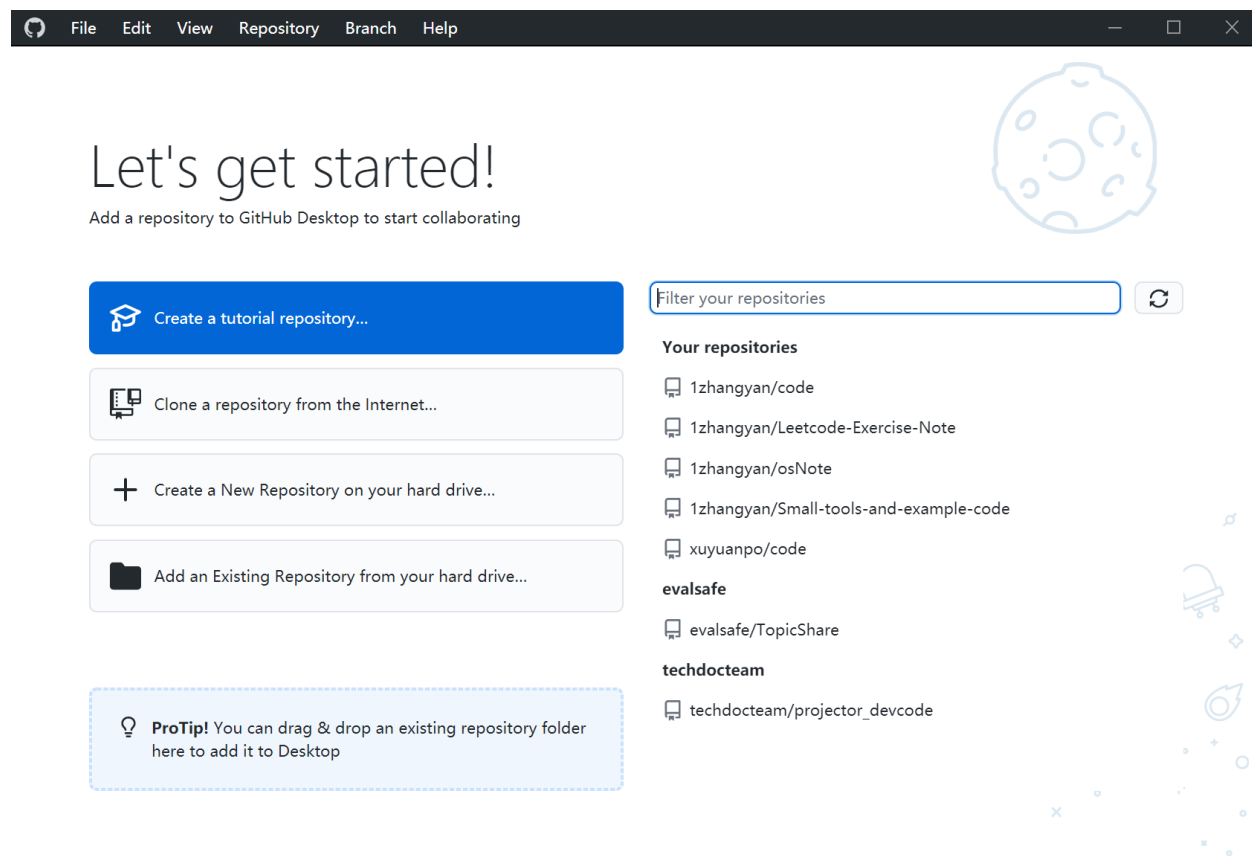
Cancel

Skip this step





就可以进入 Desktop 的界面，开始使用它了。



如果你偏好中文访问这个网址 https://github.com/lkyero/GitHubDesktop_zh 可以给你一个汉化的 Desktop。

2.3 如果你不喜欢 Github Desktop

这本教程尽可能以简单易操作为宗旨来进行介绍。如果你偏爱命令行，偏爱程序员方式地使用 Git 以及 Github，那么这里有另外一份广受好评的教程供你参考。 <https://www.liaoxuefeng.com/wiki/896043488029600>

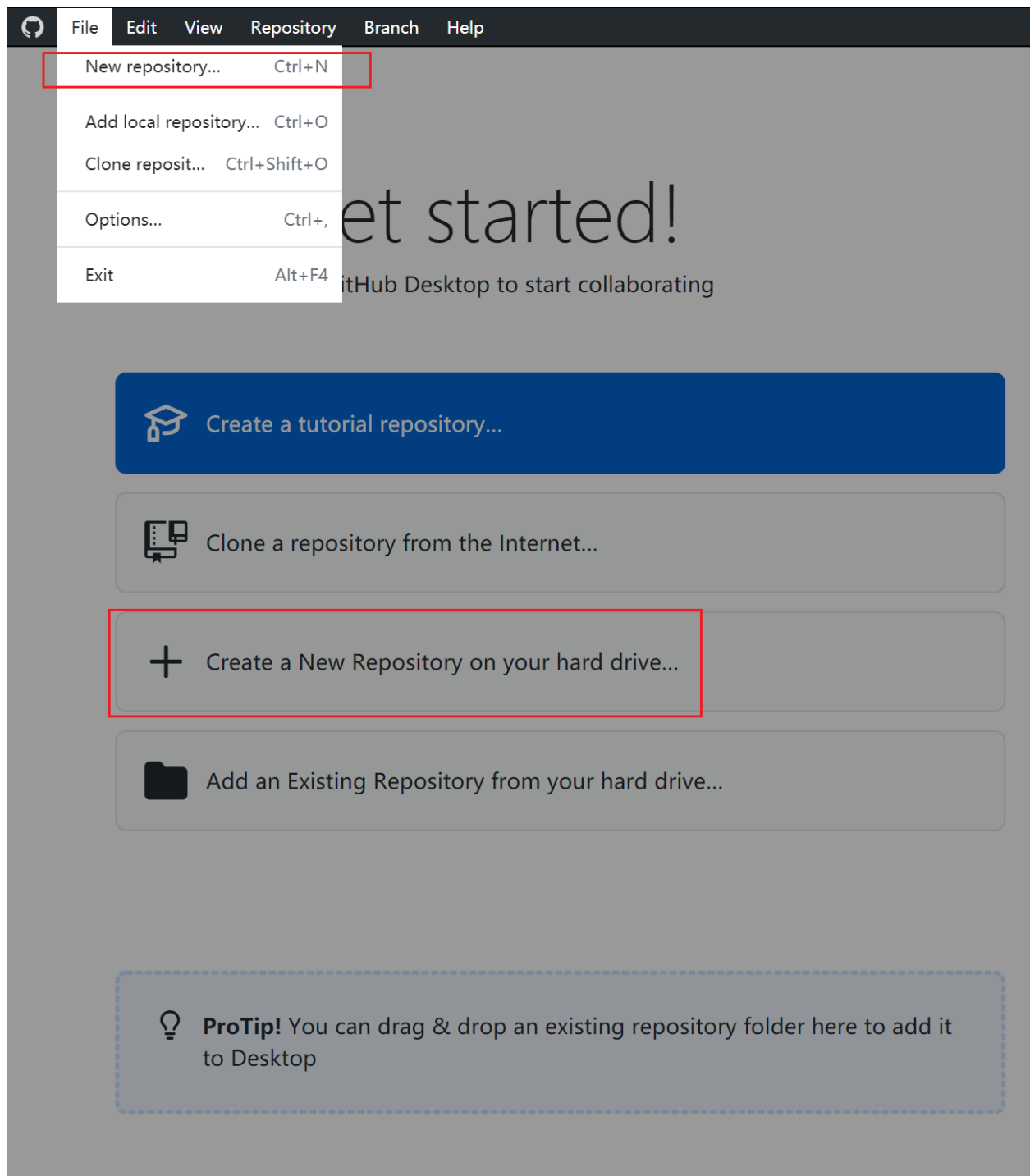
创建第一个仓库

3.1 什么是仓库

一般来说，我们开发一个文档，会把有关这个文档的所有内容，用到的资源，保存在一个地方。一个仓库就是一个这样的文件夹，这个文件夹中保存了你写的 markdown 文件，保存了你需要用到的图片，音频视频等等。他就相当于一个大的项目文件夹。Github 中每一个仓库就对应了一个项目文件夹。而我们对项目的创建，更新，删除，都是使用仓库（repository）作为单位的。

3.2 创建本地仓库并推送到远程

我们知道，Github 有点像远程的网盘。自然而然地，我们可以想创建一个项目的时候会想到先在本地新建文件夹，然后再上传到云端。这当然是一个正确的思路，但是 Github 仓库虽然表现为一个文件夹，但是它有自己的配置文件，不是任何时候创建的文件夹都可以成为一个仓库，可以直接提交到远程服务器。我们使用 Github Desktop 帮助我们完成这个过程。首先打开 GithubDesktop。



找到选项 New Repository 或者 Create a New Repository on your hard drive... 点击。

Create a new repository

Name

repository name

Description

Local path

C:\Users\18810\Documents\GitHub

Choose...

☐ Initialize this repository with a README

Git ignore

None

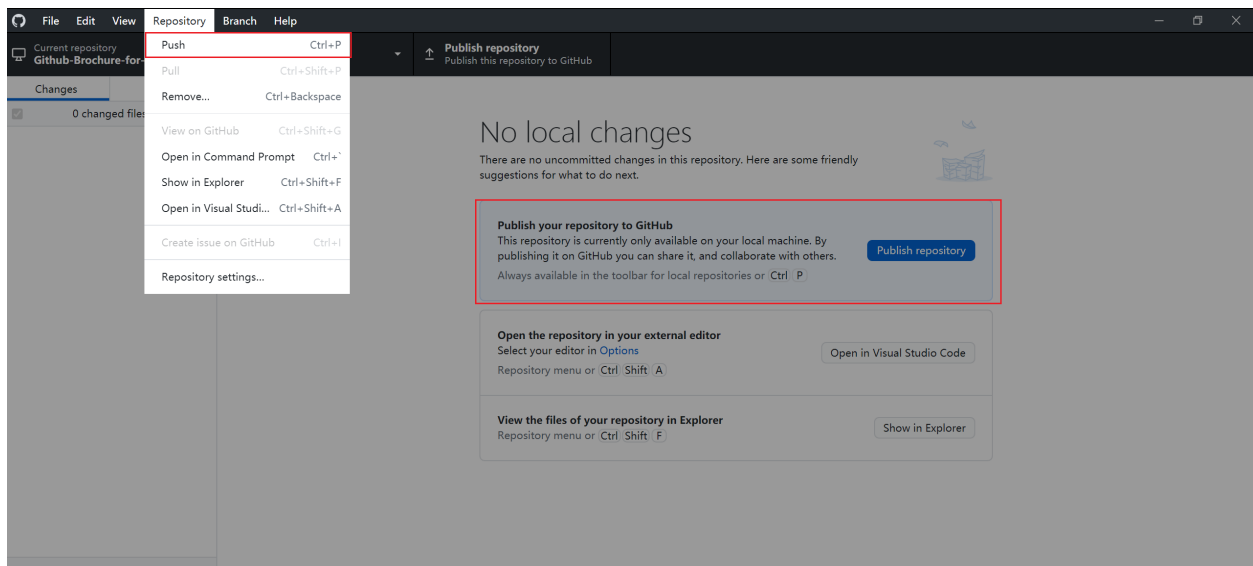
License

None

Create repository

Cancel

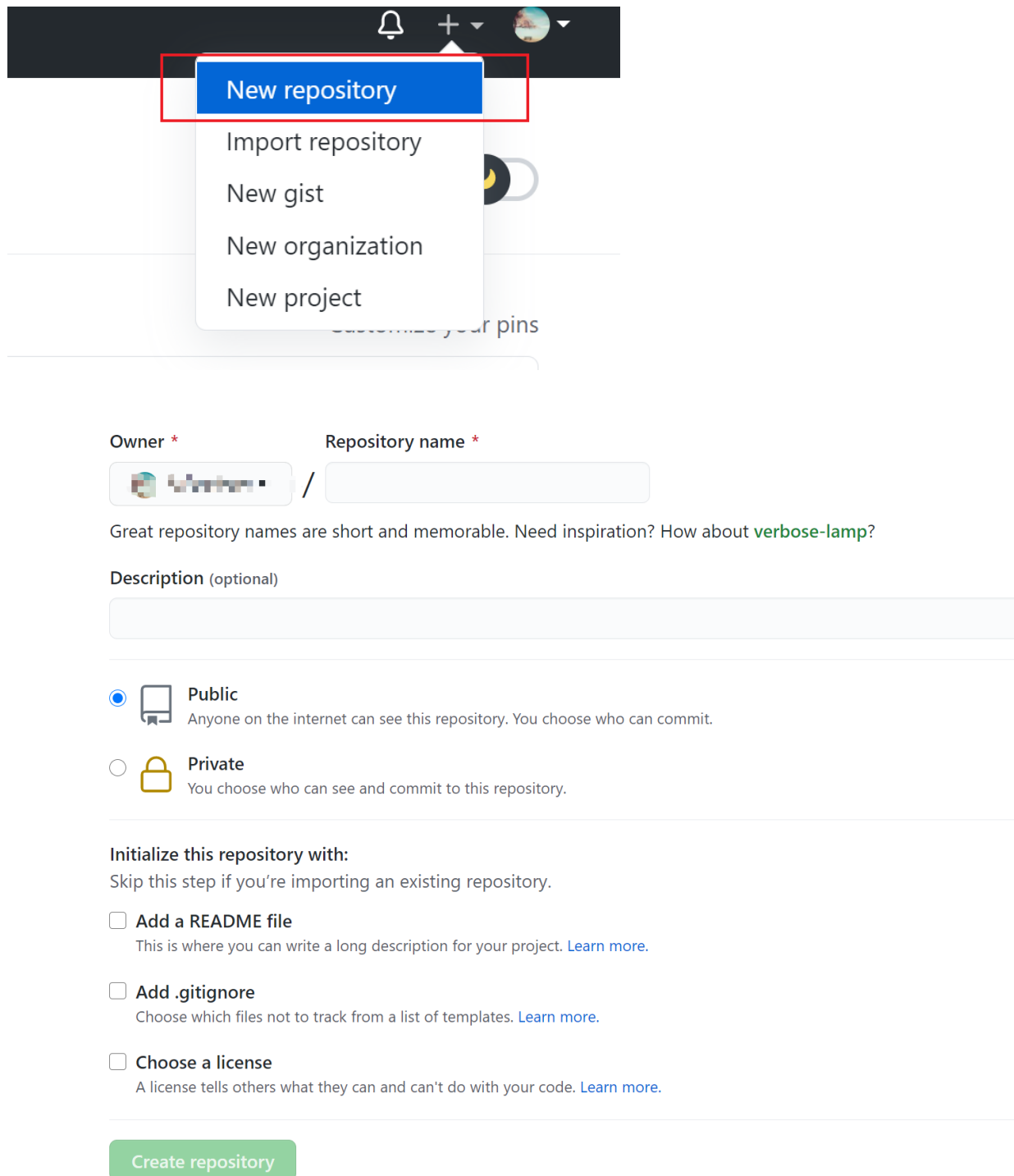
按照提示填写仓库名和仓库在本地的存储位置。完成后，一个初始化完成的本地仓库就创建好了。但是，这时候的仓库只存在于本地，而不存在于云端，所以我们选择 `push` 选项将仓库推送至云端。



这样，在 Github 服务器上和本地都有一份项目的文件夹了。以后，你每次提交以后推送都会让远程的仓库和你本地的仓库保持一次同步更新。这样你在任何地方都能继续你的工作，而不是仅仅局限于你存储项目文件的电脑。

3.3 创建远程仓库并从本地拉取

你当然也可以在远程的网盘创建一个项目文件夹，然后把这个文件夹下载到本地来！访问 Github 官网并登录。找到创建仓库的按钮。填写相关信息后创建。



New repository

Import repository

New gist


New organization


New project

Owner * Repository name *

Great repository names are short and memorable. Need inspiration? How about **verbose-lamp**?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

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

Initialize this repository with:

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

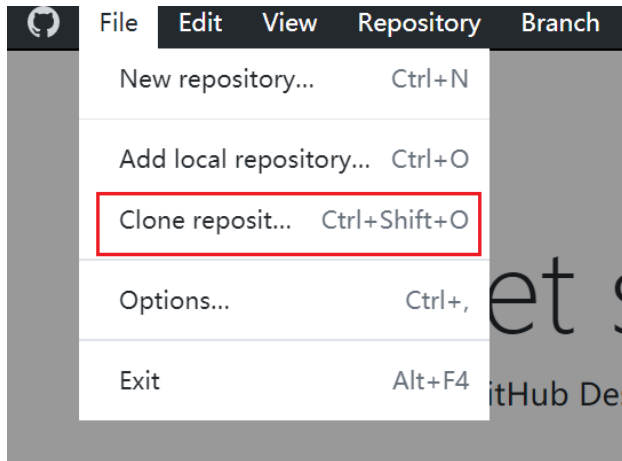
☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

创建完成后打开 Github Desktop 选择 Clone Repository。



选择自己新建完成的仓库 clone 就可以得到一份和远程仓库相同的本地仓库。

4.1 提交到本地分支

不同于传统文档开发流程中的做法，在 **GitHub** 中，对仓库的修改是一件非常重要严肃的事情，我们的每一次修改都需要被记录下来，这样我们就可以随时找到我们之前的想法，随时让仓库恢复之前的状态。在我们对代码进行修改的时候，我们并不能完全依赖软件自动帮我们记录所有的东西，那样会增加许多无聊而又烦杂的数据。所以，**Git** 把这个自主权交给了我们。我们在完成一个阶段的工作以后，可以选择把自己认为有用的所有修改（包括你增加的内容和删除的内容）记录下来，我们把这个过程叫做提交（**commit**）。你当然可以每修改一次就提交一次，但是会很累。建议每次工作结束时提交，或者在进行重要的工作时频繁提交。

The screenshot shows the GitHub Desktop application interface. At the top, there's a menu bar with File, Edit, View, Repository, Branch, and Help. Below the menu bar, the current repository is 'Github-Brochure-for--Tech...' and the current branch is 'main'. A 'Fetch origin' button is visible, indicating the last fetch was 6 minutes ago.

The 'Changes' tab is selected, showing 2 changed files: 'c:\Chapter3 创建第一个仓库.md' and 'images\create_rep7.png'. Both files have checkboxes next to them, which are highlighted with a red box and a red arrow pointing to a red text label: '勾选需要提交更改的文件' (Check files to be submitted).

The 'History' tab is also visible, showing a list of commits. A red box highlights a specific commit, and a red arrow points to a red text label: '这个区域显示了被修改文件的具体修改内容' (This area shows the specific modification content of the modified file).

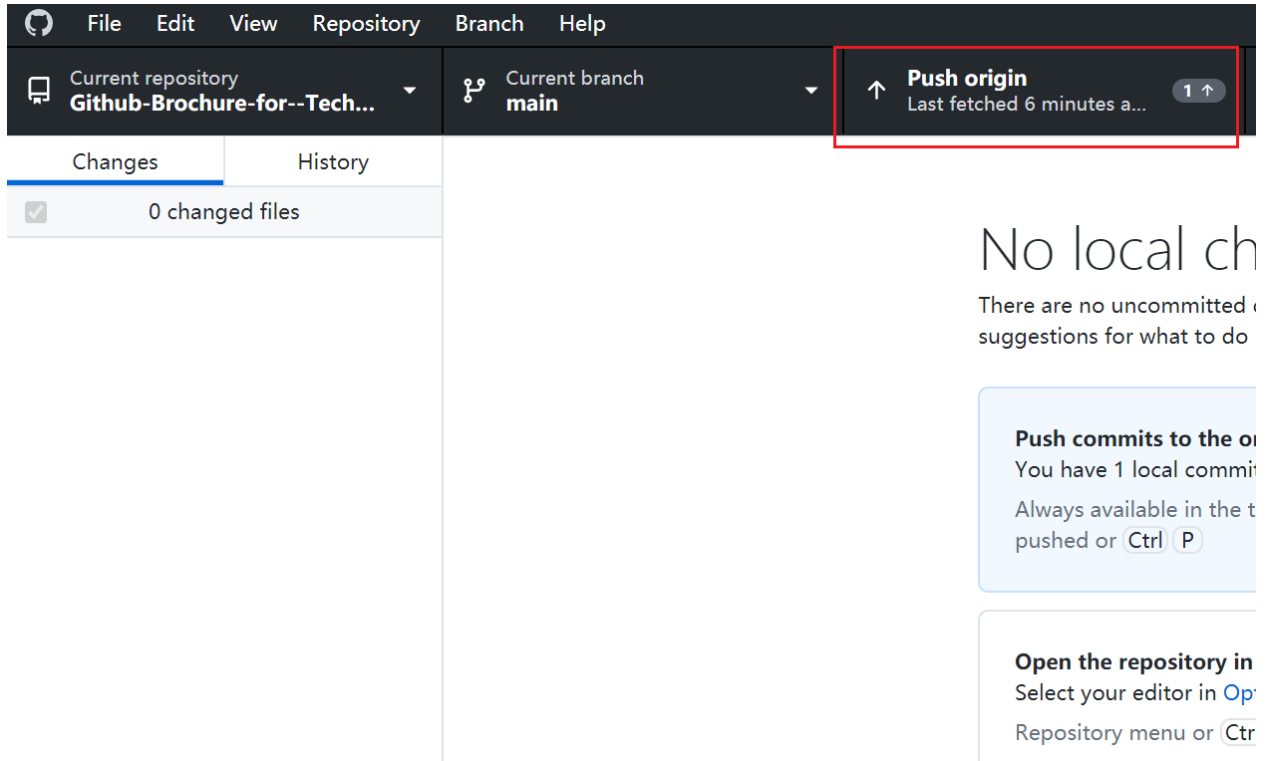
At the bottom, there's a 'Commit to main' button, which is highlighted with a red box and a red arrow pointing to a red text label: '提交按钮' (Commit button).

Below the commit button, there's a 'Summary (required)' field, which is highlighted with a red box and a red arrow pointing to a red text label: '对于本次提交的描述' (Description for this commit).

在 GitHub Desktop 中，你的每次修改都被详细地显示出来。在提交之前，你需要勾选出需要提交的文件，一般来说我们会把所有的文件都选上，避免遗漏。但是这个区域存在的目的是为了告诉你，你是你作品的主宰，你决定着哪些修改会被提交。勾选出需要提交的文件后，在 Summary 的选项框中，填写本次提交的概述，最后点击 Commit 按钮，这样就可以把本次所有的修改提交到分支上了。这里我们暂时不用对分支进行什么操作，所以我们把分支放到最后解释。

4.2 提交到远程分支

我们在上一个步骤已经将自己的修改提交到了本地的 `main` 分支上了。说人话，就是我们已经把修改好的文件保存在本地了。但是远程仓库不会自动更新它的内容。就像你的百度网盘永远不会存下你没有上传的文件。所以，保证远程仓库和本地仓库一致，还需要一个步骤，推送本地分支到远程。



就像图里那样，当你已经进行了提交操作，右上角的红框中会自动显示 **Push origin**，点击它，它会把本地的分支推送到远程仓库，从而保证你在 **GitHub** 网页上看到的那个仓库和你现在本地的仓库一模一样！

4.3 什么是分支

我们在上面的介绍中频繁提到了分支，那么，什么是分支呢？我们打开 **DeskTop** 左边的 **History** 选项卡。

The screenshot shows the GitHub web interface for a repository named 'Github-Brochure-for--Tech...'. The current branch is 'main'. The top bar indicates the repository is up-to-date with the origin. The left sidebar shows the commit history, with the 'History' tab selected. The main area displays a diff for the file 'Chapter4 GitHub基本操作.md'. The diff shows changes between commit d741efa and the current commit. The changes include adding a new section titled '## 提交到远程分支' and updating the 'Summary' section. The diff is highlighted with a red box, and the commit history sidebar is also highlighted with a red box.

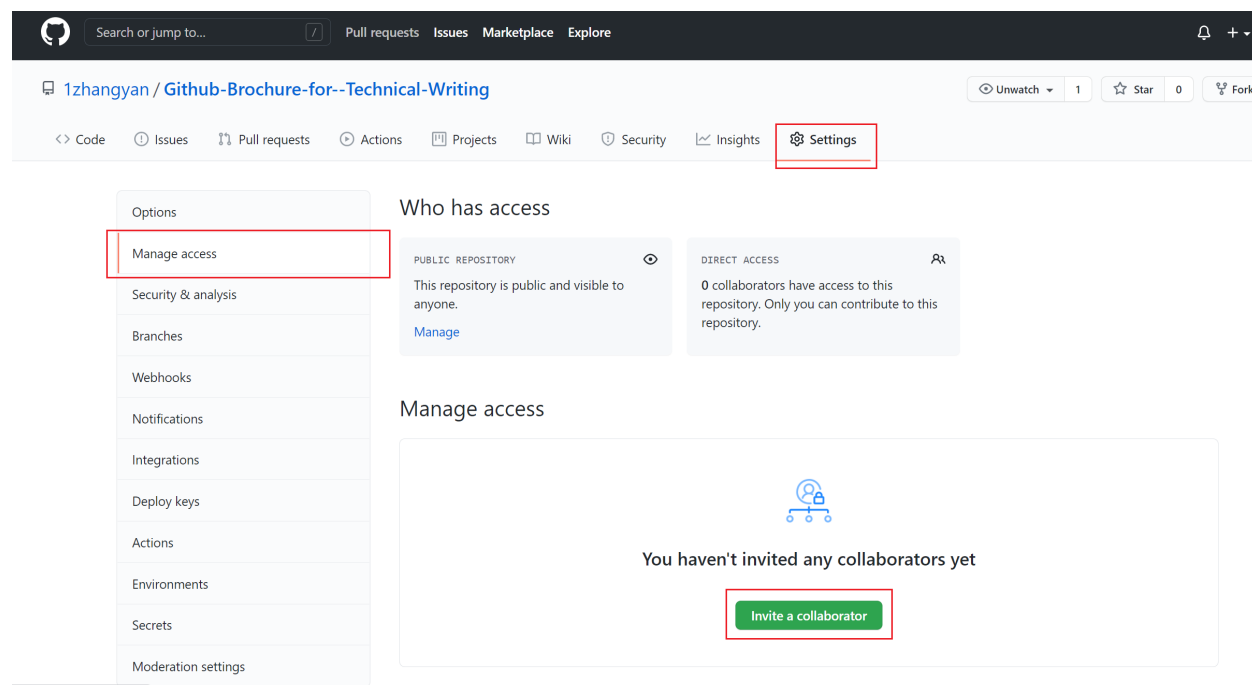
这个区域显示出仓库从创建到目前所有的提交

这个区域显示每次提交涉及的文件和修改的具体内容

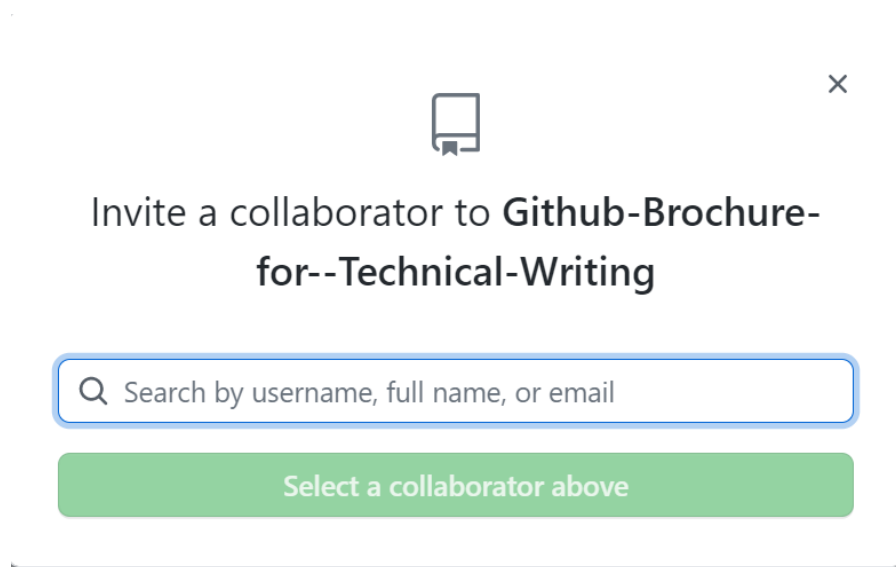
可以看到 **History** 展示了从仓库创建以来到目前为止，我们对项目的所有提交。这一个个的提交就构成了我们的工作流，分支就是记录了我们工作流的一个东西。我们之所以之前没有对分支产生什么具体的印象，那是因为，我们每个仓库都有一个默认的分支，叫 **main**（之前叫 **master**）。我们的工作流都默认被记在 **main** 分支上了。但是一个仓库可以建立多个分支。你可能会认为，一个分支就已经足够我使用了，为什么要建立多个分支呢？当你的文档需要在不同的情况下显示不同的内容，或者你需要组团开发一个文档的时候，你就会发现分支的强大！

5.1 邀请你的队友

传统的文档协作过程中，可能在任务分配后每个人完成不同的任务，最后再通过拷贝文件或者网络传输到一台机器上进行汇总。**GitHub** 不希望你做的这么麻烦，他允许你和你的队友们将仓库克隆到自己本机，进行修改后再将各自的工作推送到远程仓库。最终在远程仓库中讲你们的工作合并。那么首先，你得需要队友。



选中 **GitHub** 仓库，点击 **setting** 选项卡，再选中 **ManageAccess** 选项，打开后点击 **invite a collabrator**，添加你的队友。

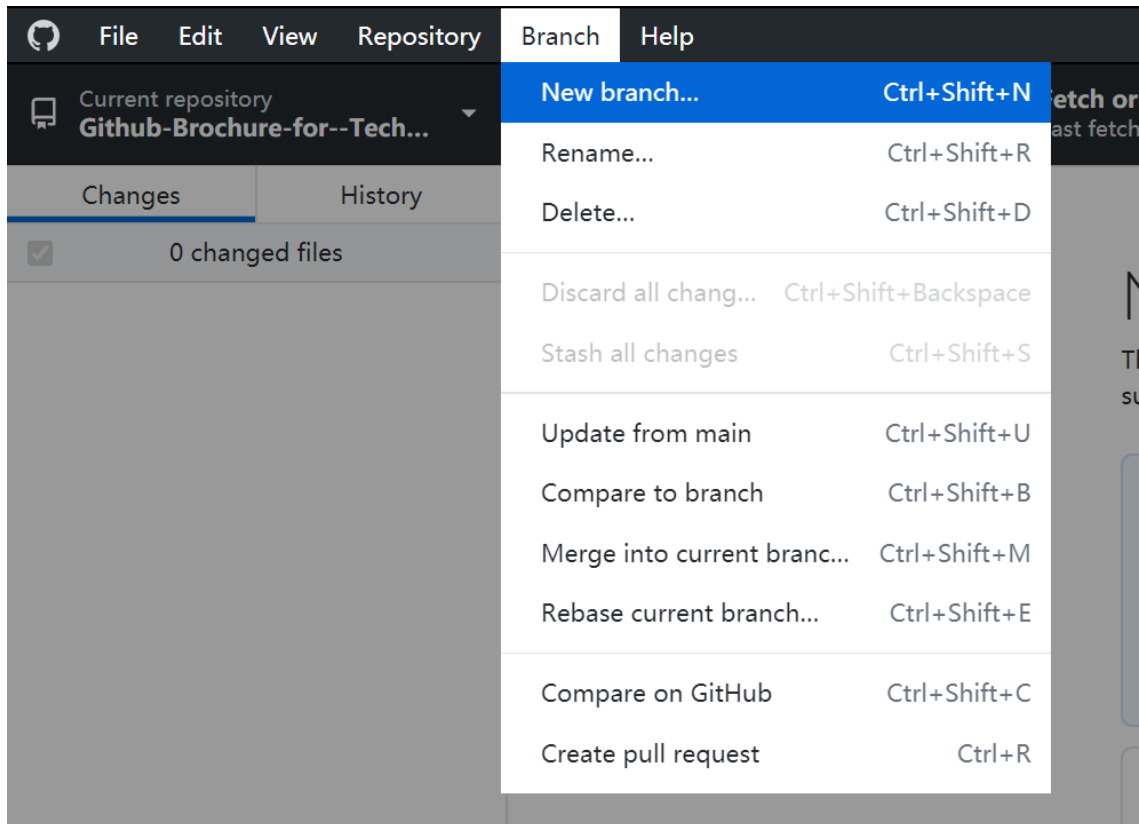


5.2 再谈分支

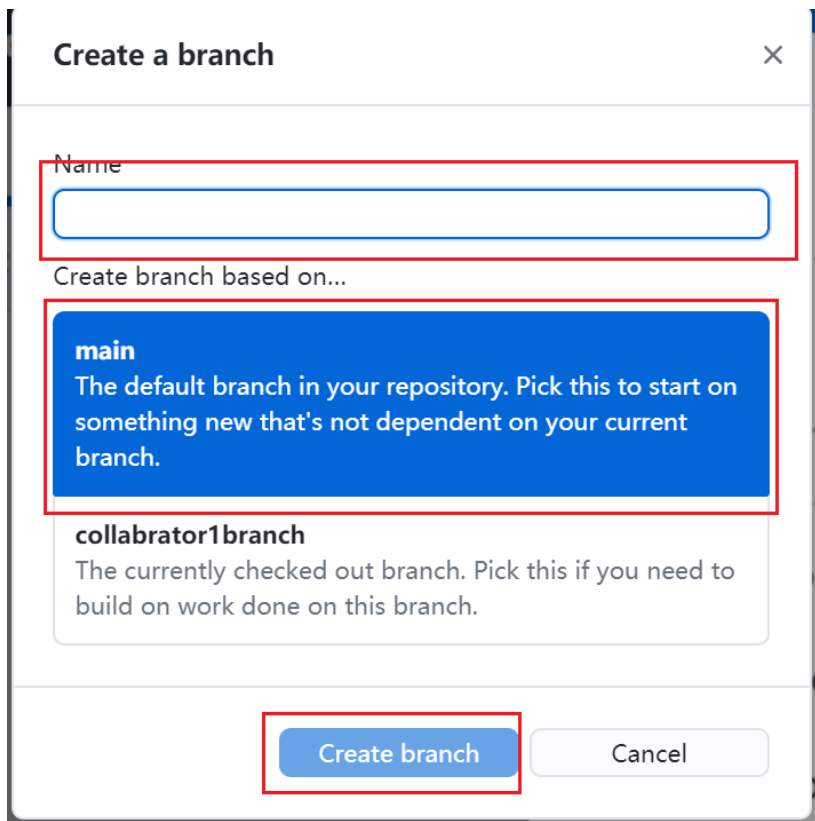
我们知道我们的提交默认都提交到了 `main` 分支上。当我们多个人协作的时候，无法保证每个人每时每刻的本地内容和远程内容都相同，这就使得如果使用单一的 `main` 分支，每次提交之前需要将远程的内容同步到本地。如果操作不当，本地的修改可能被覆盖。同时这样每次提交都需要考虑这些内容，就会使协作多出很多不必要的麻烦。但是如果每个人都建立一个分支呢？你和你的队友们各自在各自的分支上工作，保证不动别人的分支，你们只管负责自己的部分，等到所有人都结束整个项目后，再把所有的分支都汇总到一起，汇总到一个分支上来，是不是就极大地减少了合作期间的交流成本？

5.3 克隆仓库并新建分支

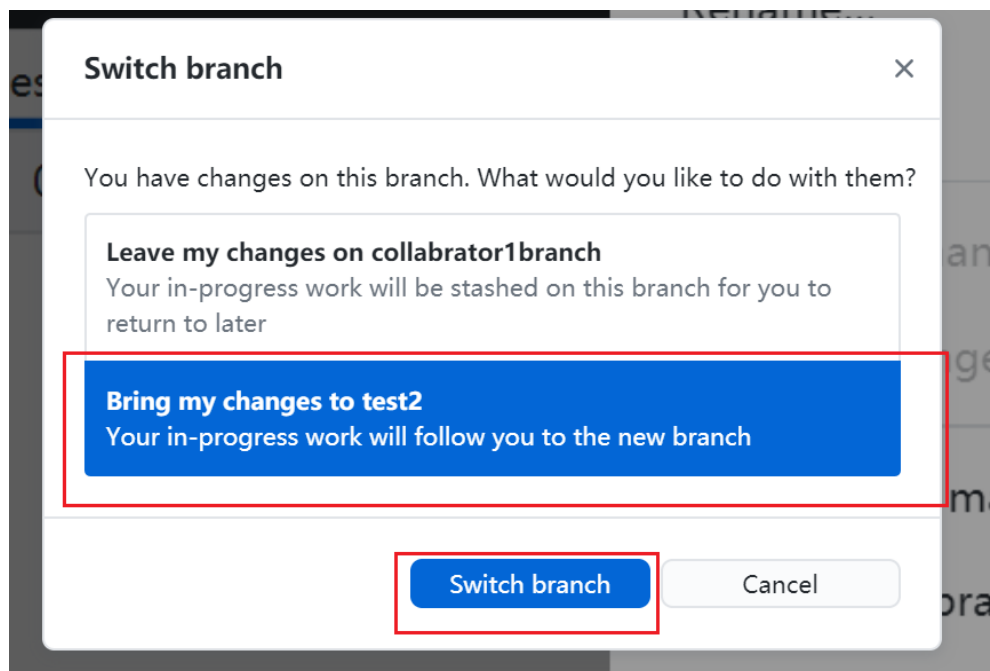
在第三章中创建远程仓库并从本地克隆这一小节中，我们学习了如何把远程的仓库克隆到本地。下面我们介绍如何新建分支找到 `New branch` 选项卡。



按照提示创建分支，所谓分支就是从一个分支中分出来的，必然需要一个模板分支来创建，一般选择 main。



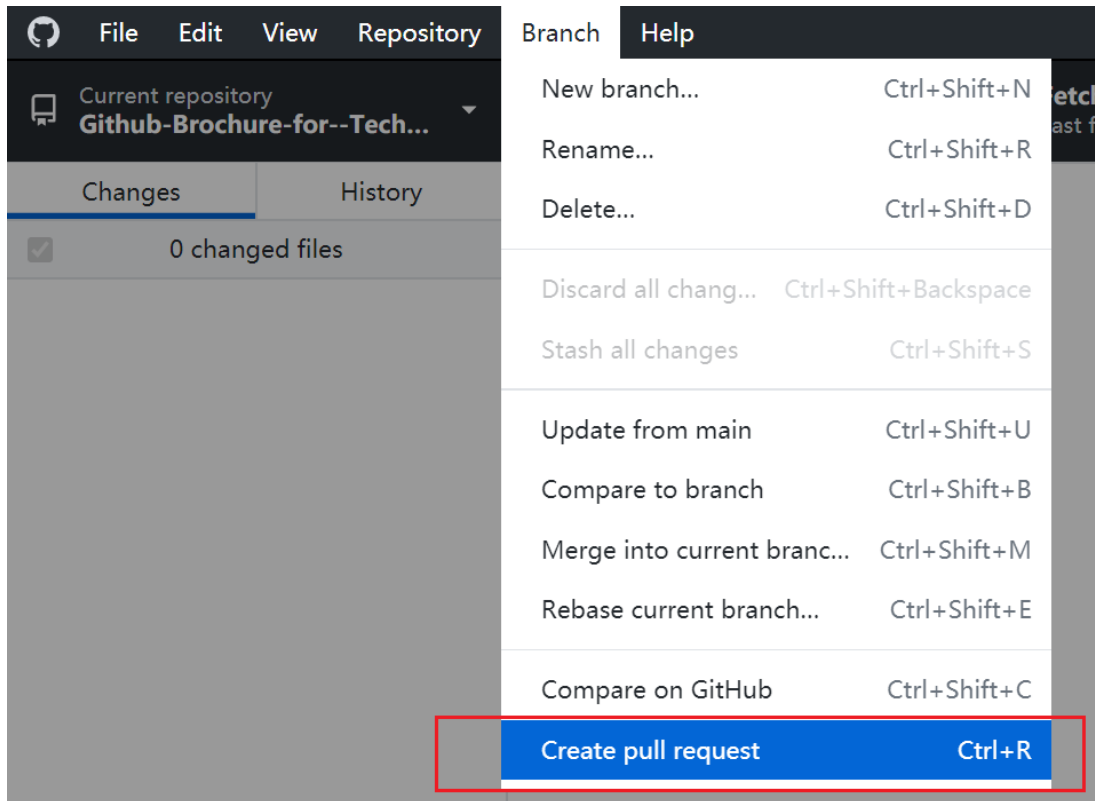
当你创建分支的过程中发现当前分支已经有修改的内容了，但是没有提交，需要勾选下面的选项，将修改带到新的分支中去。



这样就创建了新的分支，每个人都可以在分支上工作而互相不打扰了。

5.4 合并分支

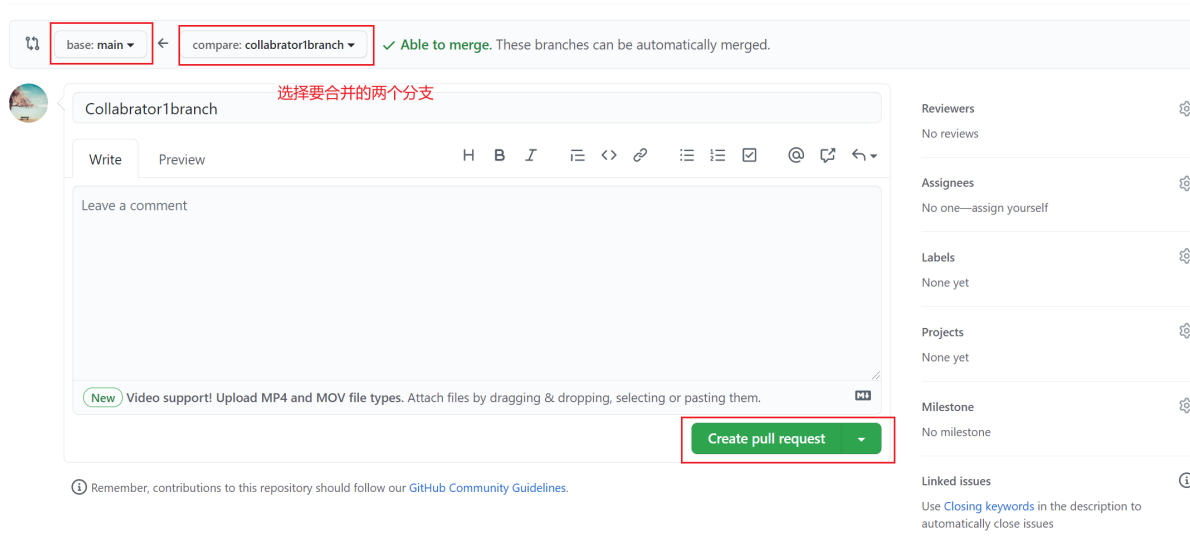
尽管我们有很多种方法进行分支合并，但是我仍然建议使用 pull request 这种方法。找到 Pull request 选项卡，点击。



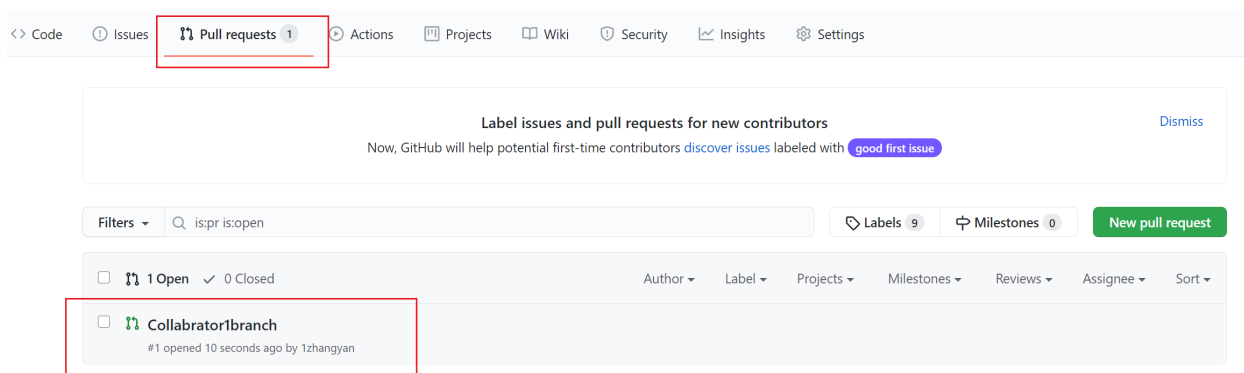
之后跳转到 **GitHub** 页面，选择需要合并的两个分支，你可以为这次合并添加一个描述，描述具体内容，而后创建这次合并请求。

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

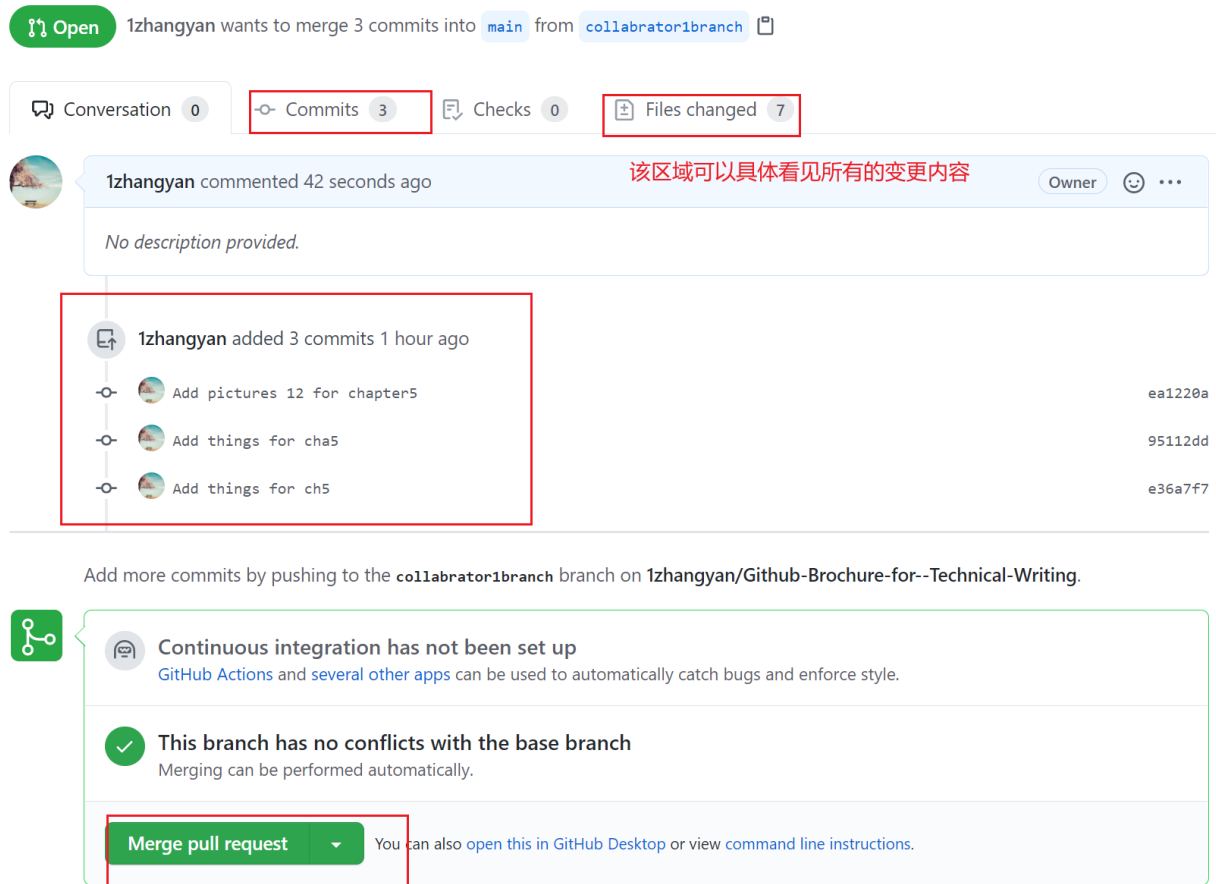


创建合并请求之后，在仓库的 **Pull Request** 选项卡中，所有的合作者都可以看见这个请求。



这就意味着所有的人都可以审阅这次请求，帮助提交者确认是否有错误。如果没有错误，那么由管理员合并分支，完成这次请求。

Collabrator1branch #1

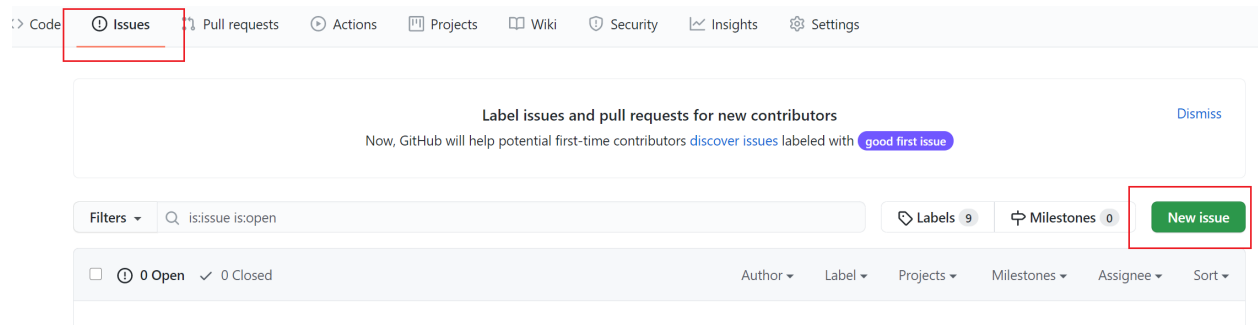


这样一个标准的合并流程便完成了。通过这种方式，可以让团队开发效率变得更高。

Chapter6 Issue 和 Wiki 的使用

6.1 Issue

当我们发现仓库中的代码有什么疏漏或者错误的时候，我们可以提 Issue，帮助小组或者自己记录要解决的问题。在仓库的 Issue 选项卡中，找到 New Issue 按钮。




按照指示填写具体的 Issue 内容

41 | https://doi.org/10.1007/978-3-658-33233-3_4



[Home](#)

 edited this page now · 1 revision

Welcome to the Github-Brochure-for--Technical-Writing wiki!

- + Add a custom footer

当前Page的内容

Edit

New Page

所有的Page目录

▼ Pages 1

Find a Page...

[Home](#)

- + Add a custom sidebar

Clone this wiki locally

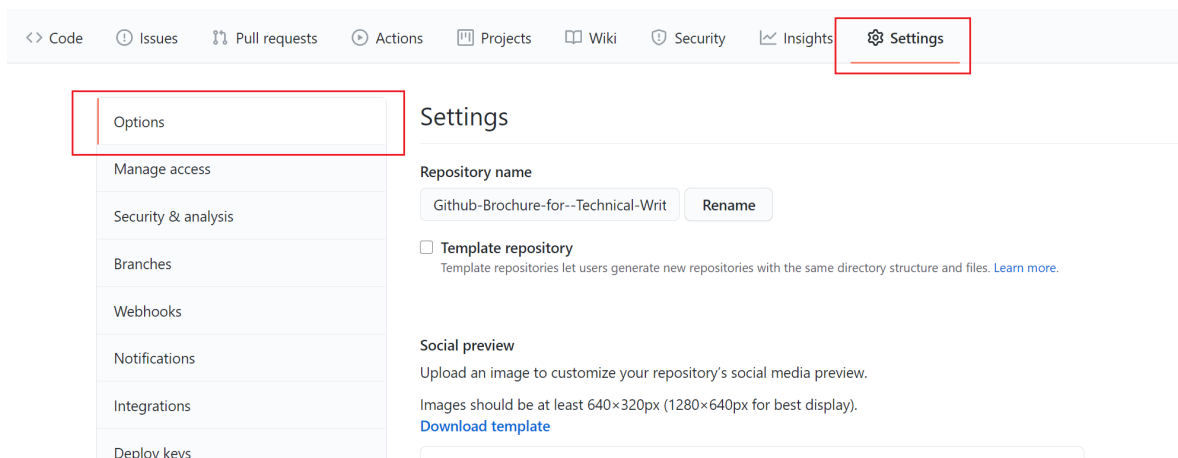


7.1 Github.io

经历上六章，我们已经可以将一份文档的源码部分合作开发完成。剩下的部分就是如何展示这一份文档。Markdown 或者其他格式的文档经过 Sphinx 等工具的编译可以生成 html 类型的文件。我们可以把这种类型的文件挂在 Github.io 上，供所有能上网的人访问。

7.2 发布

- 新建 Github 仓库这一步就不再详述。将编译好的文件上传到该仓库的 main 分支
- 对 Github 仓库进行设置打开设置选项卡



找到 Github Pages 选项

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is ready to be published at `https://1zhangyan.github.io/Github-Brochure-for--Technical-Writing/`.

Source

Your GitHub Pages site is currently being built from the `/docs` folder in the `main` branch. [Learn more.](#)

 Branch: main ▼

 / (root) ▼

Save

Theme Chooser

Select a theme to publish your site with a Jekyll theme. [Learn more.](#)

Choose a theme

Custom domain

Custom domains allow you to serve your site from a domain other than `1zhangyan.github.io`. [Learn more.](#)

Save

☒ Enforce HTTPS

— Required for your site because you are using the default domain (`1zhangyan.github.io`)

HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site. When HTTPS is enforced, your site will only be served over HTTPS. [Learn more.](#)

完成设置保存。

- 按照 Github Pages 选项提示的网址直接访问文档。