​

目录

收起

入门系列

一、初识界面

1、Git 和 GitHub

2、加入GitHub

二、仓库操作

1. 创建仓库

2. 提交更改

3. 查看更改

4. 名称修改

5. 删除仓库

三、Git的安装与使用

1. Git 安装

2. Git 基础

3. 总结

四、本地与远程仓库的交互

1. SSH 配置

2. 通过 Git 下载代码

3. 通过 Git 提交代码

五、Git 进阶

1. Git 基础

2. Git 进阶

六、GitHub 提交文件夹快速版

**入门系列**

**一、初识界面**

和很多小白一样，我也是一个看到大量文字和图片就头疼的人，下面这篇GitHub教程，结合我自己的理解，通俗易懂的方式讲述，目的就是讲明白。  
下面和我一起学习吧！

**1、Git 和 GitHub**

导读：我们先来了解一下 Git 和 Github 是什么，有什么功能

先说结论，Git 和 GitHub 是两个东西，Git 是一个软件/工具/系统，GitHub 是一个网站/平台，GitHub 这个网站使用了 Git 这个工具。

**1.1 Git**

Git是一个分布式[**版本控制系统**](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6%E7%B3%BB%E7%BB%9F&zhida_source=entity)。

版本控制系统（version control system）像个数据库，它会记录所有对项目文件的更改（比如一个文件，前天加了三段文字，昨天删了一句话，今天改了几个词，这三个版本历史都能保存下来）。 版本控制系统不仅可以应用于软件源代码的文本文件，而且可以对任何类型的文件进行版本控制。

使用版本控制系统可以**协同合作**（多人编辑文件或代码而不出错），**版本存储**（你改动的每一版本都保存下来，如果改错可以回到之前的版本，如果想加上删除的内容也可以返回去找，也可以对比现在和之前的版本，看改了什么），**文件备份**（服务器和本地都有完整的历史版本，如果服务器坏了，本地还有一份完整的历史记录）。

总之， Git 可以避免文件丢失，改错，多人合作不同步导致的后果。

**1.2 GitHub**

GitHub 是通过 Git 进行版本控制的**软件源**[**代码托管**](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=%E4%BB%A3%E7%A0%81%E6%89%98%E7%AE%A1&zhida_source=entity)**服务平台**，可以理解成放代码的地方，但往上放代码时用 Git 进行了版本控制。也就是 GitHub 使用了 Git 完成版本控制，下面来看看 GitHub 有哪些有用的功能！

**代码托管**：可以单纯地把它当成一个网盘放你的代码，同时使用 Git 功能记录你的代码历史。当然除代码外还能放其他文件。  
**学习优秀的开源项目**：学习别人优秀的源码，写代码之前看别人是怎么写的（比如写作业的时候参考参考），看论文也可以上 GitHub 找源代码，还能找一些开源的软件，插件用。  
**当资料库**：可以查资料，GitHub 上有总结好的面试宝典，入门指南，技术分析，论文合集，课程资料等。这些博客、公众号上有的内容 GitHub 上也有，还可能更全。  
**多人协作**：多个人要一起写个程序，一起写本书，一起翻译一篇文章等，用 GitHub 可以管理项目保证你们的文件同步，写好后提交合并成一个完整的项目。  
**搭建博客**：基于 GitHub Pages 搭建属于你的博客，你可以随心所欲的定制自己的样式，这是一个属于你的空间。  
**社交**：就像微博、知乎一样，在这个网站你可以关注（following）别人，也可以有自己的粉丝（followers），看到好的开源项目可以给他点赞（star），你有啥想法还可以给这个项目改进改进（fork）。  
**个人简历**：如果你的 GitHub 上有不错的项目，或者你改进过别人的项目，这些都能反映到你的账号上。GitHub 一定程度可以反应到你的能力，如果你的项目点赞多（star）,还有很多粉丝关注（followers）,你就像个大 V 一样，这就是你的另一份简历。  
**写作**：Gitbook 可以写电子书。  
GitHub 能做的还远不止这些，等待你的探索发现。

另外，GitHub 不用FQ，有网就能打开，可能比国内网站反应慢些，这是正常现象；虽然网站都是英语，但模块不是很多，跟着下面的教程就能学会，用几次就会了；网站有很多中国用户，所以你可以搜到很多中文资源，不用担心语言问题

[GitHub国内0延迟打开方法【小白必看】​gjdy.watk.xyz/git.html](https://link.zhihu.com/?target=https%3A//gjdy.watk.xyz/git.html)

总结：无论你是不是程序员，你都可以用 GitHub。你可以把它当个网盘存包括代码的文件，可以和团队一起做个项目（文件啥的就不用保存一堆迭代版本啦），可以找资源（里面有很多宝藏资源），可以建个博客，也可以像在微博、知乎、博客一样，分享你的东西。

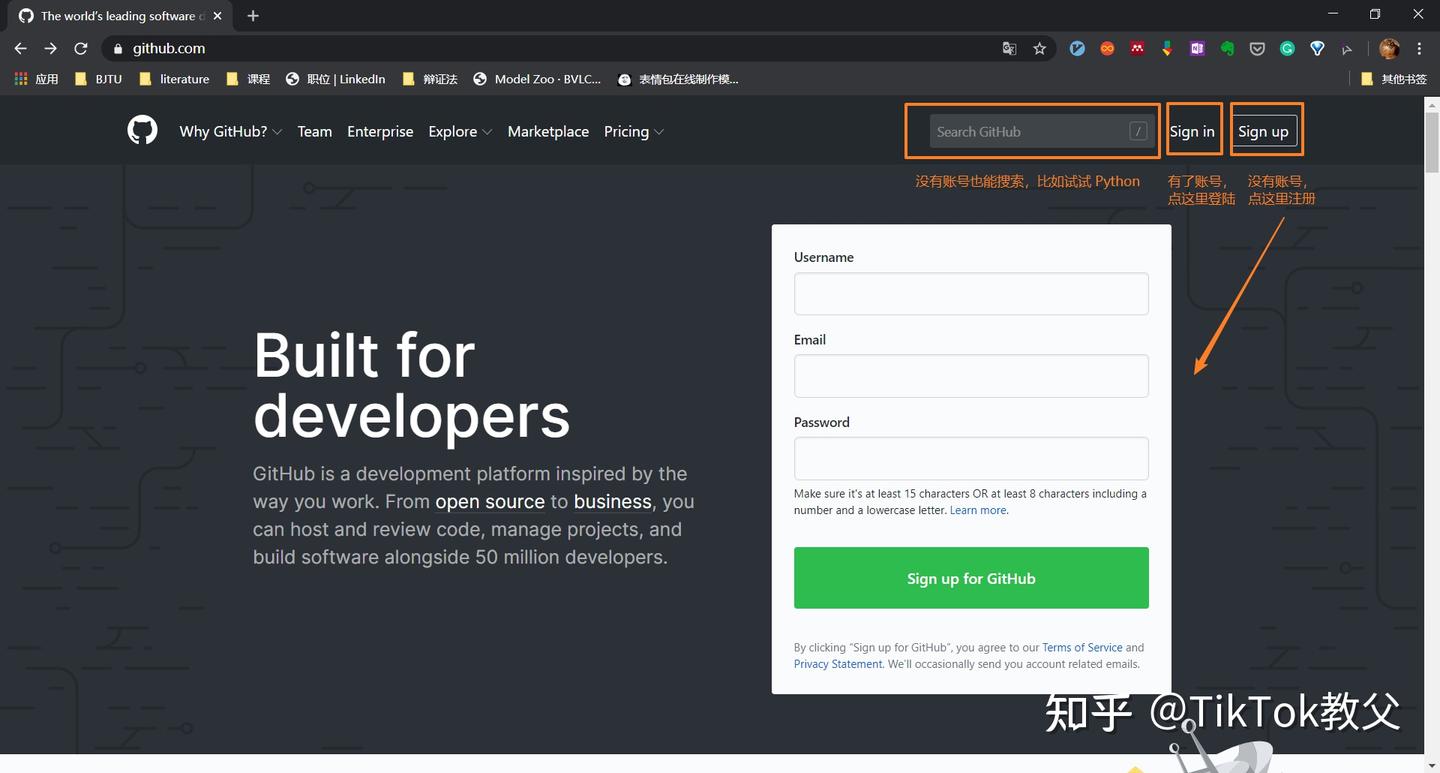
**2、加入GitHub**

导读：注册个账号，看看网站长啥样吧。

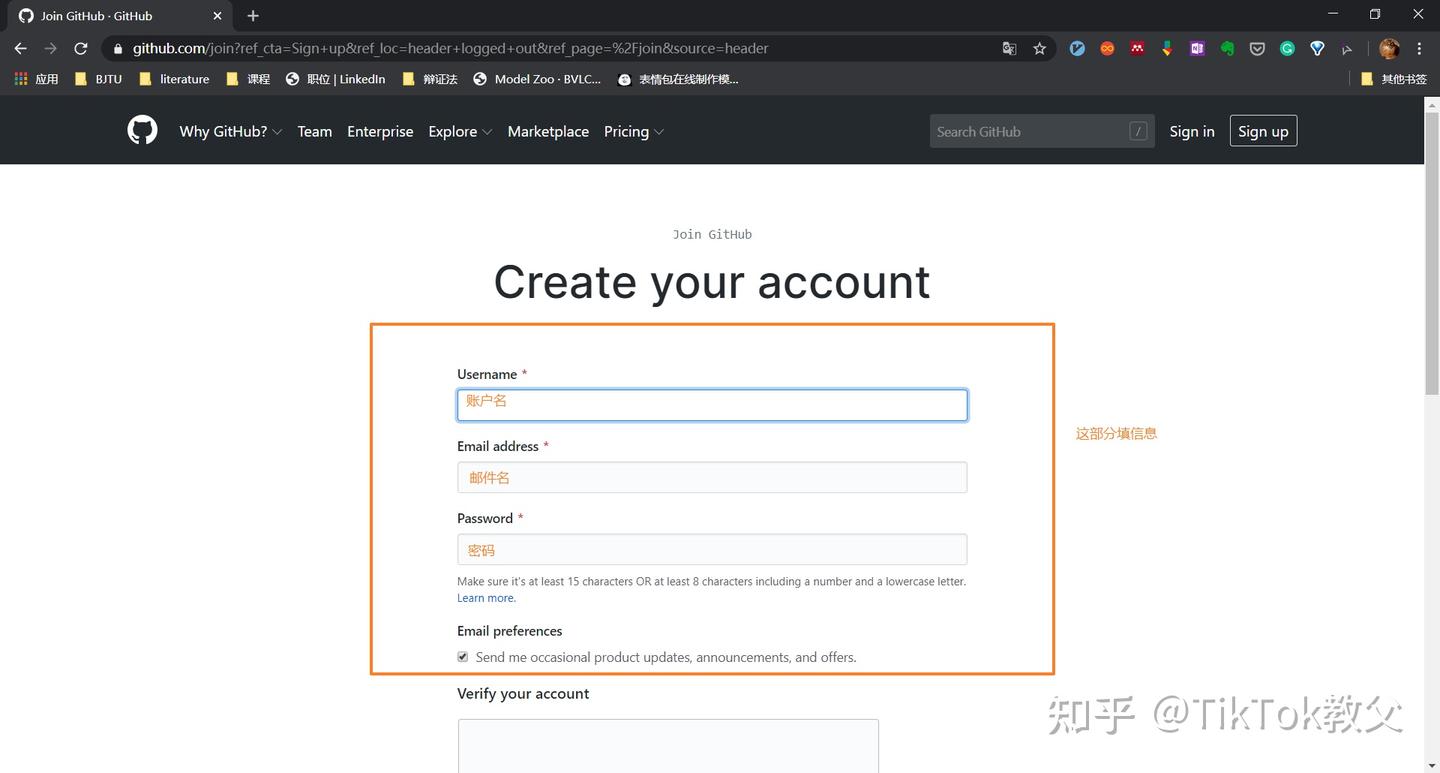
**2.1注册**

打开这个网址[https://github.com](https://link.zhihu.com/?target=https%3A//github.com/%25EF%25BC%258C)如果没有登陆的话，就会看到下面的主界面。

首先，你没有账号也能搜索（游客模式），**Sign in**用来登陆，**Sign up**用来注册账号。



现在还没账号，可以直接在上面的页面注册，也可以点击**Sign up**到下面这个页面（其实注册内容都一样，注册网址：[https://github.com/join?source=login）。](https://link.zhihu.com/?target=https%3A//github.com/join%3Fsource%3Dlogin%25EF%25BC%2589%25E3%2580%2582)



输入上面的信息后，点击页面下面的 **Create account** 即可。

这几年的注册界面有变化，如果创建账号后，让你选择账户类型（**pick the plan** 或者 **choose your plan**），你就选择免费（**free**），公共（**public**）的那个。

如果需要验证邮箱（**verify your email address**），就打开刚刚写的邮箱，打开收到的邮件，点击 **verify email address** 即可。

如果注册有困难，可以看看本年的图文教程。

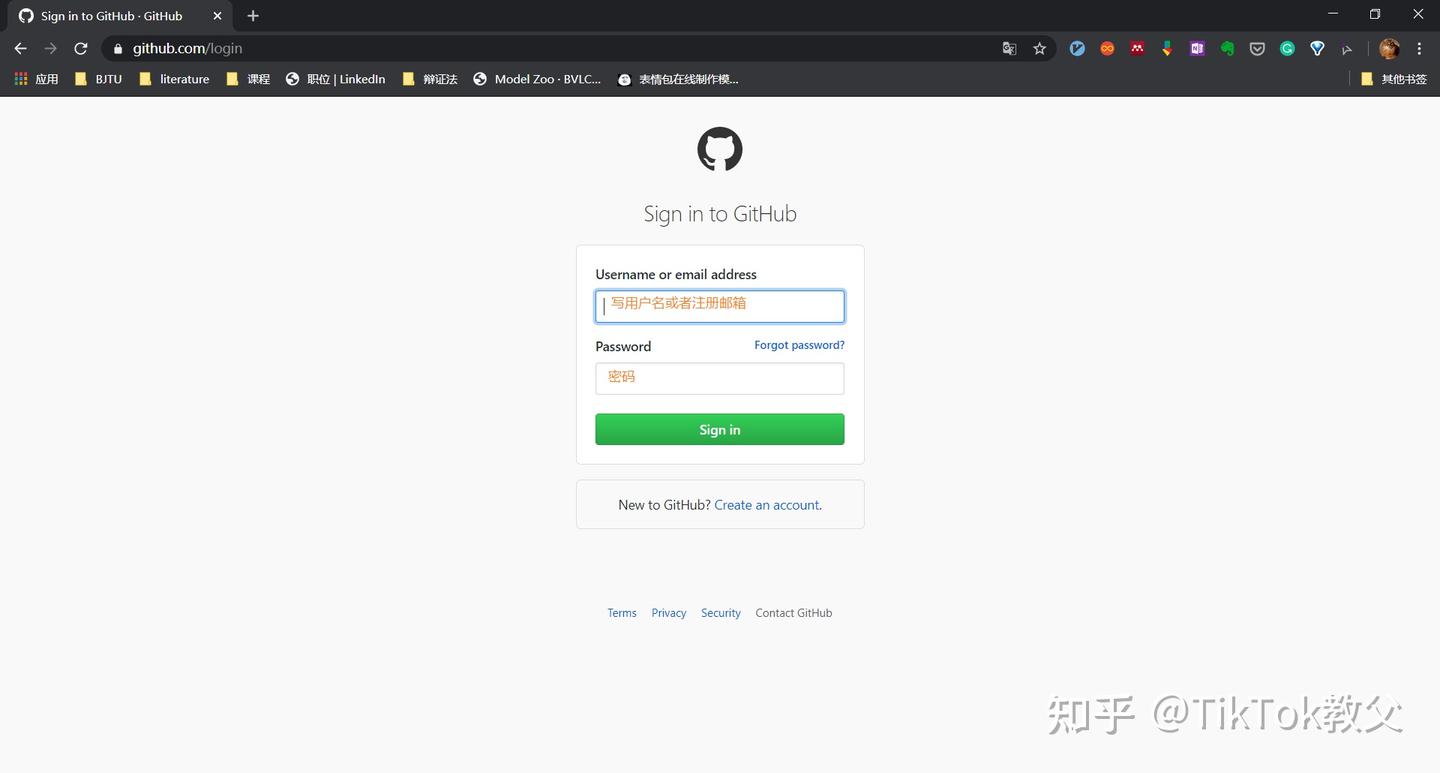
注册成功你会进入 GitHub 主页面，我们第 2.3 讲。

**2.2登录**

当你有账号后，你就可以登陆啦。

登陆是点击上面任意页面的 **Sign in** 按钮，进入下面这个界面（登录网址：[https://github.com/login）。](https://link.zhihu.com/?target=https%3A//github.com/login%25EF%25BC%2589%25E3%2580%2582)

输入用户名或邮箱，密码，点击 **Sign in** 或者回车即可。



登录成功你也会进入 GitHub 主页面。

**2.3 界面**

**2.3.1 主界面**

登录状态下，网站主界面（网址：[https://github.com/）如下：](https://link.zhihu.com/?target=https%3A//github.com/%25EF%25BC%2589%25E5%25A6%2582%25E4%25B8%258B%25EF%25BC%259A)

**首先看最上面黑色工具条。**

最左边是一个 GitHub 的 logo，点它就返回现在这个主界面。

最右边是和你个人有关的东西（个人，新建，消息），这个和用过的任何软件一类比，就知道功能了。

左边搜索框，和搜索引擎一样用来搜索（搜源码，搜资料）。

中间的四个按钮：

* **Pull requests**：你想参与别人的项目，或想改进别人的项目，改进后需要提交申请，用 pull requests。
* [**Issues**](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=Issues&zhida_source=entity)：你发现别人项目的 bug，或者有什么问题，怎么和作者交流呢？提一个 Issue ；别人也可以给你的项目提 Issue 反馈问题。Issue 追踪各种想法，增强功能，任务，bug，有点儿像评论功能。
* **Marketplace**：应用商店，里面有免费/收费的开发工具。
* **Explore**：你可以理解成软件里的发现页，点进去推荐一些和你相关的话题、项目、新闻等。

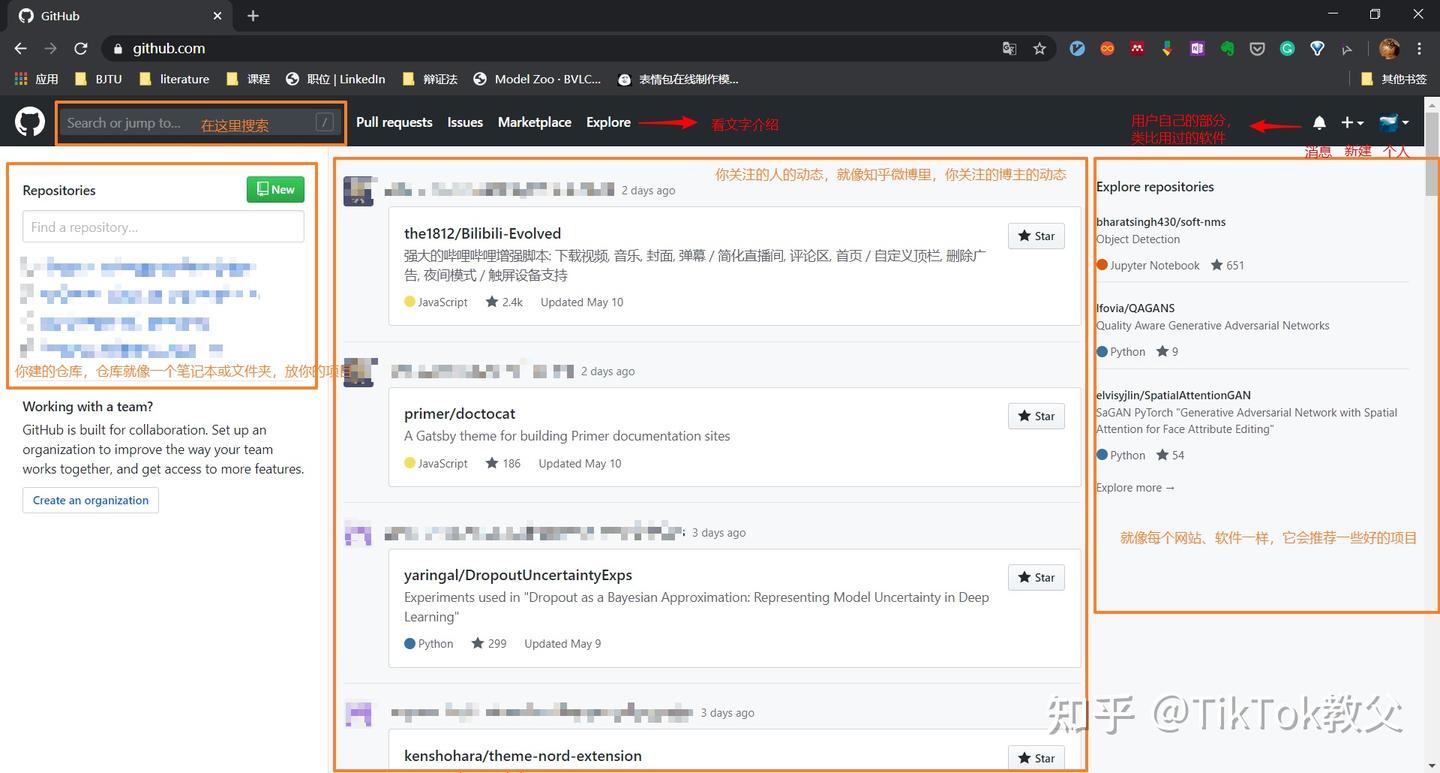
**再来看看下面的一大块**。

左边是你的项目/[Repository](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=Repository&zhida_source=entity) 。

* **Repository**：翻译为仓库，也是你的项目。你可以理解成一个大的文件夹，或者笔记本。一个项目对应一个 Repository。

中间是你关注的人的动态，比如他给别人点赞，他又建了一个项目。

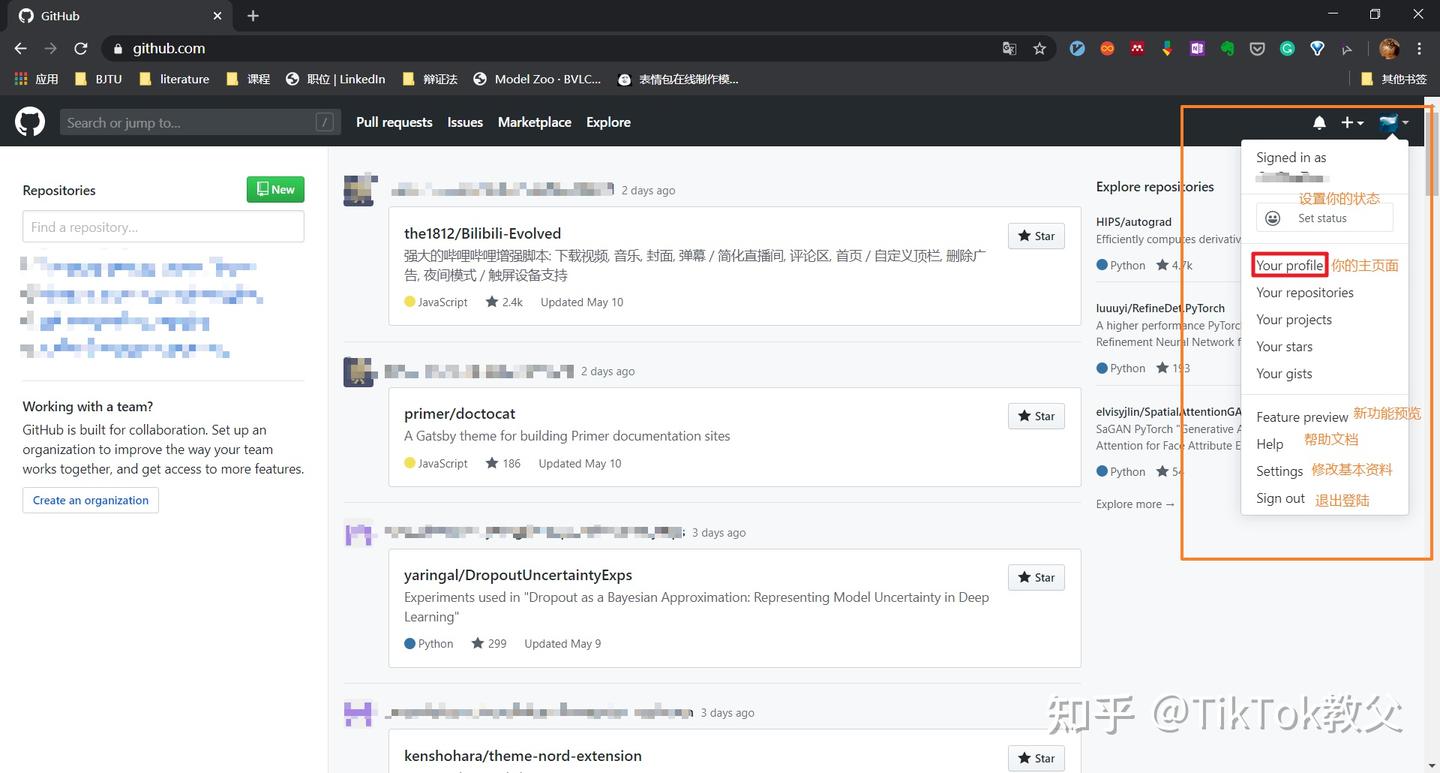
右边类似于推荐页，网站根据你的喜好推荐一些相关项目。



**2.3.2 个人界面**

刚刚说了右上角那个头像就是**个人** 。

点一下头像，就会弹出一个小页面，点击 **Settings** 可以更改你的资料，Your XXX 都是你创建的一些东西（这里把点赞 star 也理解成你创建的），点击 **Your profile** 就可以进入你的主页。



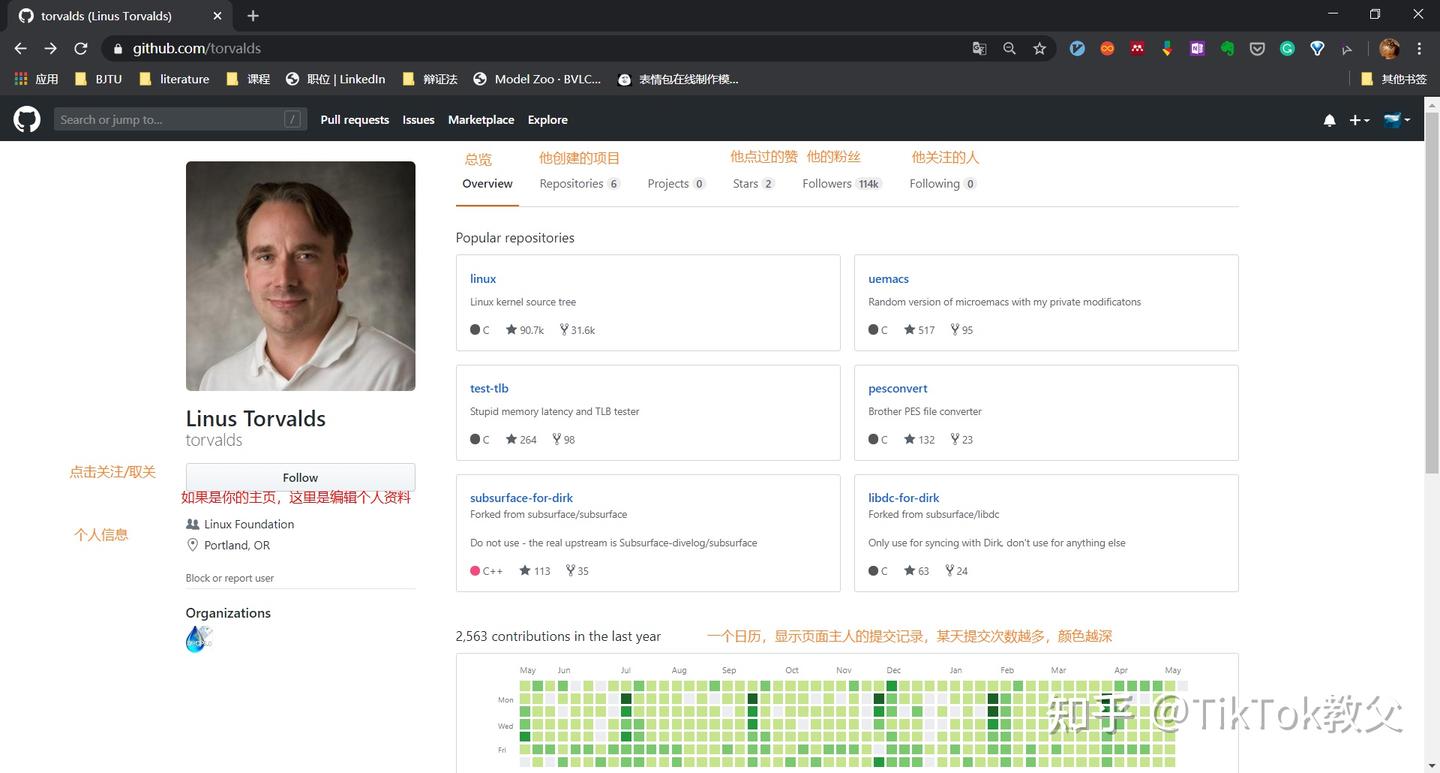
由于个人主页面的分布都一样，我的界面又没啥东西，我就用 Linux 之父的界面做演示了。

可以看出，在个人界面中，可以看到当前这个人的信息，类似于微博知乎的个人界面。我们自己的个人界面也长这样，不过刚注册，下面的记录一片空白。

看他的粉丝数，就知道这是个大v，这里最能体现 GitHub 的社交属性。

这里又涉及到一些专用词。

* **Projects**：它可不是项目，上面说了 Repository 是项目。它可以翻译为项目板，是 project-boards 的简写， 通过项目板可以灵活地创建适合需求的自定义工作流程，说白了是管理项目流程的，一般不常用。
* **Star**：就是点赞功能，这里用作动词，它叫 star 是因为点赞图标就是星星。不过这个点赞比较像知乎里的点赞功能，它会记录在你的动态里。由于 GitHub 没有收藏功能，所以 star 可以用来收藏。
* **Follow**：就是关注的意思，点击 Followers 看看谁关注了他，点击 Following 看看他关注了谁（大神关注的一般也是大神）



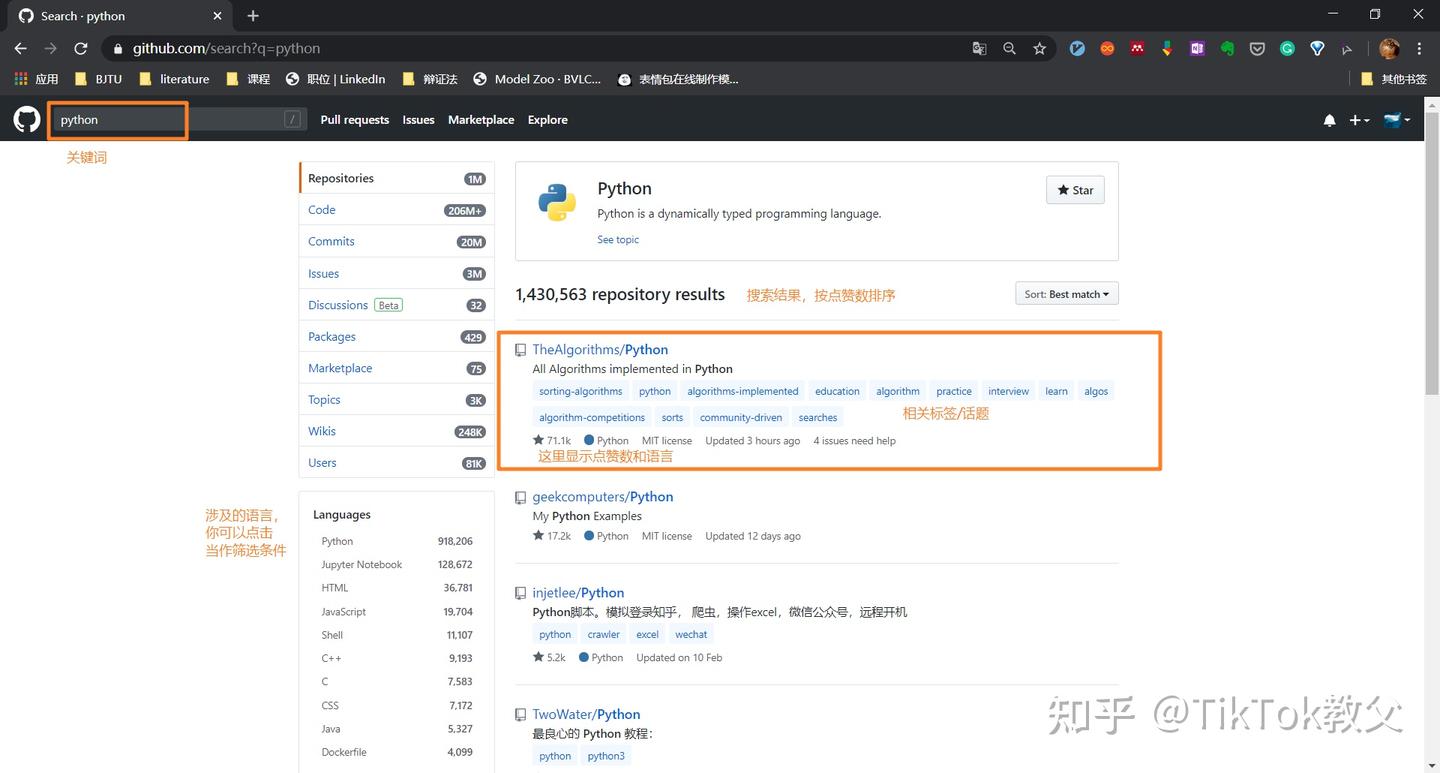
**2.3.3 项目界面**

下面我们看看项目界面。

我们使用 GitHub ，无论存放东西，还是查资料，主要都是看自己或别人的项目/仓库，所以这个界面一定要熟悉。

你可以通过搜索项目、点击别人的界面、推荐页面打开一个项目。

我们以 Python 为例介绍界面，你可以在搜索框输入 Python 搜索，选择第一个项目，看点赞数就知道它是最欢迎的一个。



点进去就是项目/仓库界面啦，我们认识一下主要功能

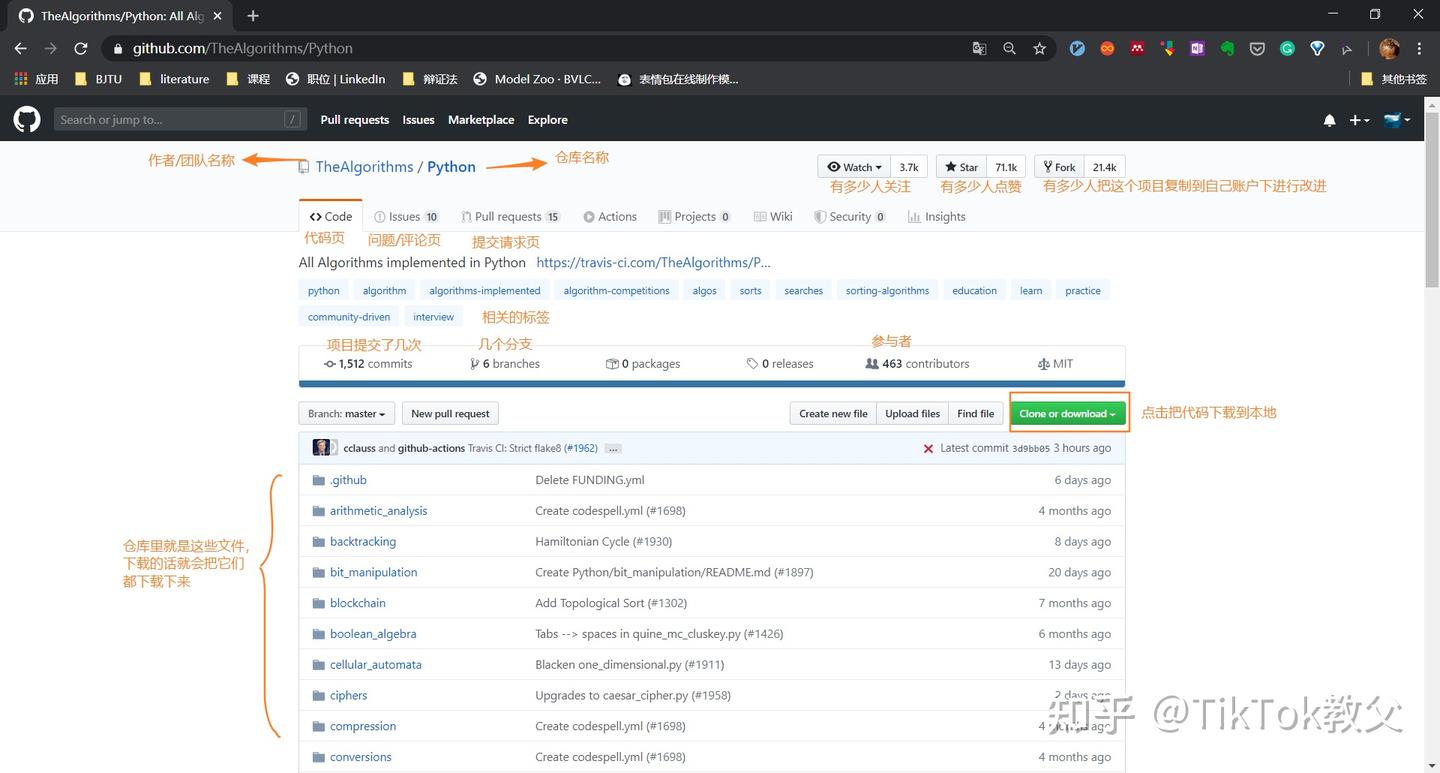
首先最上面一条我们可以看到这个仓库的信息，像关注点赞都是按钮，点击可以看具体的人。

这里涉及到两个新词语。

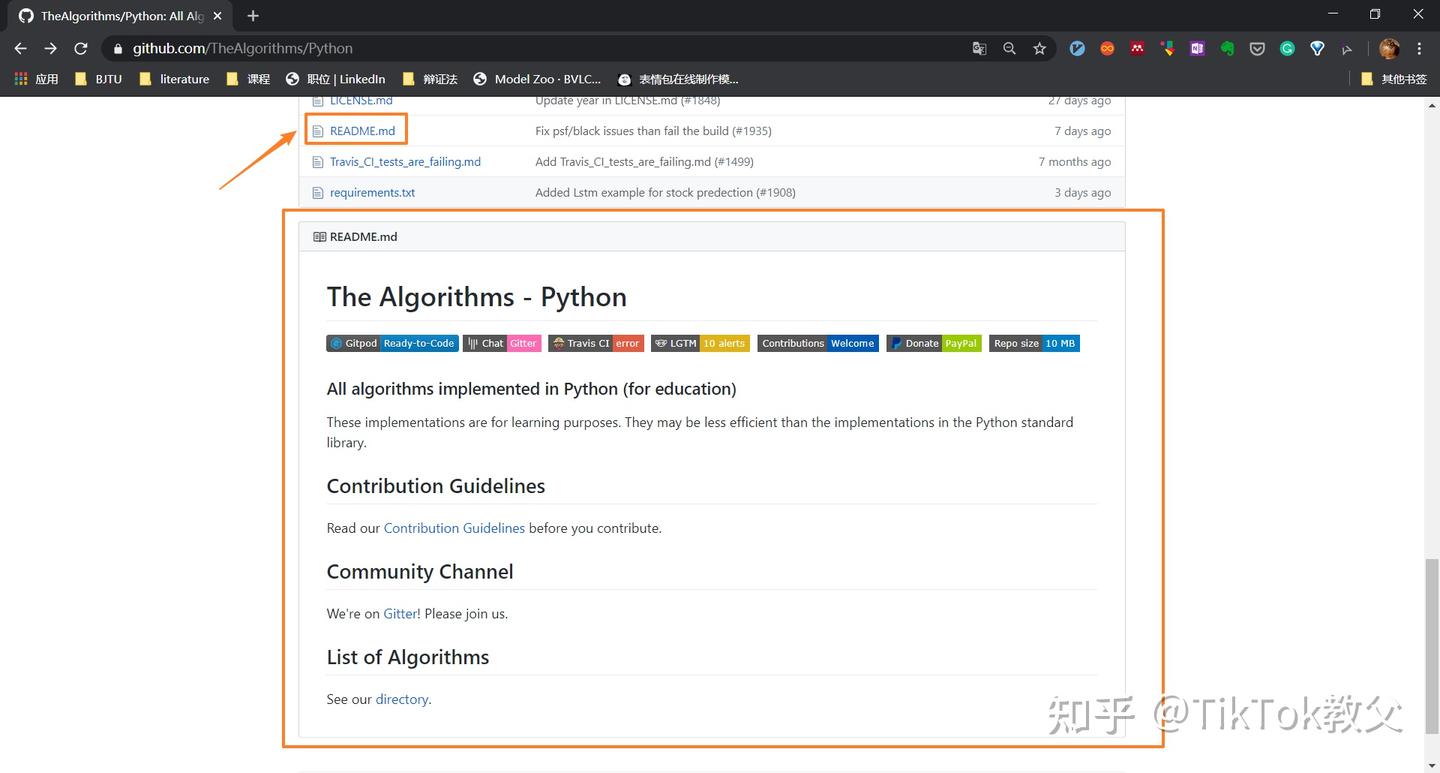
* **Watch**：关注观察，也就是你既可以关注（follow）一个人，也可以关注（watch）一个项目，你关注内容的动态都会显示在主页面。
* [**Fork**](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=Fork&zhida_source=entity)：直译是刀叉，它是指将 GitHub 的某个特定仓库（所有文件）原封不动地复制到自己的账户下。比如你想改进这个项目，加点儿自己的东西，就可以复制一下整个仓库再修改，但是不影响原作者的仓库，你点击 Fork 就能复制。

下面一条是标签页，比如默认的一个标签页 **Code** 就是展示代码的页面；如果你想看别人提的问题就点击 **Issues** 页，也许你遇到的问题别人提过并且解决了；有的人想参与这个项目，他改好后就向作者发起了 **Pull Requests**，希望作者接受他的改进，点进去可以看谁提交过什么样的改进，作者是否采纳。

下面这个主要部分就是仓库里的东西了，你可以看到就是一个个文件夹或文件，里面可能是代码文件，也可能是其他文档，图片什么的。点击可以看，你也可以点击 **Clone or downloads** 下载到本地，具体学习。



滑到最下面，可以看到一个叫 [README.md](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=README.md&zhida_source=entity) 的一段文字，仔细看，它就是仓库里的一个文件，只不过展示出来了。它就像产品说明书，或者是一个介绍页，告诉你这个仓库的有关信息，让你对仓库有了简单的了解。



以后你要建个仓库，为了方便别人了解，也要写这样的文件。

**2.4 名称**

这里再总结一下 GitHub 的基本名称，都是上面提到的，光看字面意思不好理解，总结一下方便查询。

|  |  |
| --- | --- |
| 名称 | 功能 |
| Pull Request | 你想参与别人的项目，或想改进别人的项目，改进后需要提交申请，用 pull requests。 |
| Repository | 翻译为仓库，也是你的项目。你可以理解成一个大的文件夹，或者笔记本。一个项目对应一个 Repository。 |
| Issue | Issue 追踪各种想法，增强功能，任务，bug，有点儿像评论留言功能。 |
| Star | 就是给项目点赞 |
| Fork | 将 GitHub的某个特定仓库（所有文件）原封不动地复制到自己的账户下（但不会与原仓库相同）。以此实现分散式版本管理 |
| Follow | 关注人，关注后主页有他的动态 |
| Watch | 关注项目，关注后主页有项目的动态 |
| Gist | 上面没有提到，但是主界面点击头像可以看到 Your Gist。如果你没有一个项目，只是单纯地想分享一些代码片段，就可以写 Gist。 |
| README.md | 每当创建项目、初始化时，都会帮你自动生成 README.md 文件并显示在仓库首页。一般都是使用 Markdown 语法（准确来说应该是GitHub Flavored Markdown(GFM）语法)来描述项目的概要、使用流程、许可协议等。er |

**二、仓库操作**

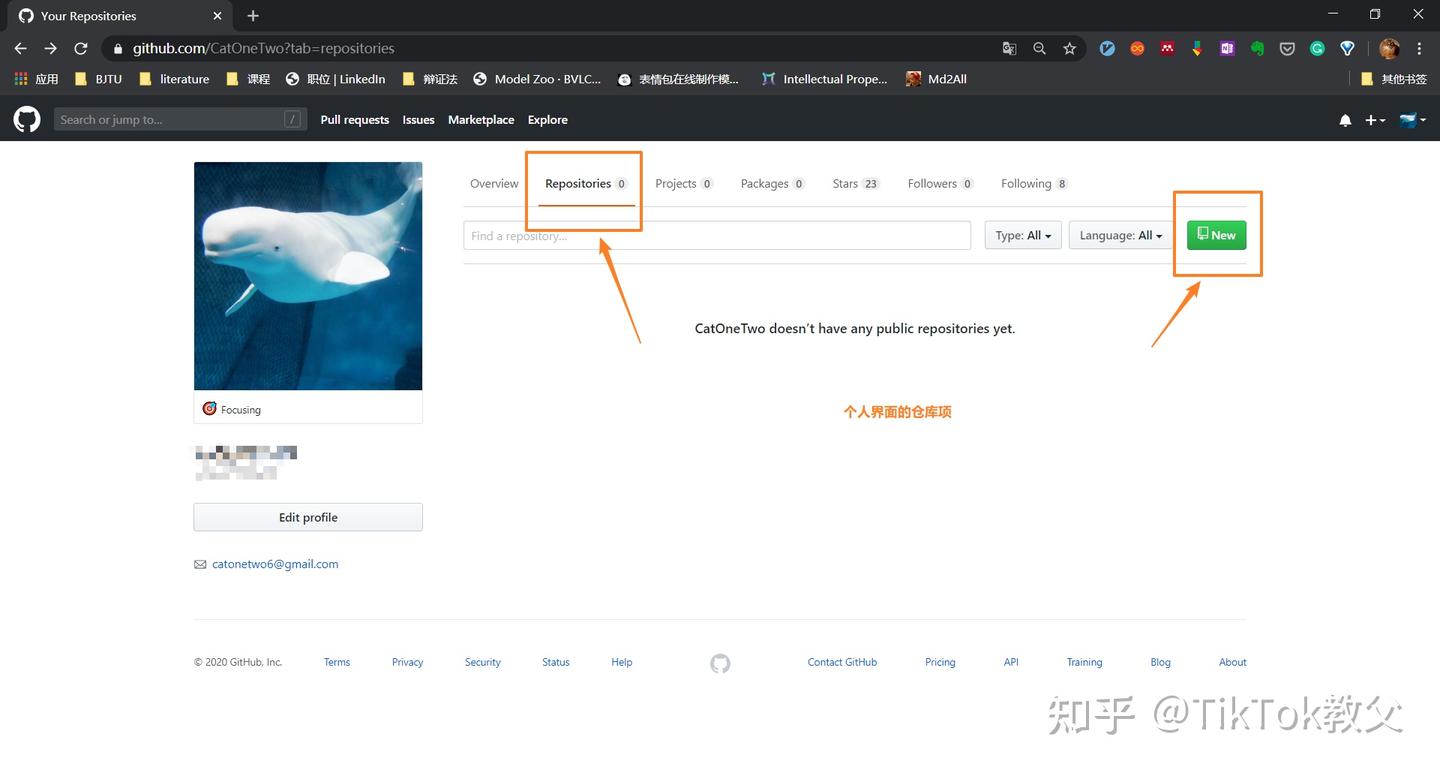
上一次我们认识了 GitHub 界面，尤其是项目/仓库界面，这一次我们学习仓库的常规操作，比如创建删除重命名，以及特殊的操作提交更改以及查看更改。  
让我们开始吧！

**1. 创建仓库**

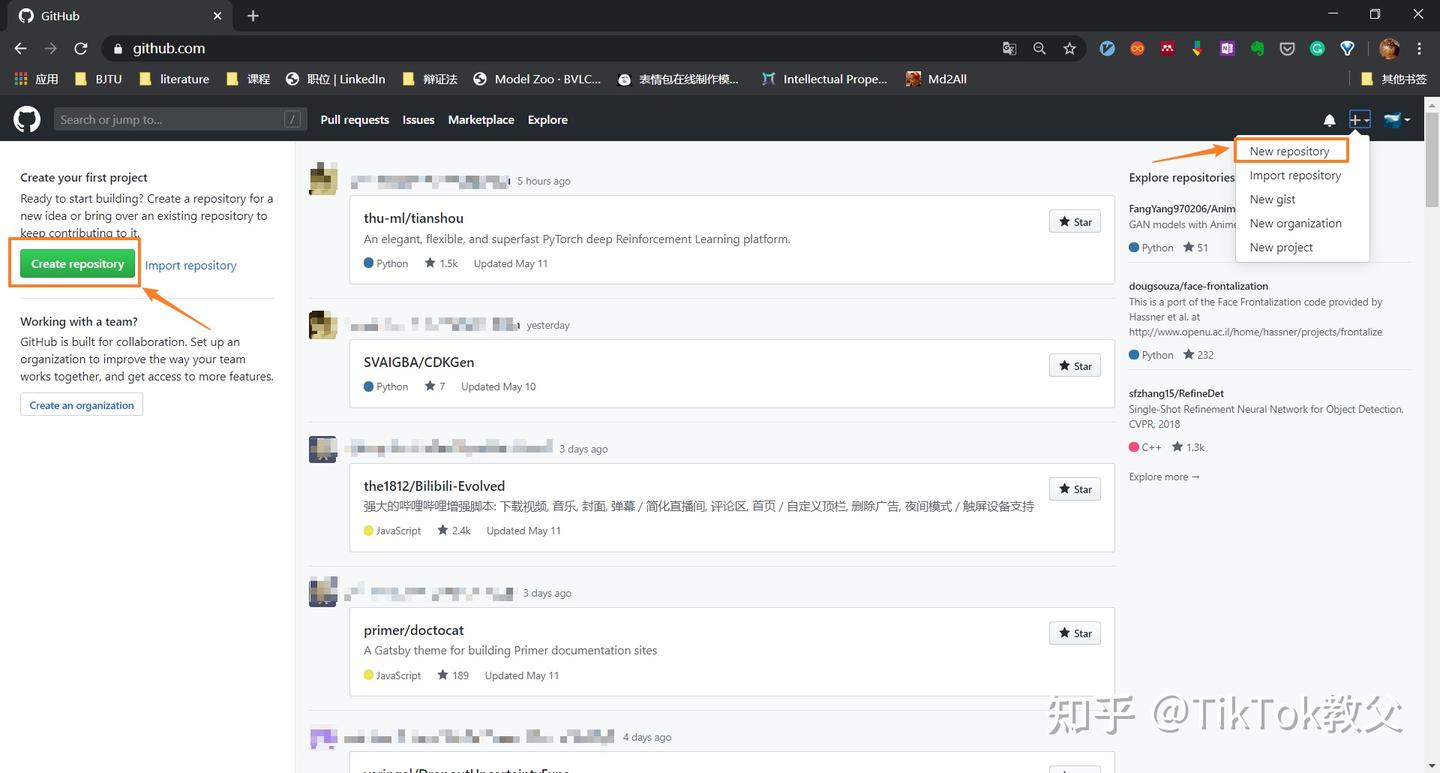
导读：当我们往 GitHub 上存东西，我们实际上是往仓库里存放，仓库就类似于我们新建的一个文件夹，它的创建并不复杂

首先我们看看哪些地方可以新建仓库。

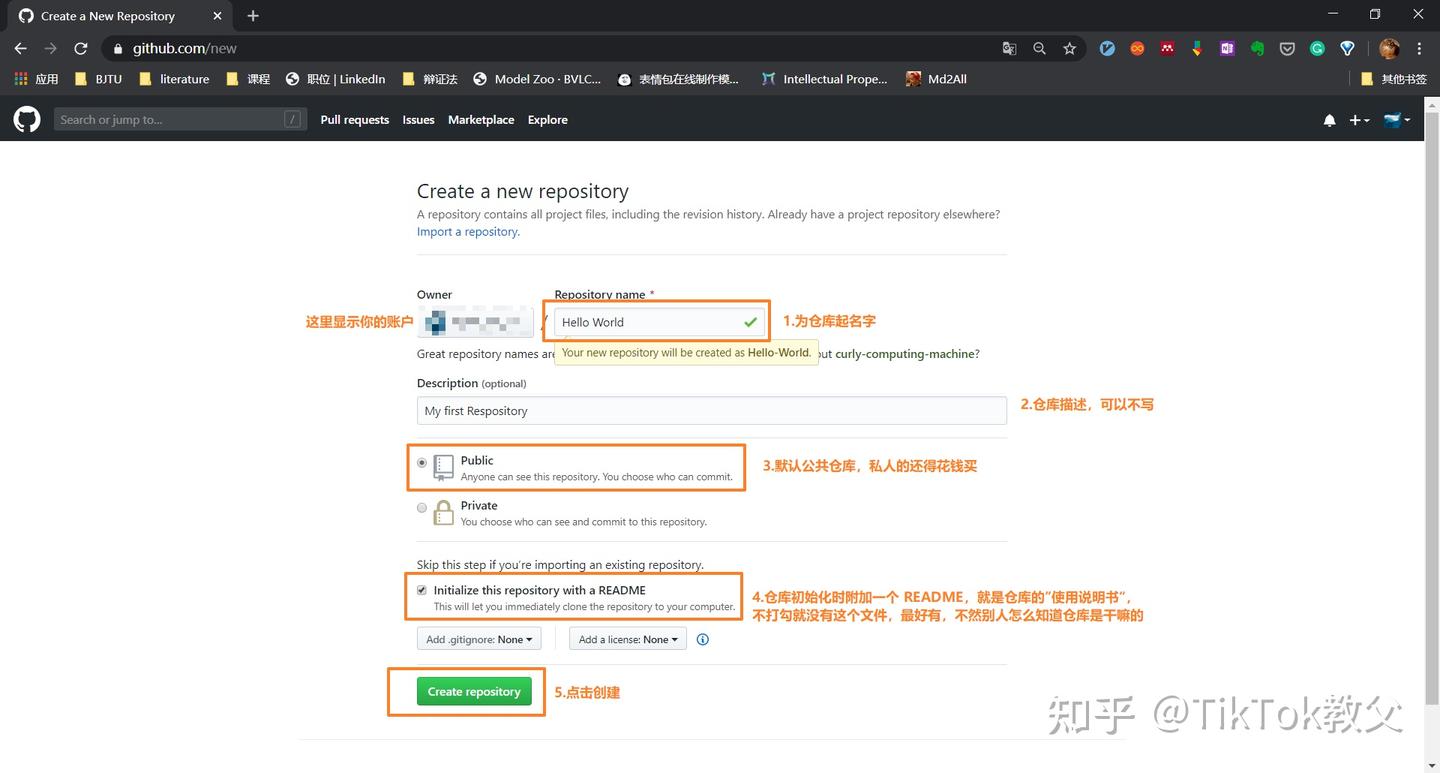
个人界面的仓库页面可以新建.



你登陆后，在 GitHub 主界面也可以新建。

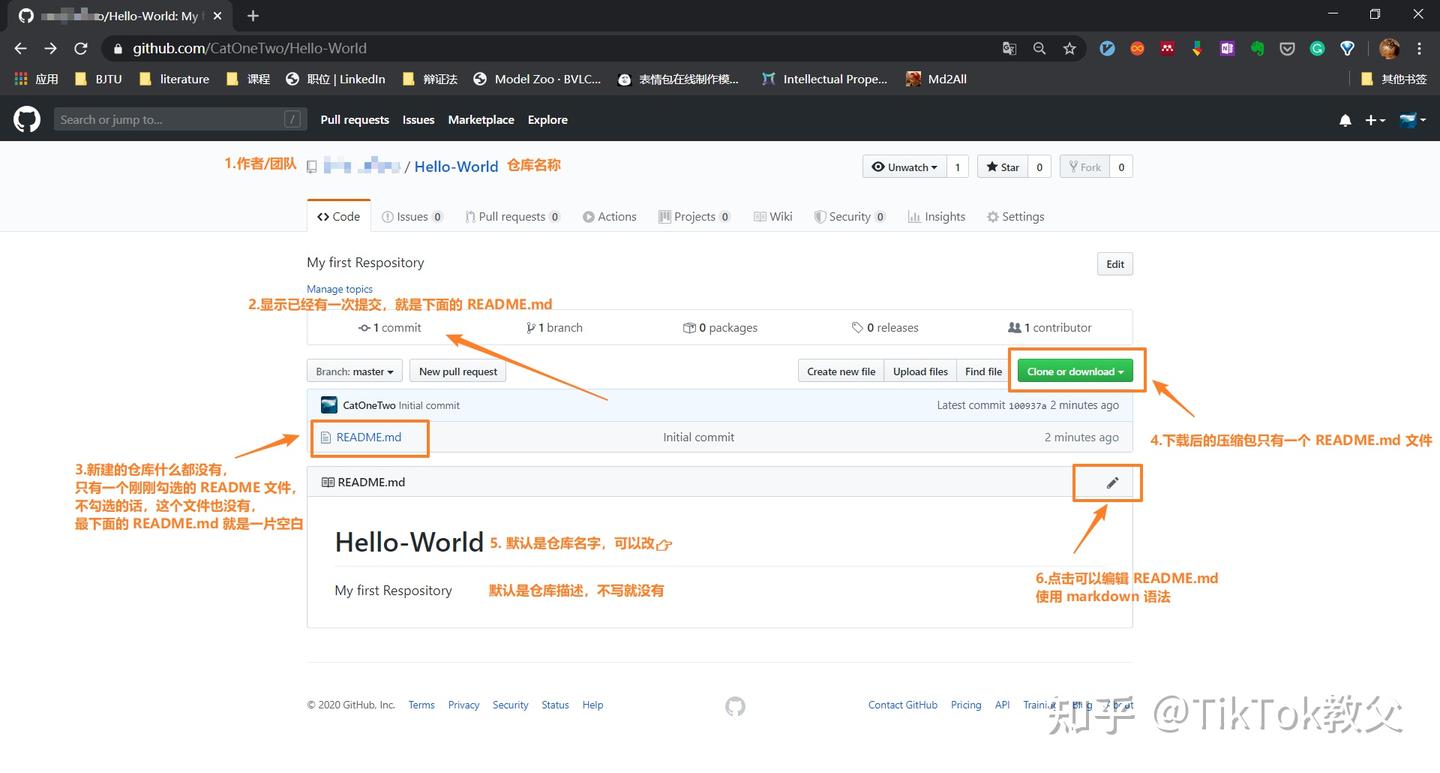


我们任选一个入口新建仓库，界面如下，步骤也都在上面了，是不是很简单。

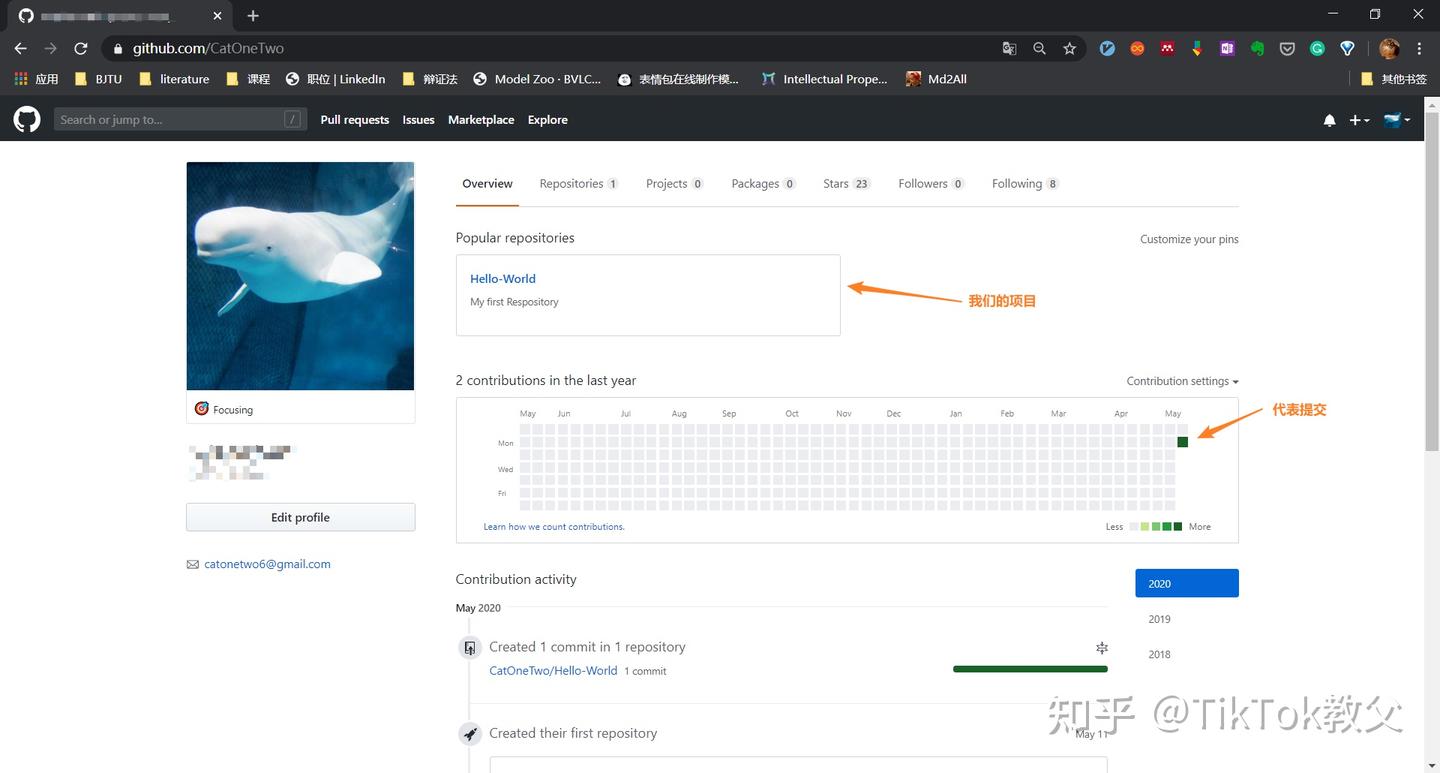


创建好后，我们看看仓库/项目界面，看过上一篇文章应该比较熟悉了。

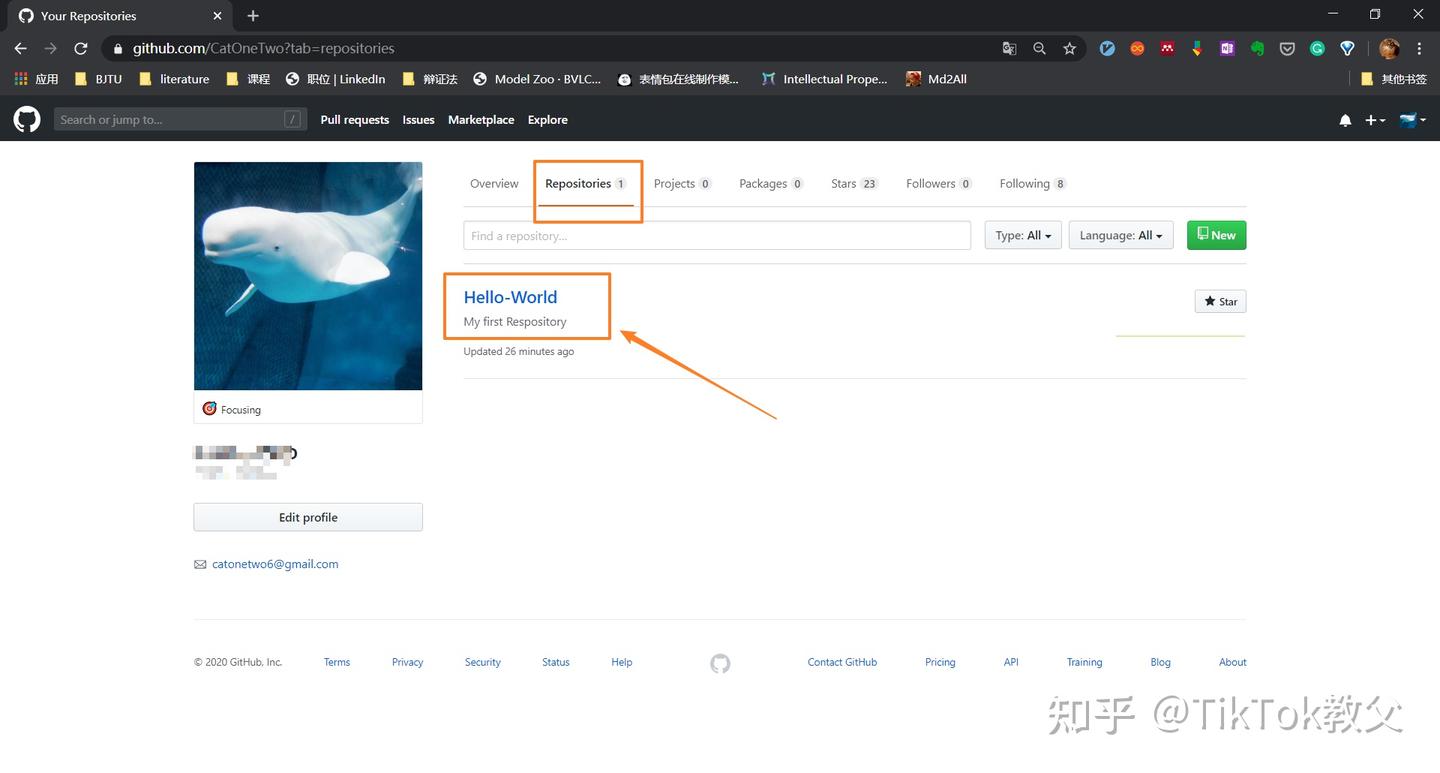
仓库名为 Hello World，包含 1 个commit，也就是我们通过勾选 Initialize this repository with a README，创建了一个初始化提交文件 README.md，其中文件后缀为 .md，表示文件为 Markdown 格式；包含 1 个branch，为master分支，即主分支；包含 1 个contributor，为贡献者，也就是我们自己。



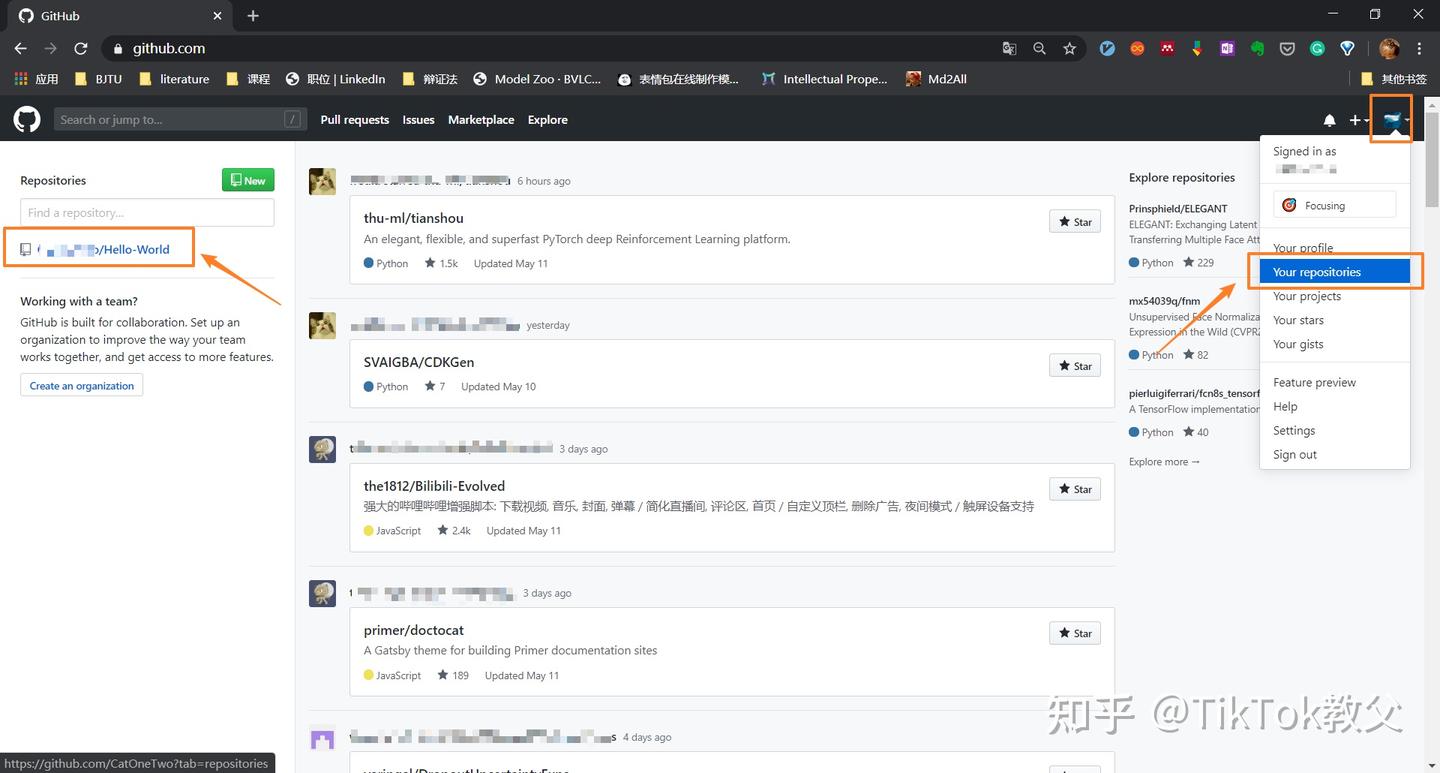
当我们打开个人界面，就可以在 Overview 看到我们的仓库啦。



在个人界面的 Responsitory 里也可以看到。



当然，在主界面的也可以看到。

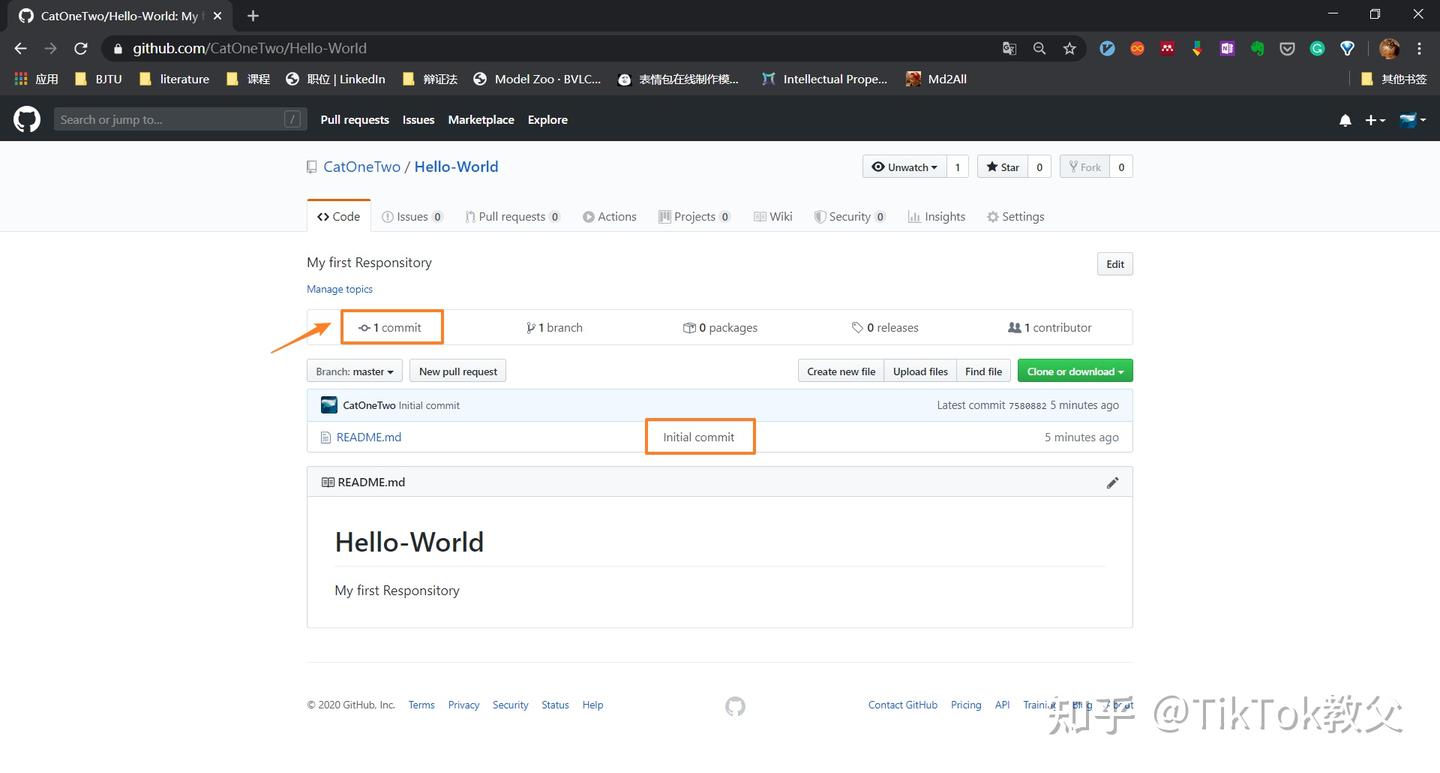


总结：经过上面的步骤，我们已经学会了创建仓库，可以感受到，我们像是创建了一个“文件夹”，里面只有说明文档（README），之后，你可以通过往里面提交文件完善它

**2. 提交更改**

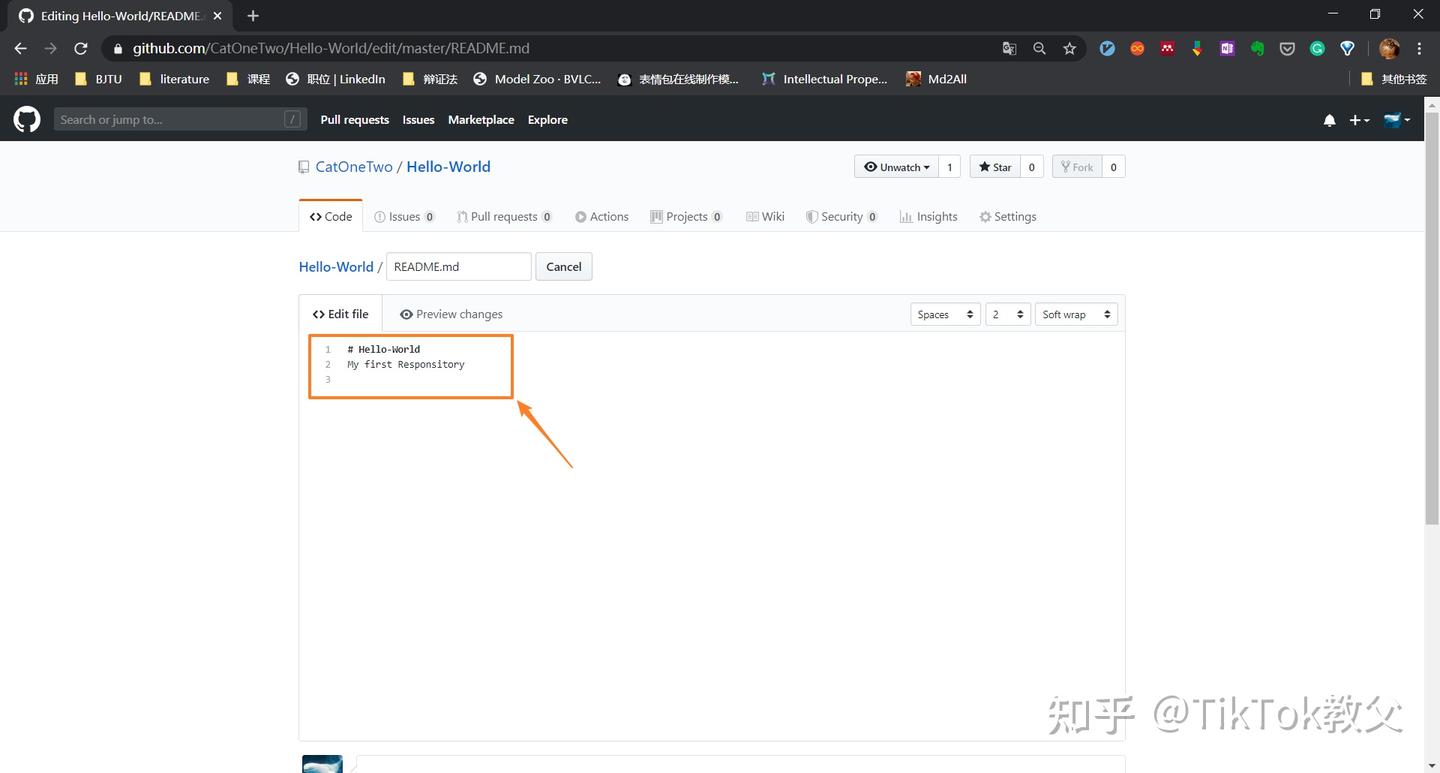
导读：这一部分我们试着改 README 文件，并提交这次修改

提交**Commit**就像是项目中所有文件在特定时间点的快照，创建新仓库时，我们用 README 自述文件对仓库进行了初始化，这是仓库的第一次 Commit。

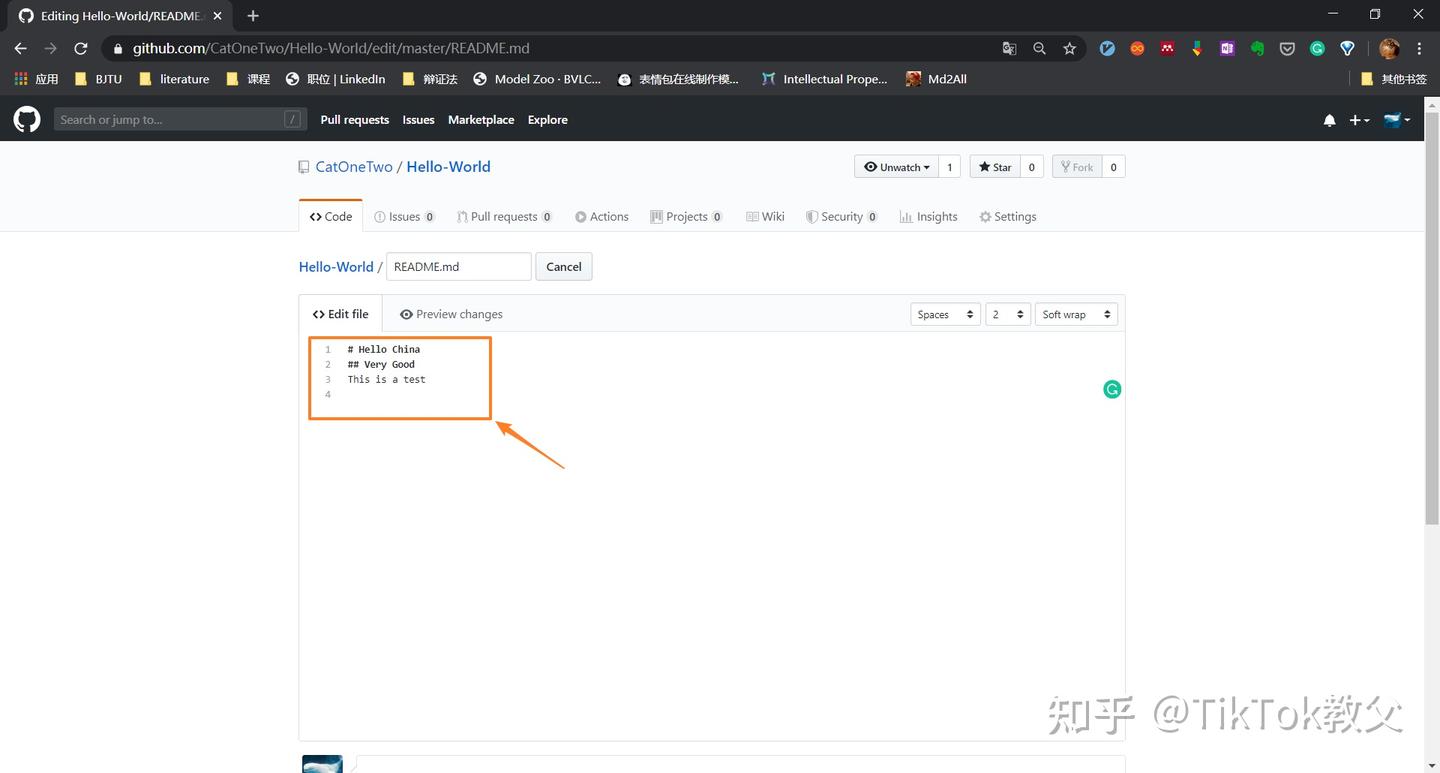


下面我们改一下 README 文件，并提交这次更改吧。

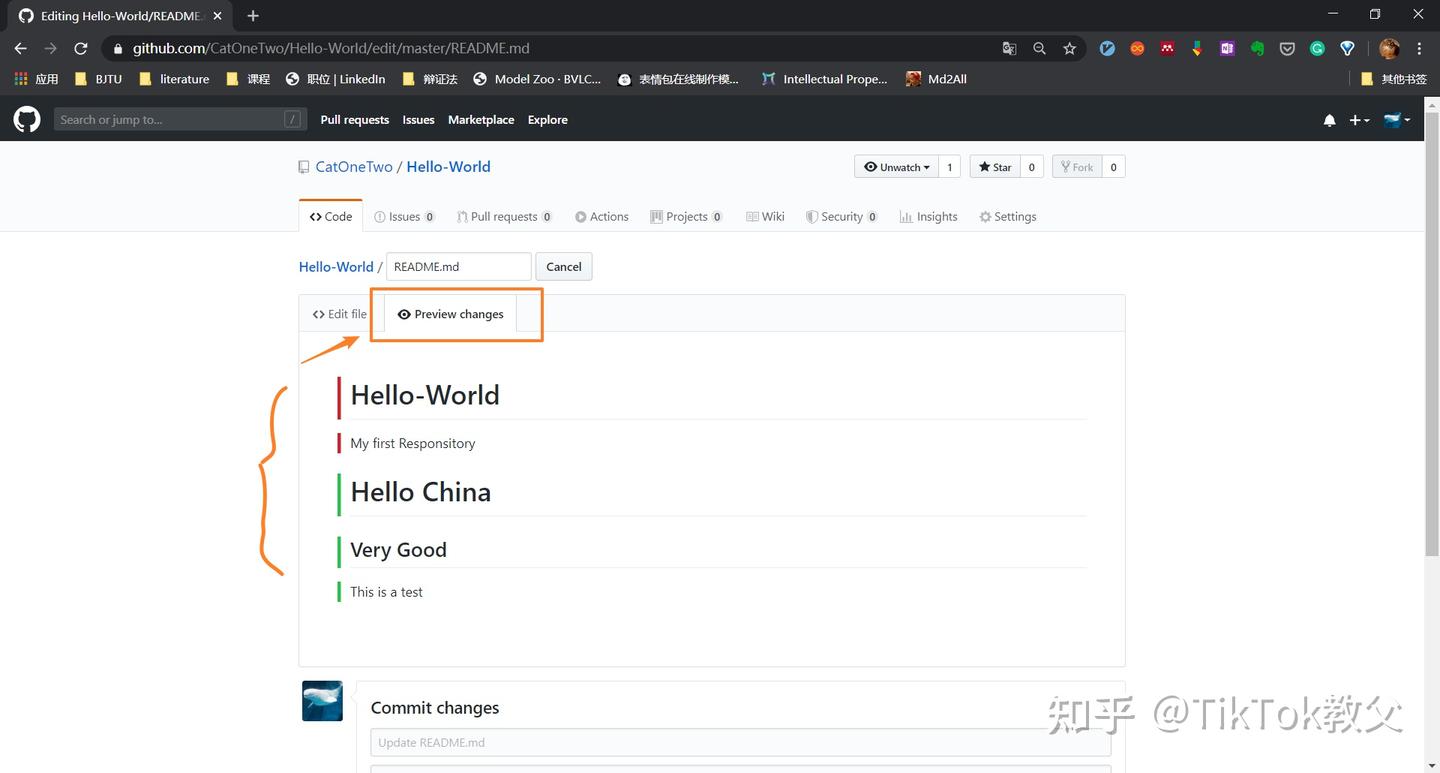
点击README.md 右边的那支笔，进入编辑模式，我们会看到 markdown 源文件。



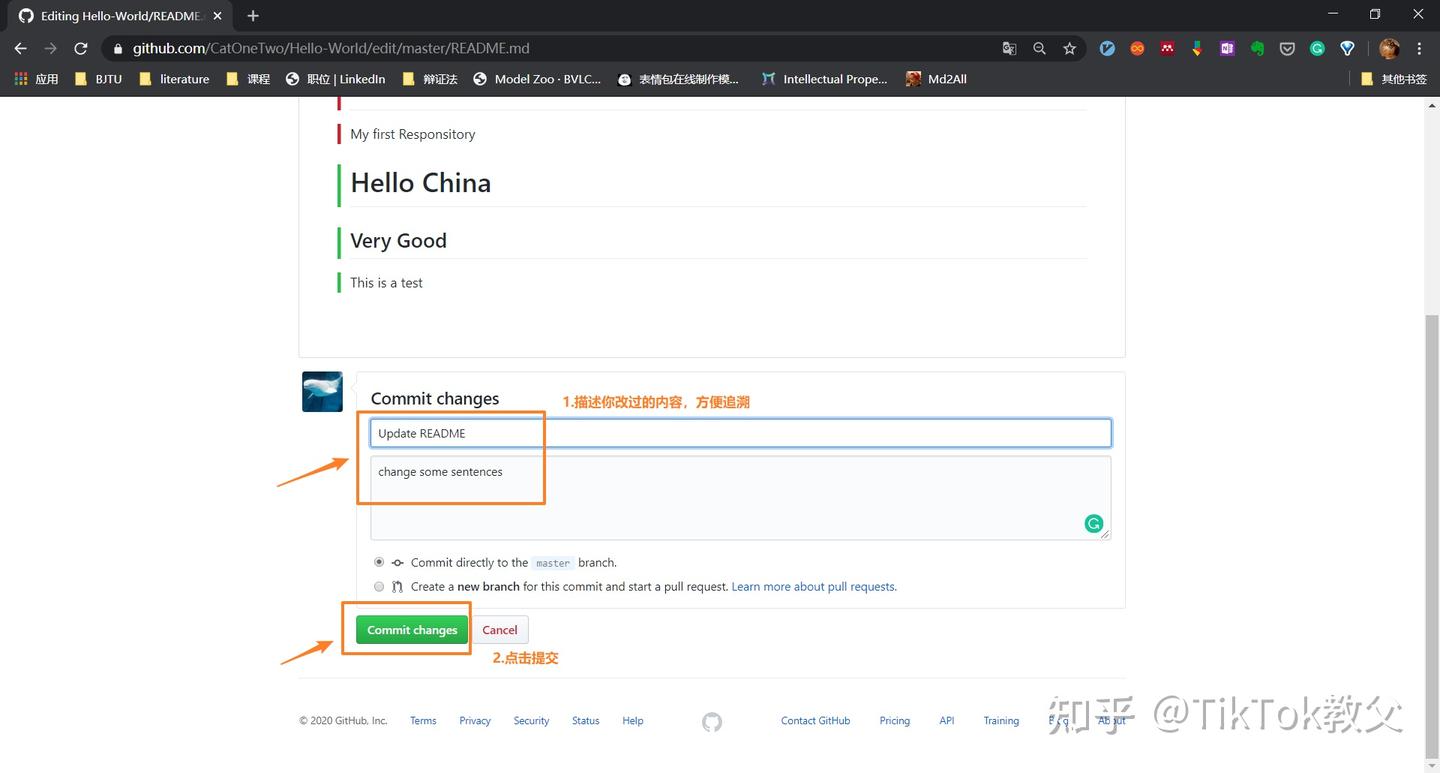
改一下文件内容。



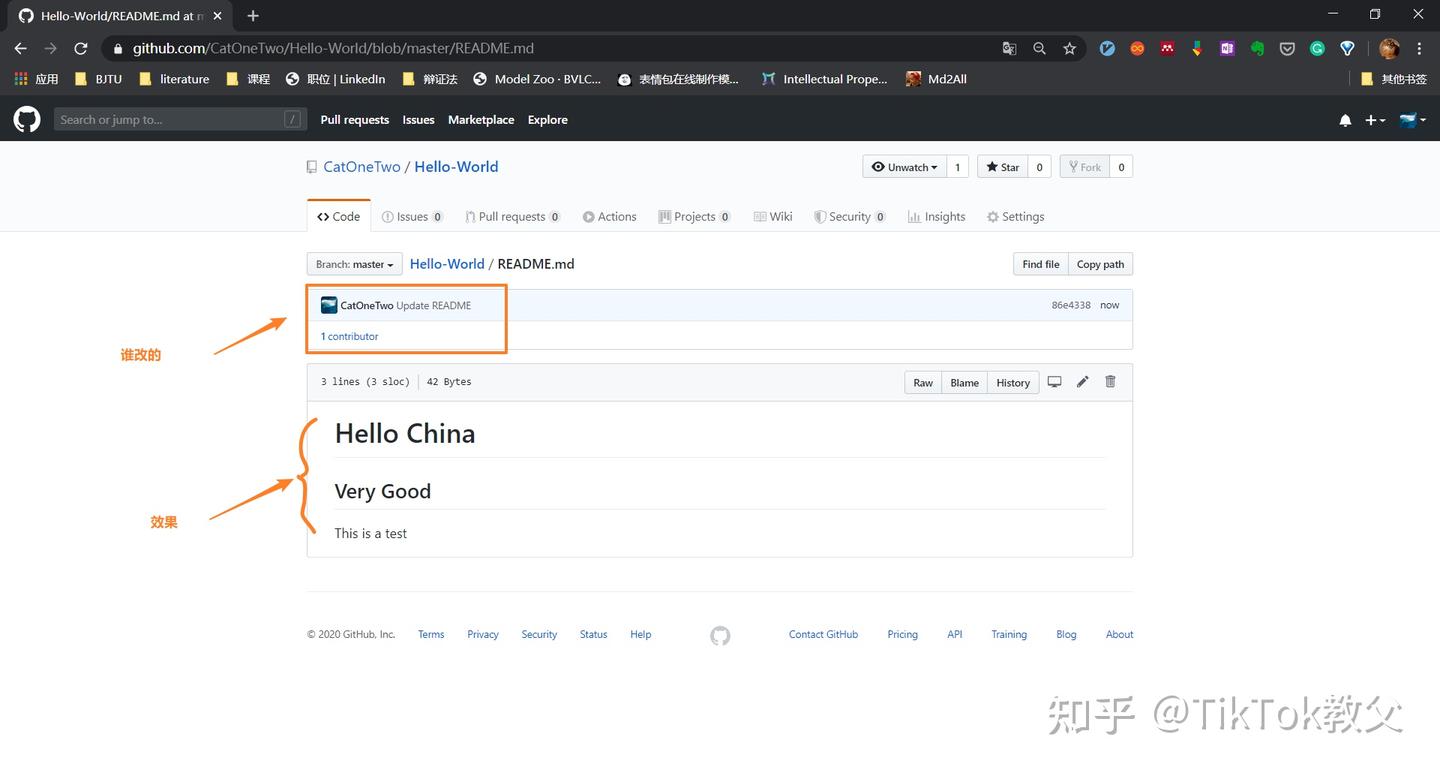
点击**Preview changes**预览更改，我们可以看到新内容以绿色显示。



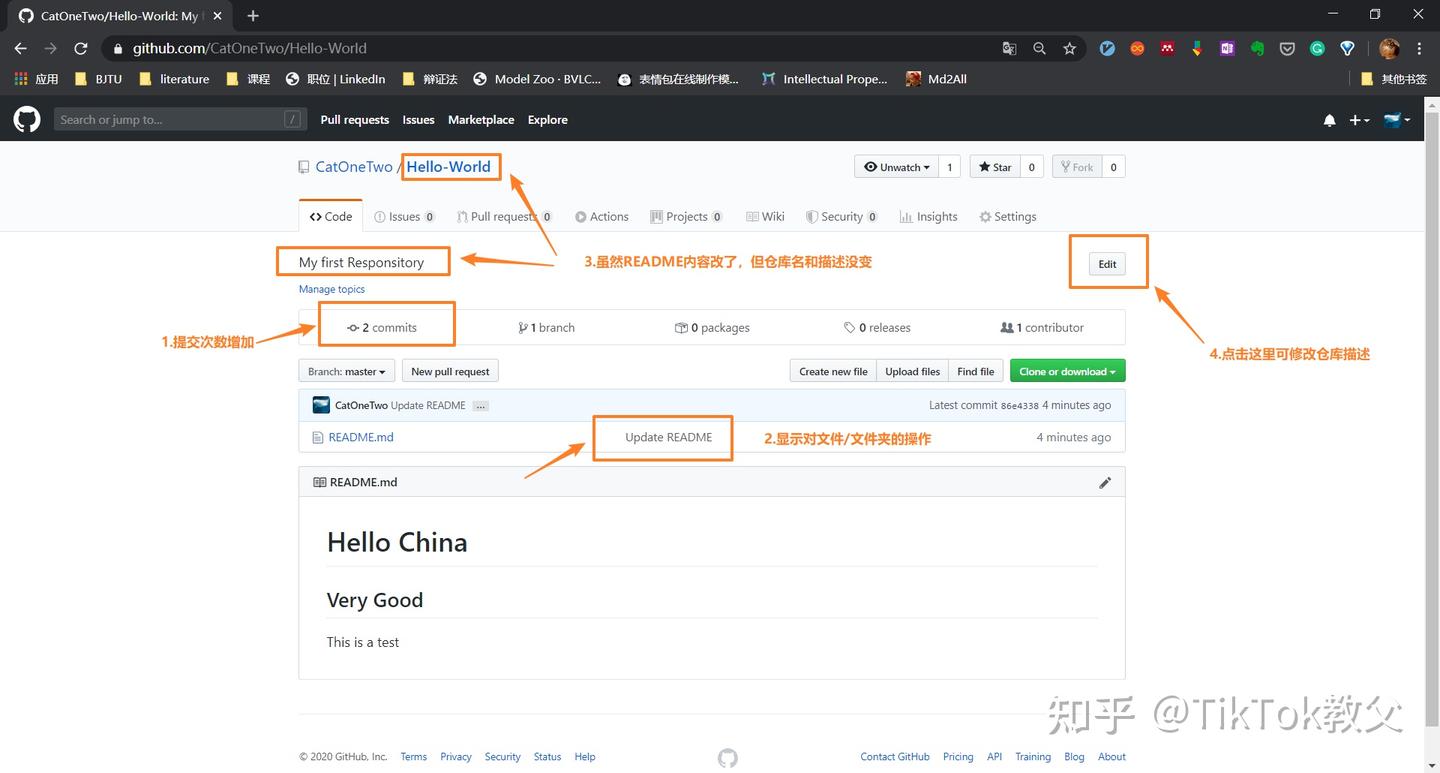
在页面底部，输入一条简短、有意义的提交消息，描述你对文件所作的更改。如果不写这条消息，以后回顾的时候不好追踪。写好之后点击提交。



提交后，页面刷新，显示README更改后的效果，可以看到 README 的标题不一定和仓库名一样，它是可以修改的。



点击仓库名 Hello-World 回到仓库主页面，可以看到提交 **commit** 的次数增加变为 2，README.md 文件页显示了刚刚的操作。



随着你更新仓库，README 也要及时更新。

总结：我们认识到，仓库里的内容不能随便改，你的每次**增删改**都需要提交/ commit，并且写好本次的操作描述，这样以后才能了解自己更新仓库的历史记录。

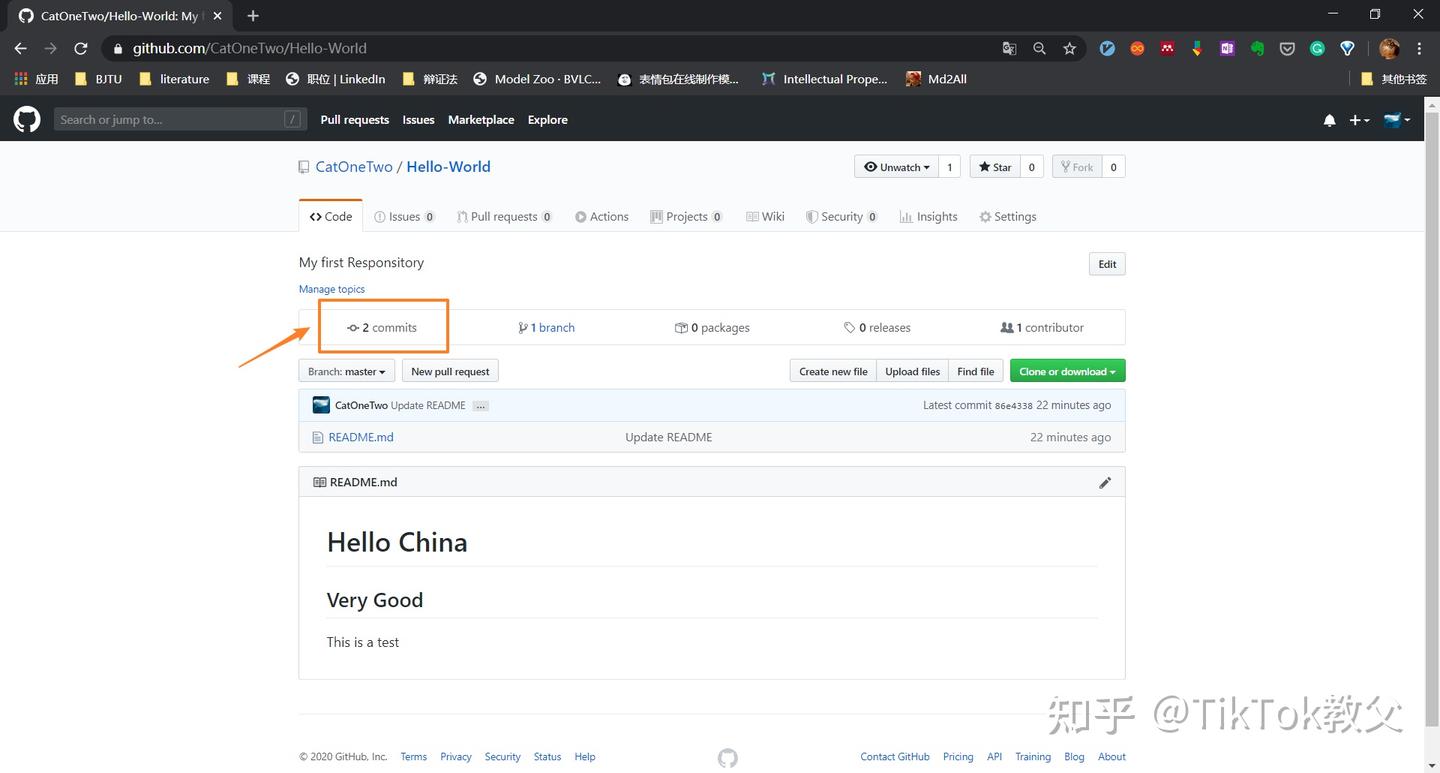
**3. 查看更改**

导读：这部分学习查看提交历史

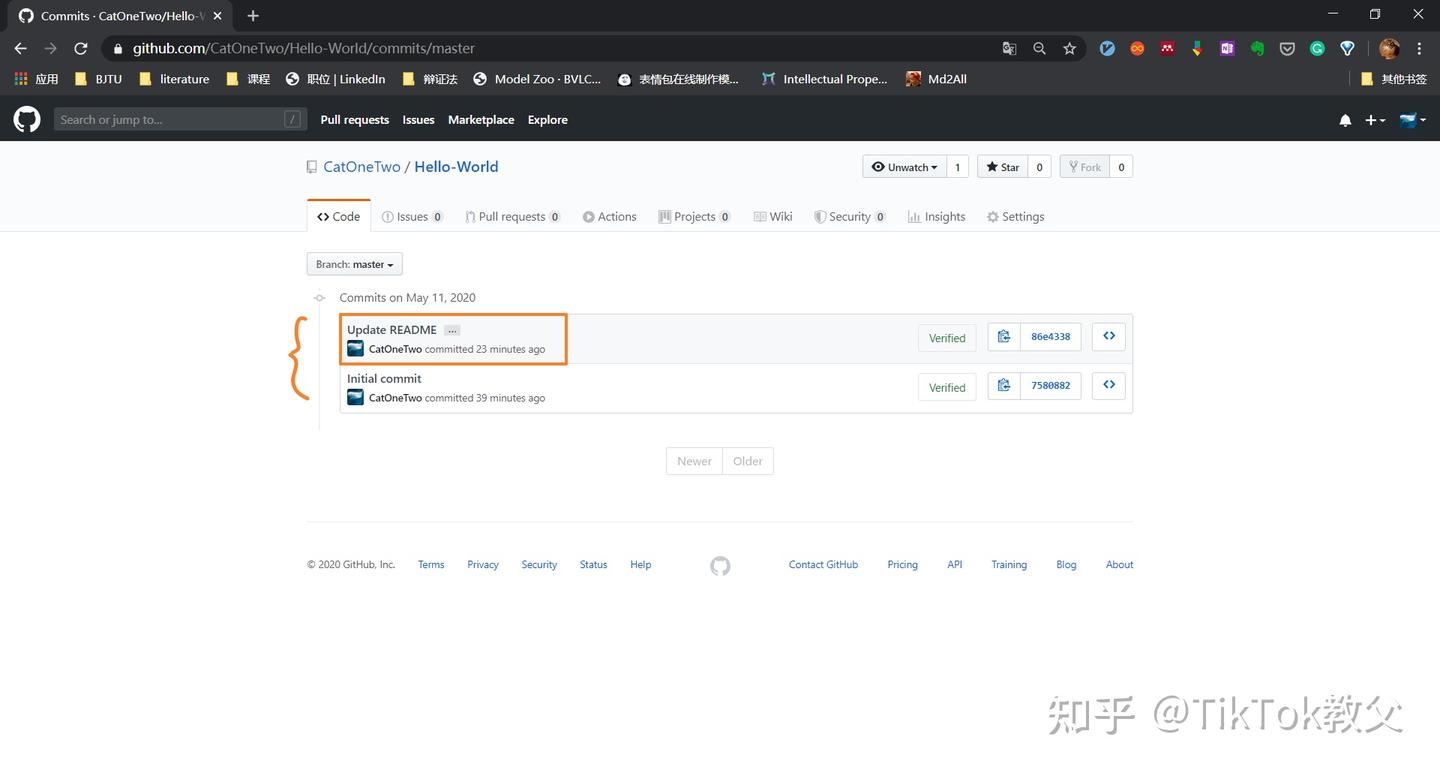
刚刚我们演示了两次提交（第一次是初始化，第二次更改了文件）。

如果有一天你进入自己或别人的仓库，看到它已经不是最初的模样，你想知道究竟发生了什么，谁动过这个仓库，这时候 Commits 就派上用场。

首先 Commits 数反映了提交更改的次数，点击它，我们就能看到提交历史。

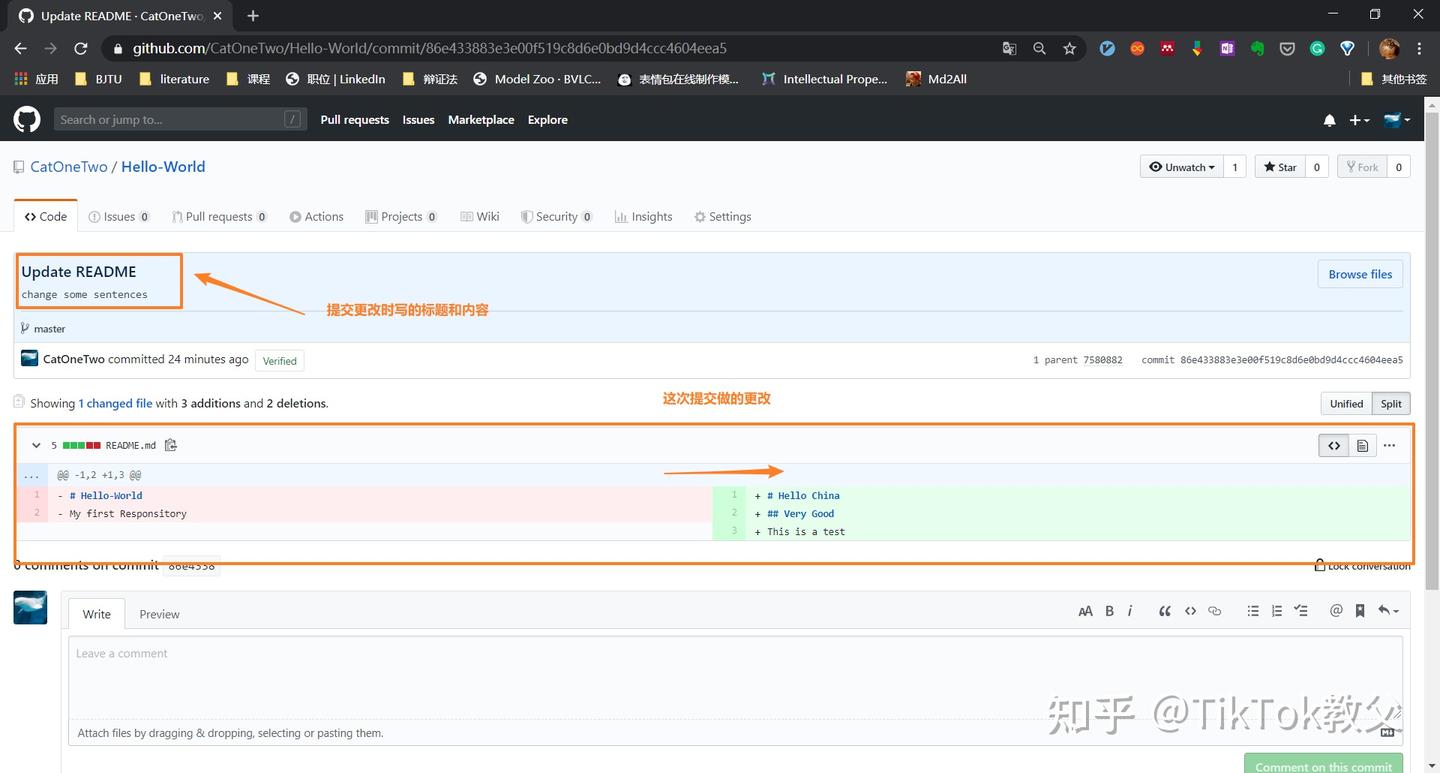


我们看到有两条提交记录，按时间倒序排列，点击最新的一条。



我们看到了这条提交记录的具体内容。上面就是这次提交的标题和内容，下面就是改动前后的对比图。

试想，如果有大量的文件修改，而我们提交时不写相关信息，我们回头查看时怎么能清楚做的改动呢？



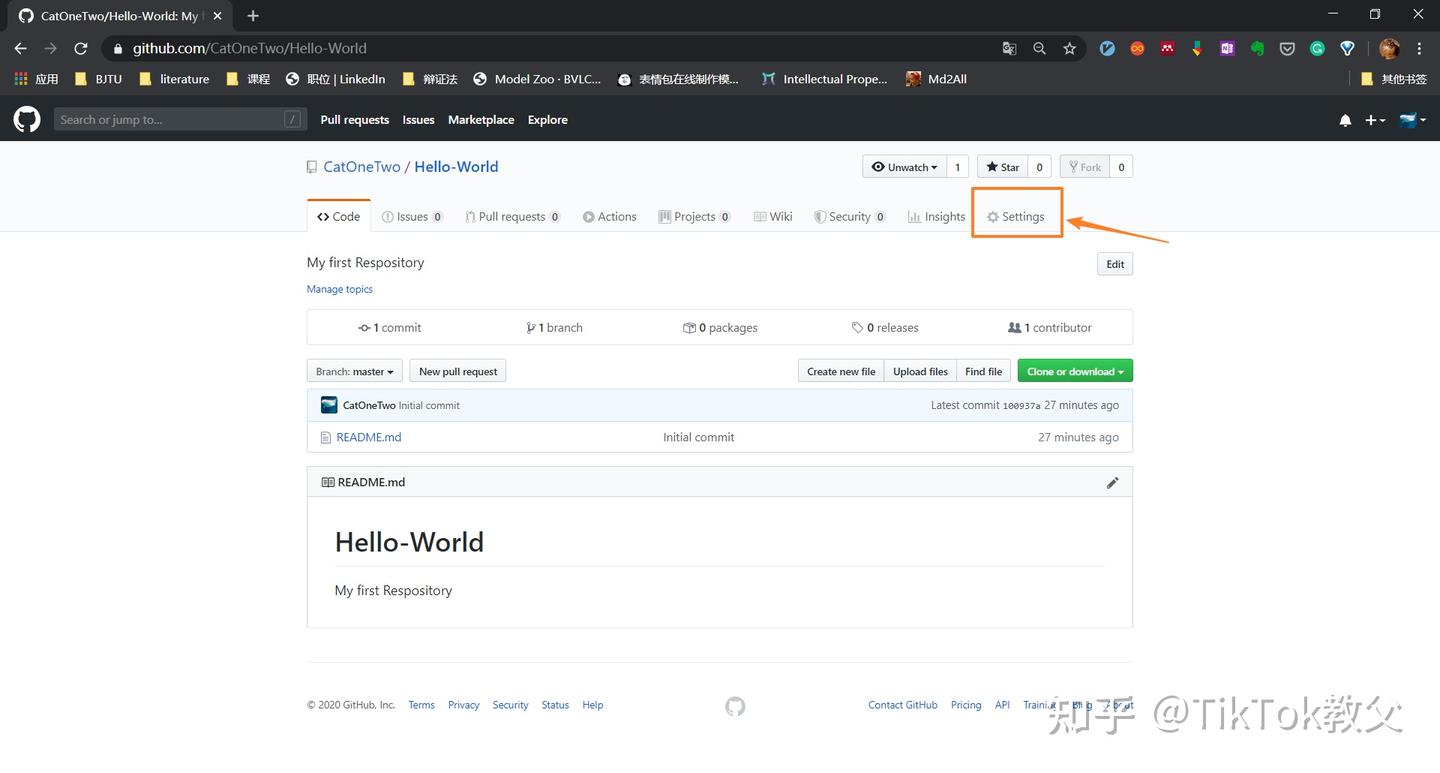
总结：提交要备注

**4. 名称修改**

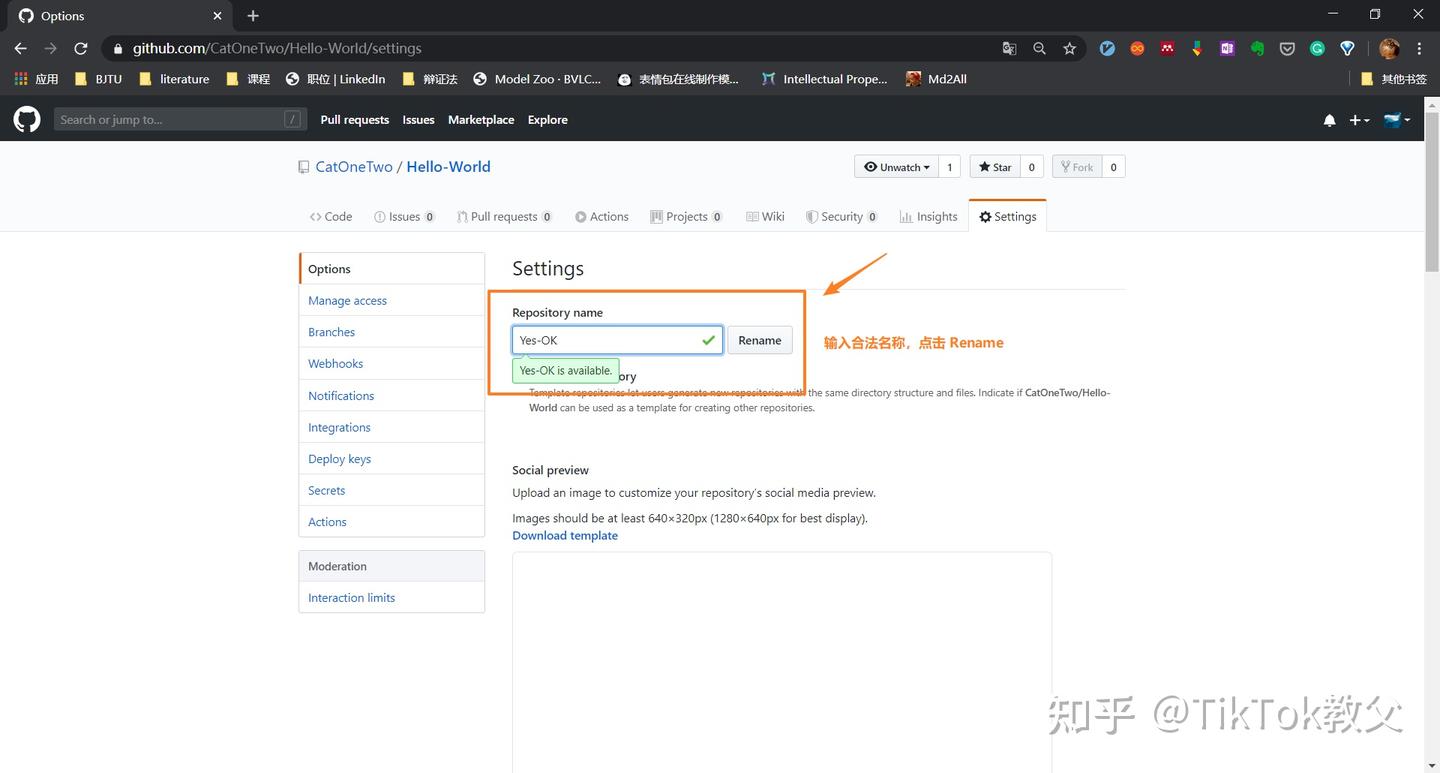
导读：改仓库名

也许有一天，你觉得自己的仓库名不够炫酷，想改掉，怎么办？

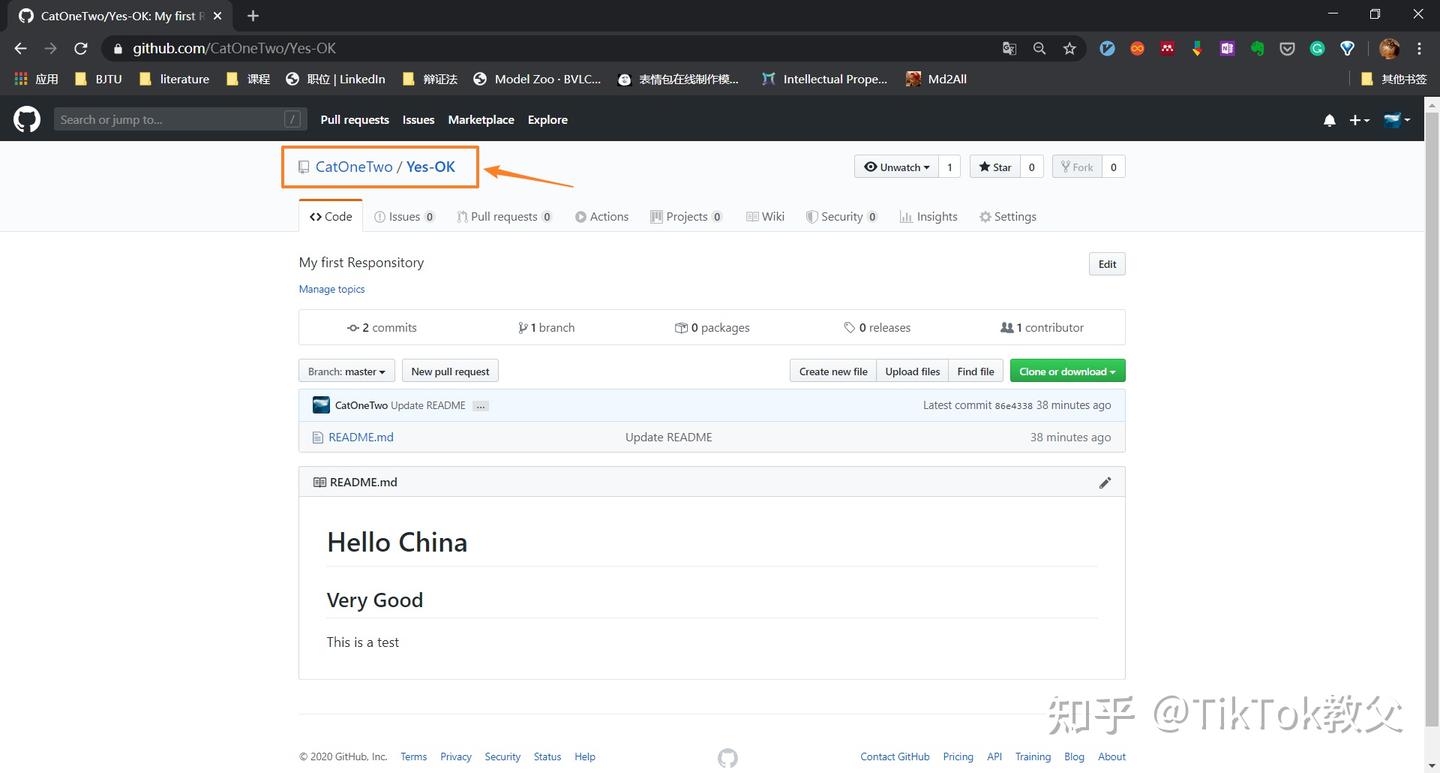
进入仓库后，点击右上角的 **Settings**：



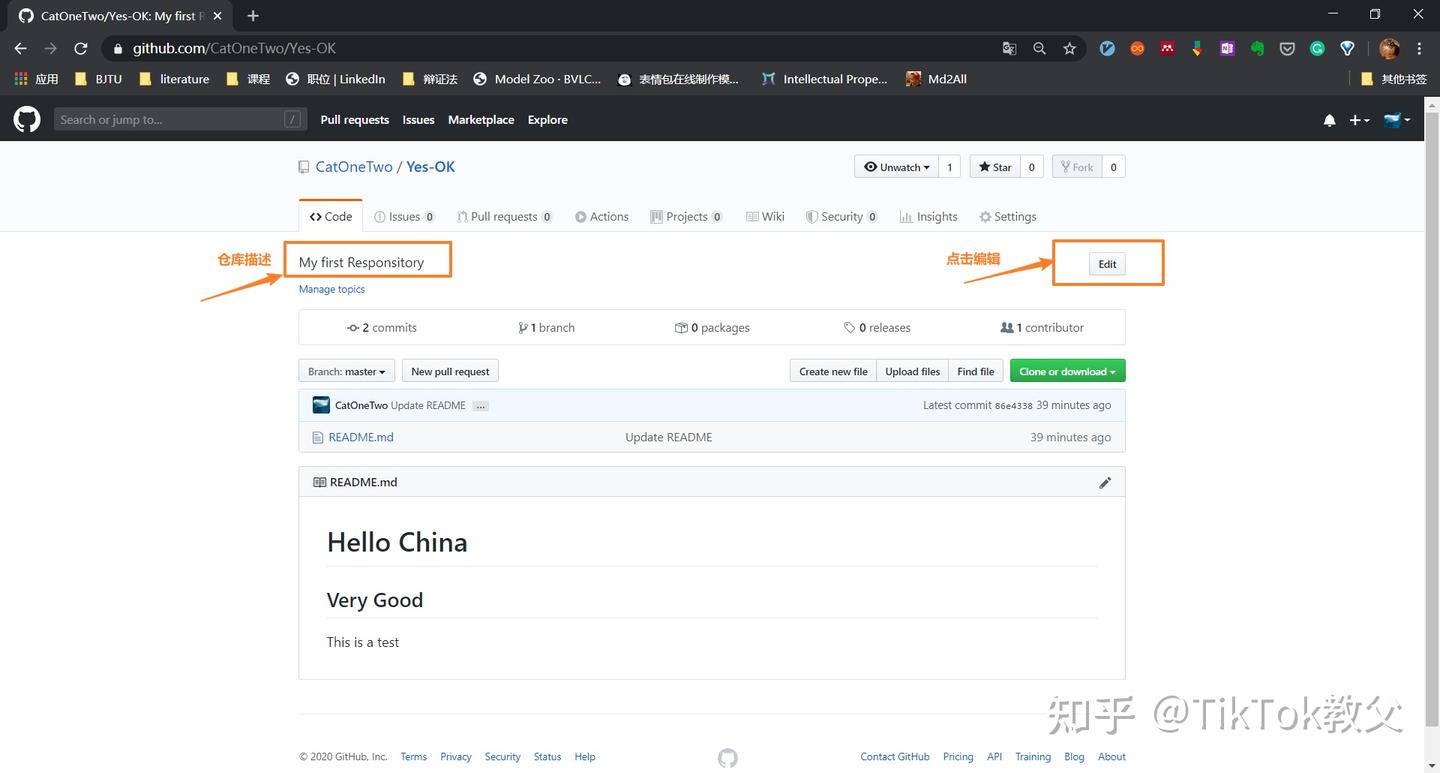
最上面就是改名的地方，输入新名称，点击**Rename**按钮。



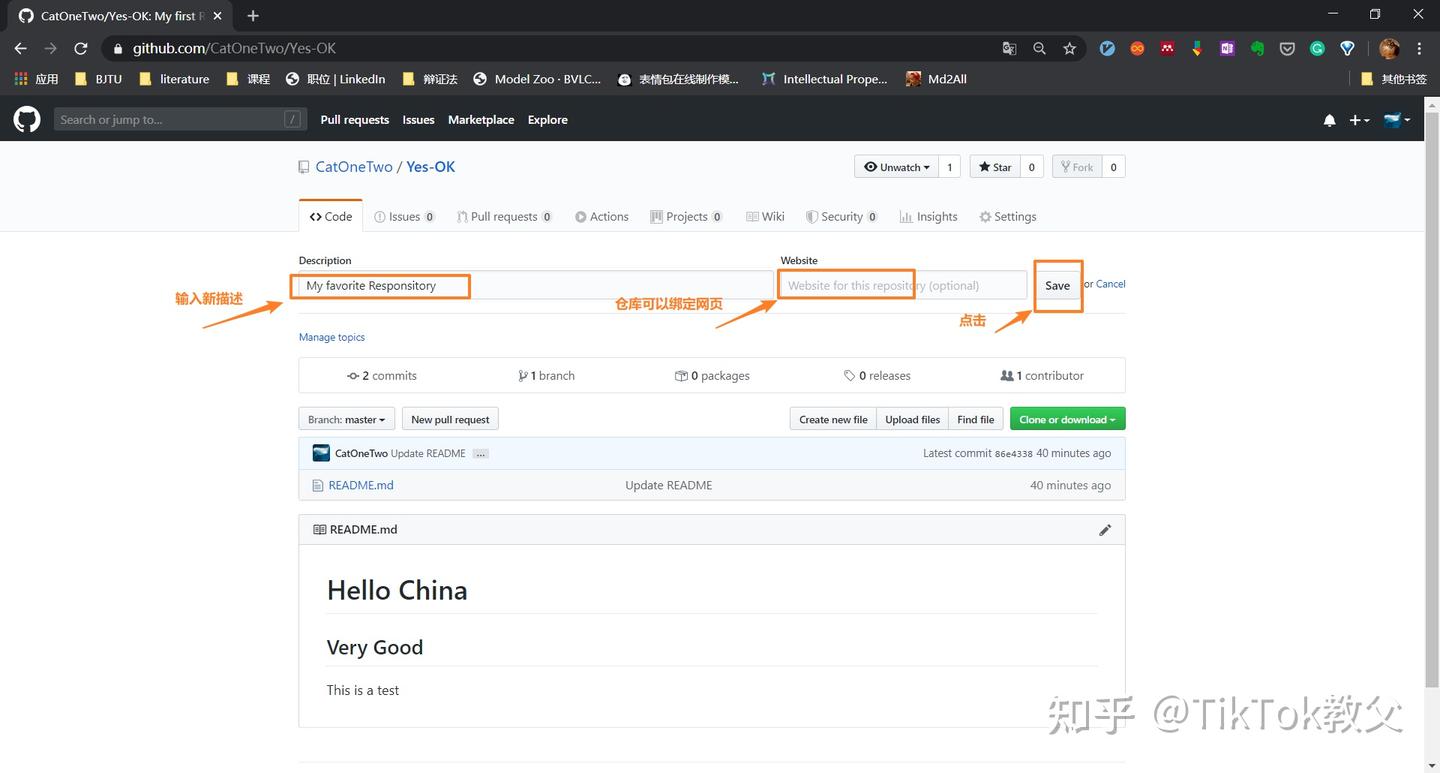
页面刷新，可以看到我们的仓库已经换成新名字啦！



当然，你也可以改仓库描述，点击描述右边的**Edit**按钮。



输入新的描述，保存即可！

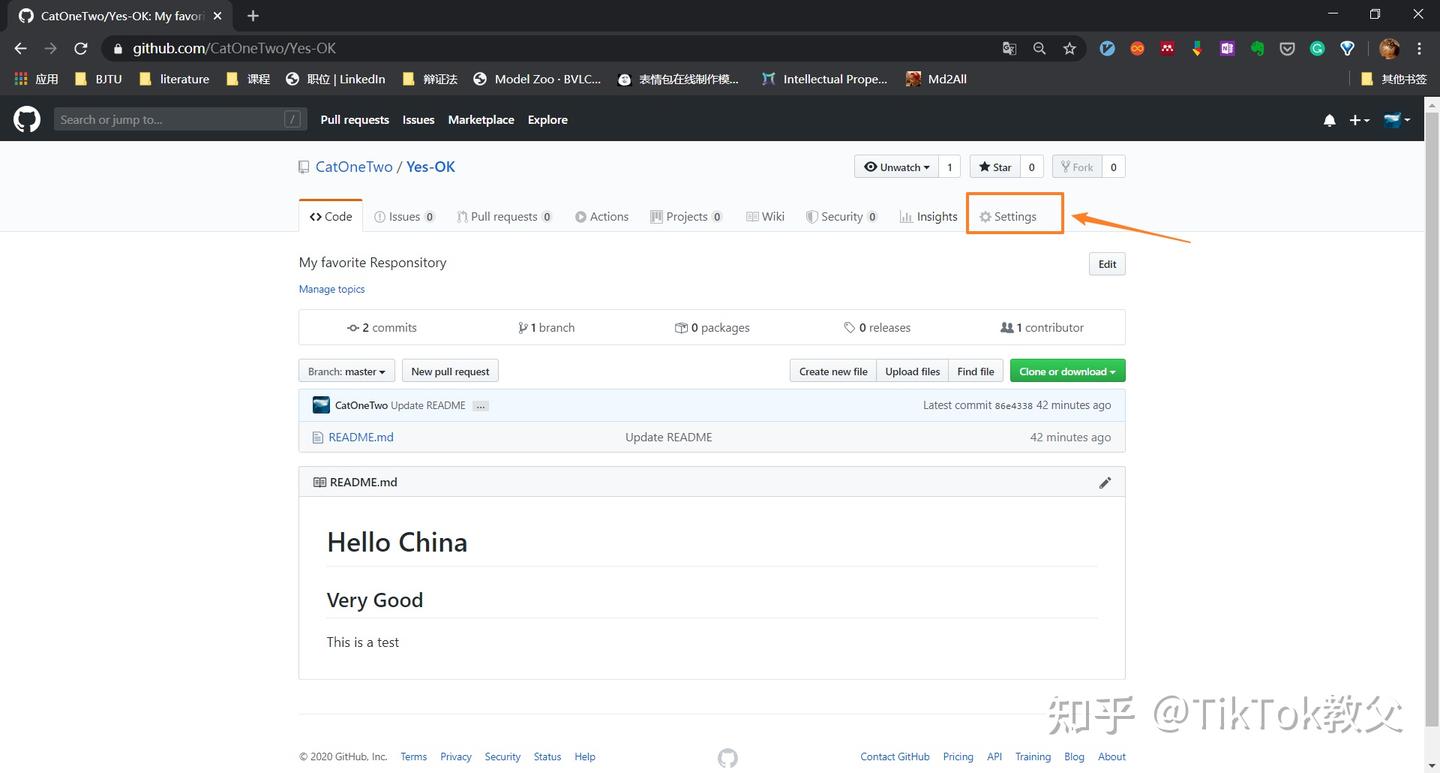


总结：更改不难

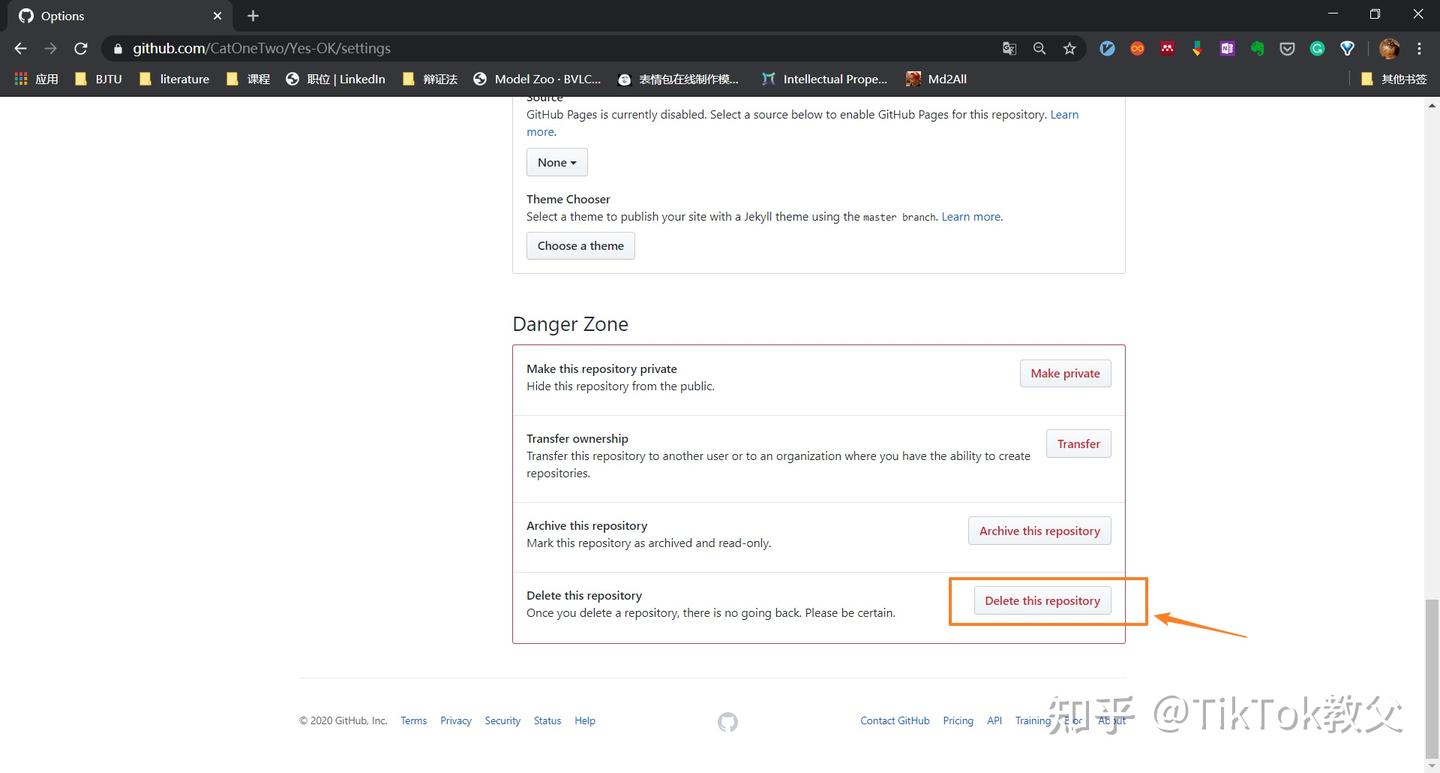
**5. 删除仓库**

导读：上面我们只是做个示例，你可能还没想好建一个怎样的仓库，那就先把示例仓库删掉吧

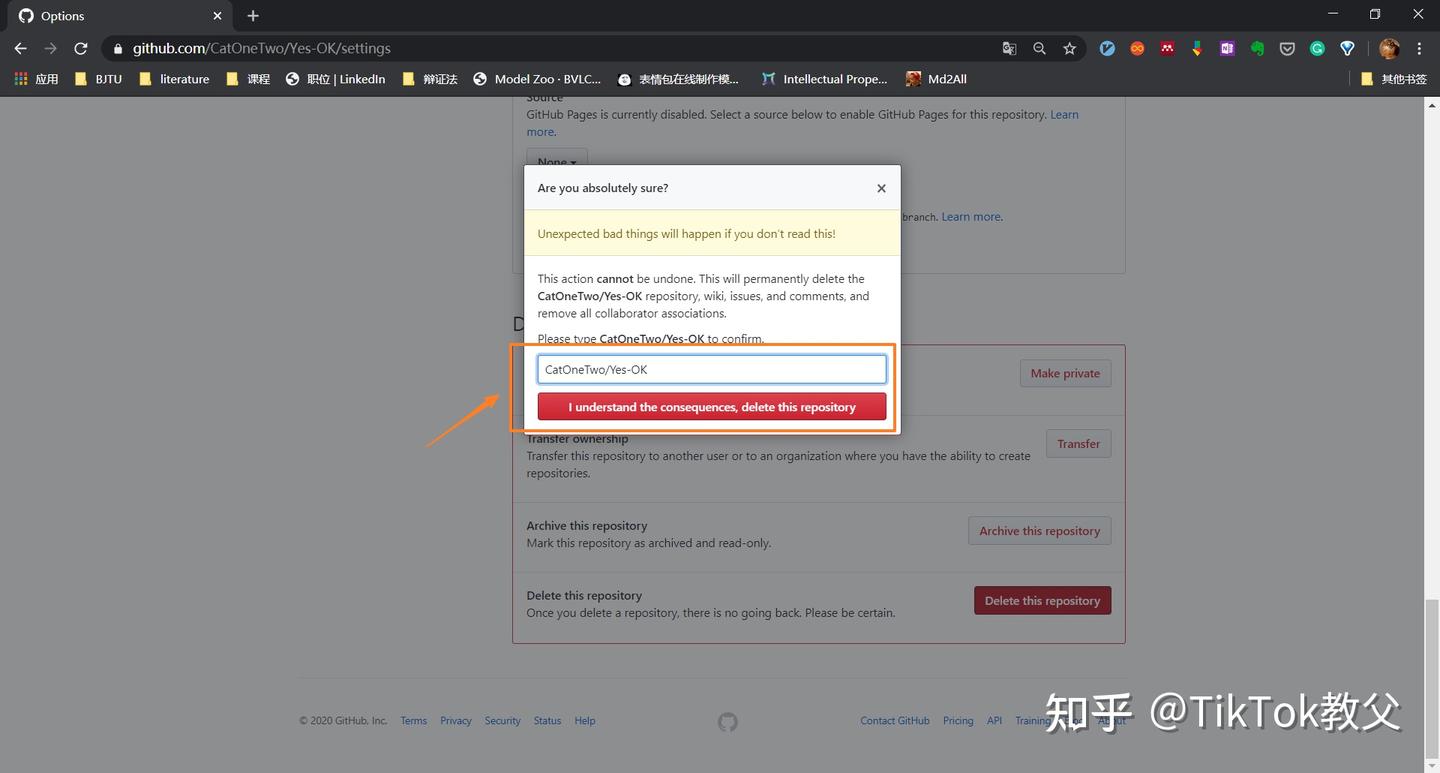
进入仓库后，点击右上角的**Settings**：



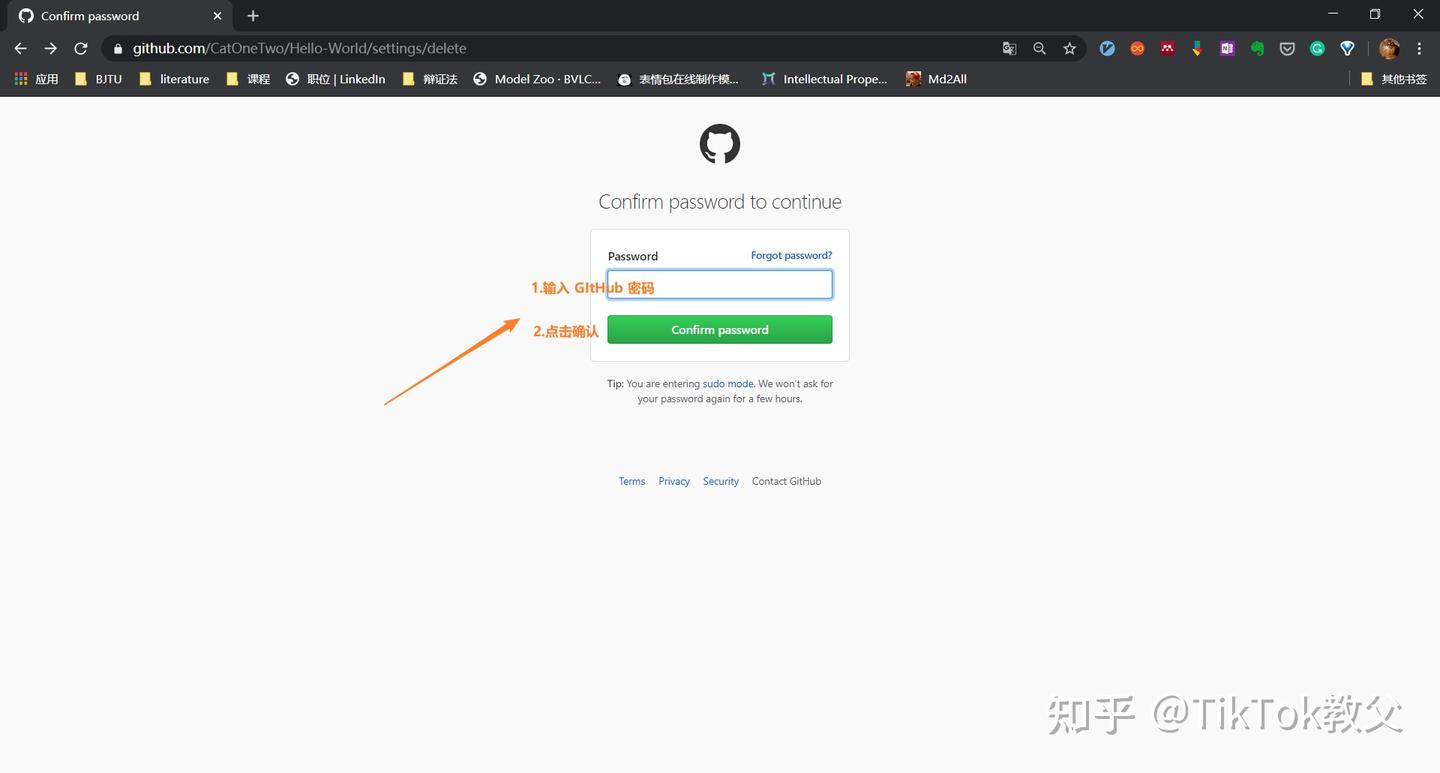
进入的设置页面可删除仓库，删除按钮在页面最下面。



删除需要你输入作者/仓库名，防止误删。



为了进一步保证安全，如果你是自动登录的话，还要输入密码。



到此为止，就删除了仓库，页面刷新，最上端会有删除成功的提示~

总结：删除需谨慎！

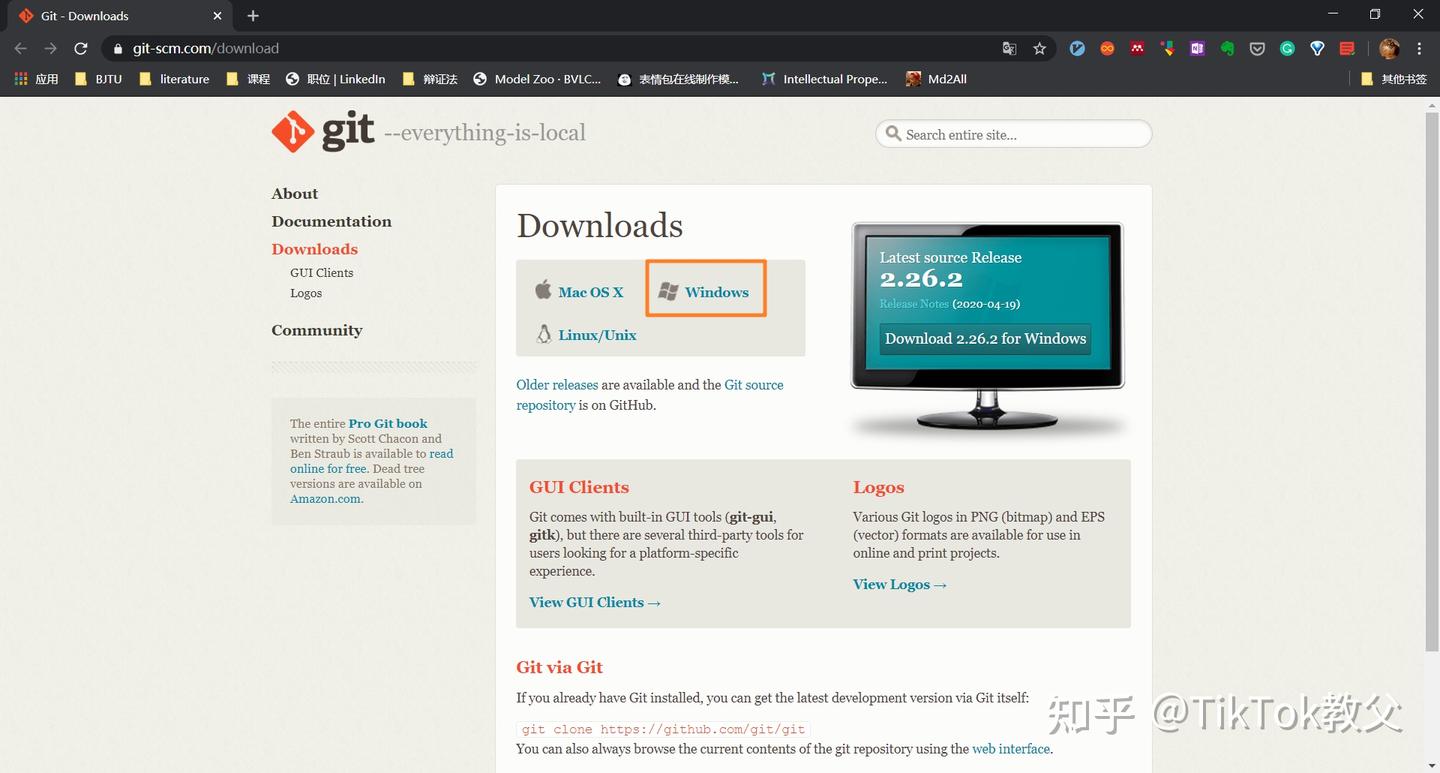
**三、Git的安装与使用**

之前我们了解到 Git 是一个版本控制工具，GitHub 基于 git 完成了版本控制功能。因此，我们要想进行代码托管，或者进行团队协作，这都少不了一个工具，那就是：**Git**  
本篇文章我们将在本地学习 **Git 的安装及基础使用**，让我们开始吧。

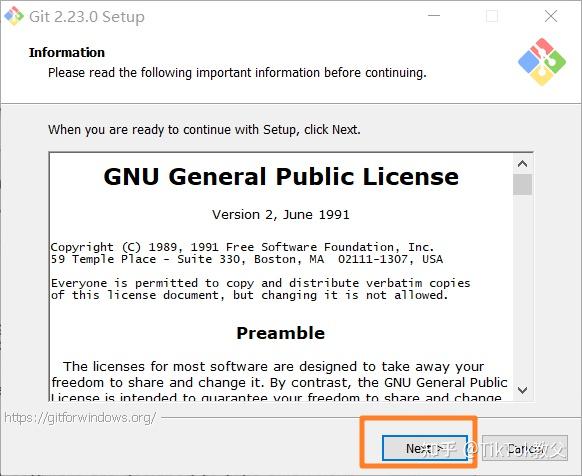
**1. Git 安装**

导读：我的操作系统是 Windows 10，Git 安装比较容易，按默认选项一直 next 即可。

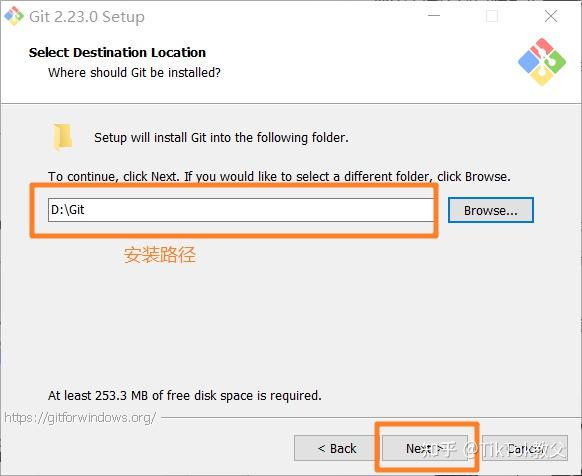
首先，去官网[https://git-scm.com/downloads](https://link.zhihu.com/?target=https%3A//git-scm.com/downloads)下载你的操作系统对应的版本。



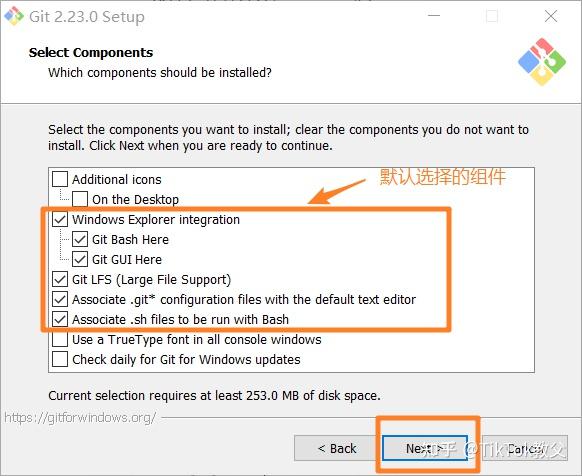
点击下载好的安装文件，就是下面这个窗口，然后点击**Next。**



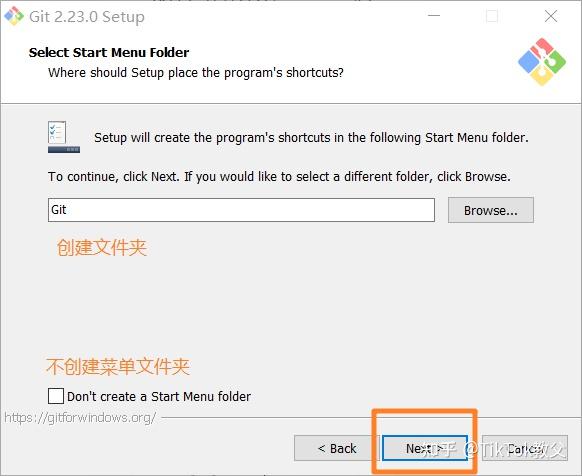
默认安装在 C 盘的 Program Files 下，你可以选择更改安装路径，比如 D 盘，然后点击**Next**：。



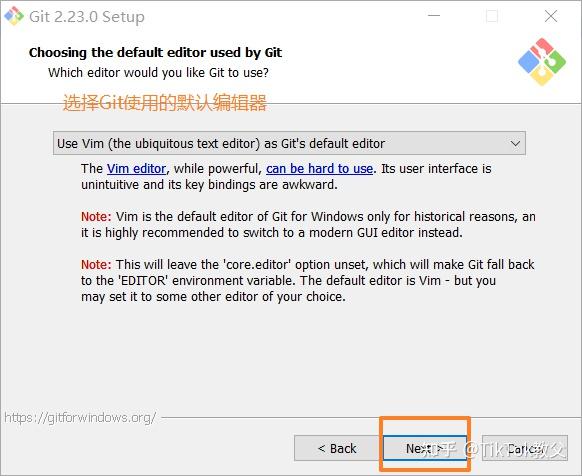
默认安装的组件如下，不用管（如果想创建桌面快捷方式，可以选第一个），直接点击**Next**：。



在开始菜单创建一个 Git 命名的文件夹，默认即可，点击**Next**：



选择 Git 使用的默认编辑器，有 vim 、notepad 等选项，可以选其他的，我选择了默认的 Vim，点击**Next**:



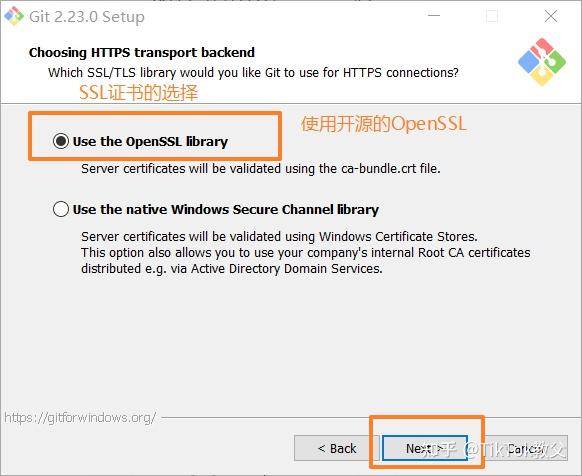
下一步是修改系统的环境变量，三个选项依次为：

* 仅使用 [Git Bash](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=Git+Bash&zhida_source=entity) 进行操作；
* 在选择使用 Git Bash 进行操作的同时，也可以使用 Windows 命令行操作，建议选择此项；
* 在选择使用 Git 的同时，也把 Unix 工具加入到了我们的配置之中，而且此操作会覆盖 Windows 的一些工具，强烈不建议选择此项。

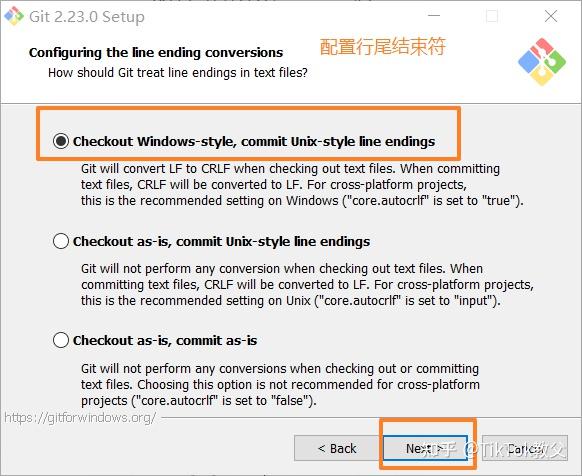
我们选推荐的第二项就好，点击 **Next**·：



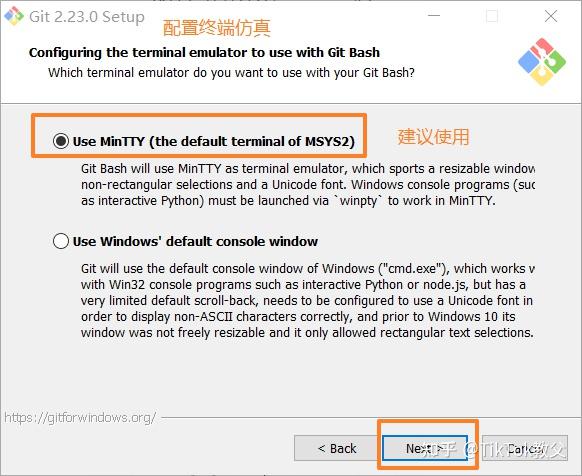
选择 HTTPS 传输后台，默认即可，点击**Next**：



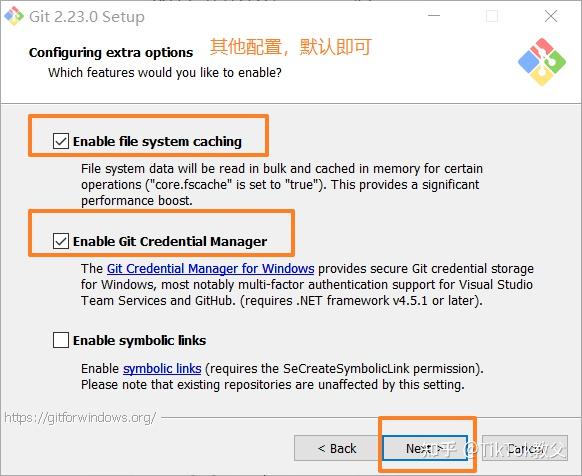
配置行结束标志，选择默认即可，点击**Next**：



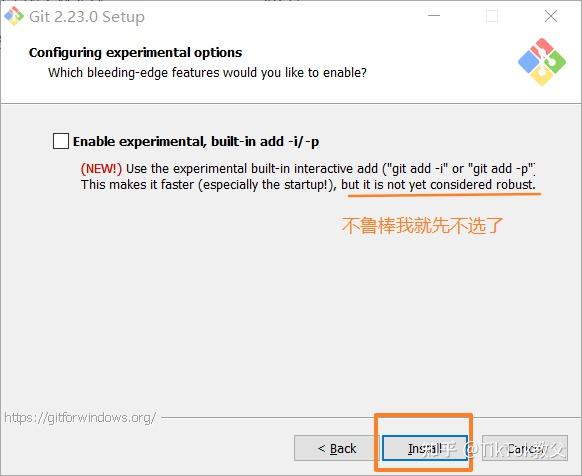
配置 Git Bash 的终端模拟器，默认即可，点击**Next**：



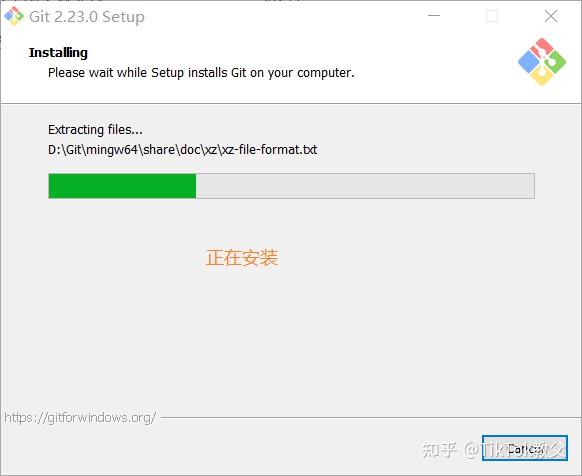
配置补充功能，还是默认即可，点击**Next**：



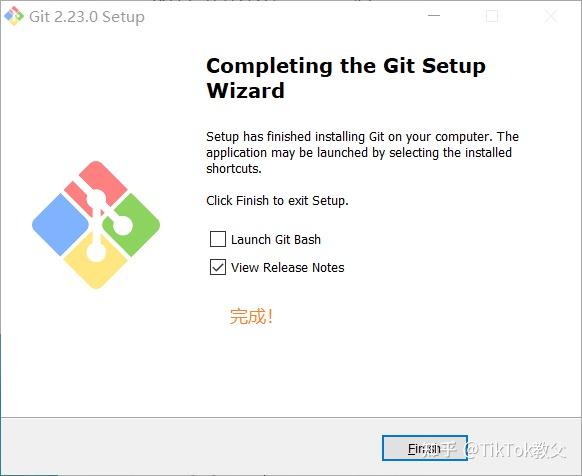
下面这个配置实验选项，因为是新功能，它也说不一定稳定，我就先不选了，点击**Install**：



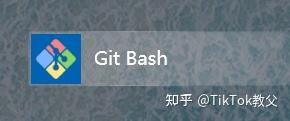
开始安装啦：



安装成功，完成！



我们测试一下，在开始菜单找到这个图标，点击 Git Bash：



输入

git --version

返回版本号说明安装成功！



**2. Git 基础**

导读：安装好 Git，我们就要学习它的语法了。其实关于 Git 有很多图形化的软件可以操作，但是建议大家从命令行学起，因为这样才能理解每一步操作的意义，之后再用图形软件完全没问题。不要抵触，基本的命令也不多，用几次就记住啦！

初次运行 Git 前的配置

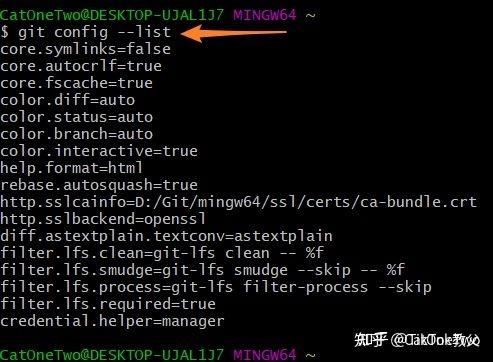
Git 自带一个 git config 的工具来帮助设置控制 Git 外观和行为的配置变量。每台计算机上只需要配置一次，程序升级时会保留配置信息。 你可以在任何时候再次通过运行命令来修改它们。

在 Windows 系统中，Git 会查找 $HOME目录下（一般情况下是C: \Users$USER）的.gitconfig文件。

我们需要配置的内容一般是**用户信息**，即用户名和邮箱。（当然也可以配置默认文本编辑器了，当 Git 需要你输入信息时会调用它。 如果未配置，Git 会使用操作系统默认的文本编辑器）

配置之前，我们可以使用命令查看已有配置。

打开 GitBash，使用第一个命令查看所有配置，通过第二个命令查看所有配置以及它们所在的文件：  
···  
git config --list  
git config --list --show-origin  
···  
这里我演示一下第一个命令，可以看到没有显示用户名（user.name）和邮箱（user.email）



接着开始配置用户名，使用如下命令即可：

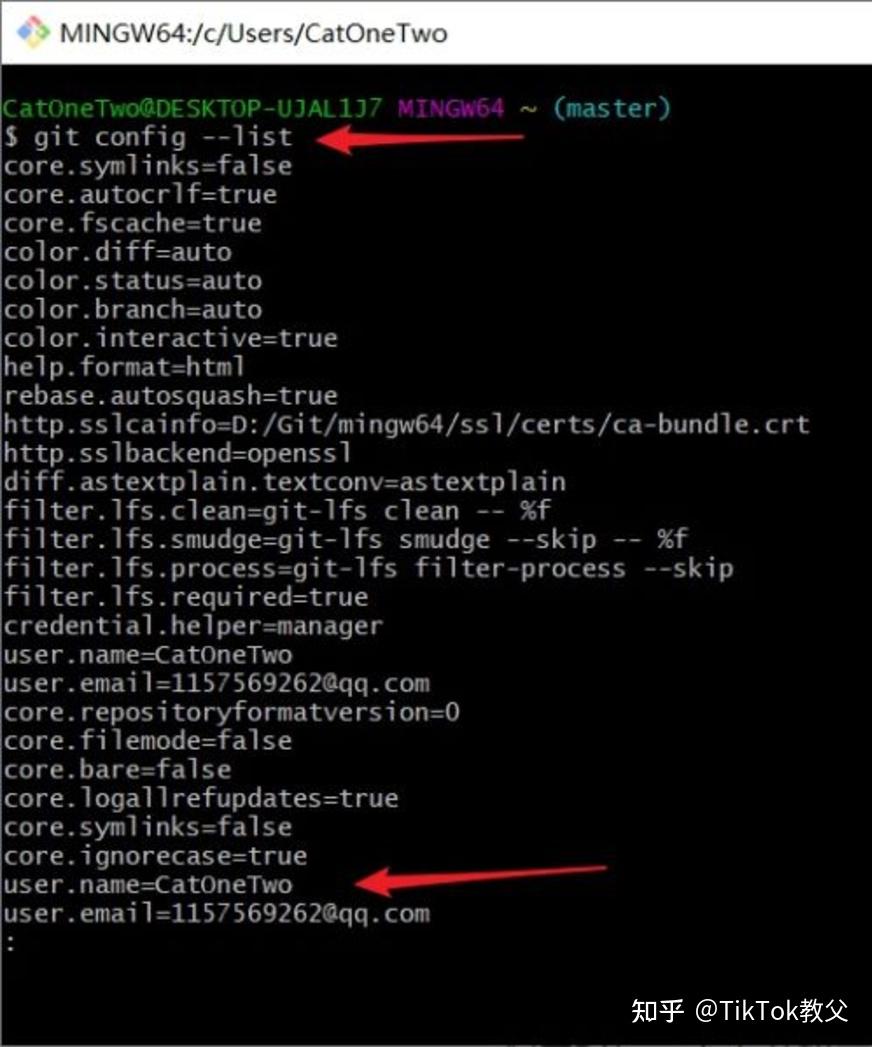
git config --global user.name "用户名"

git config --global user.email 邮箱地址



在你的 C:\Users\用户名 目录下，你会找到一个 .gitconfig 文件

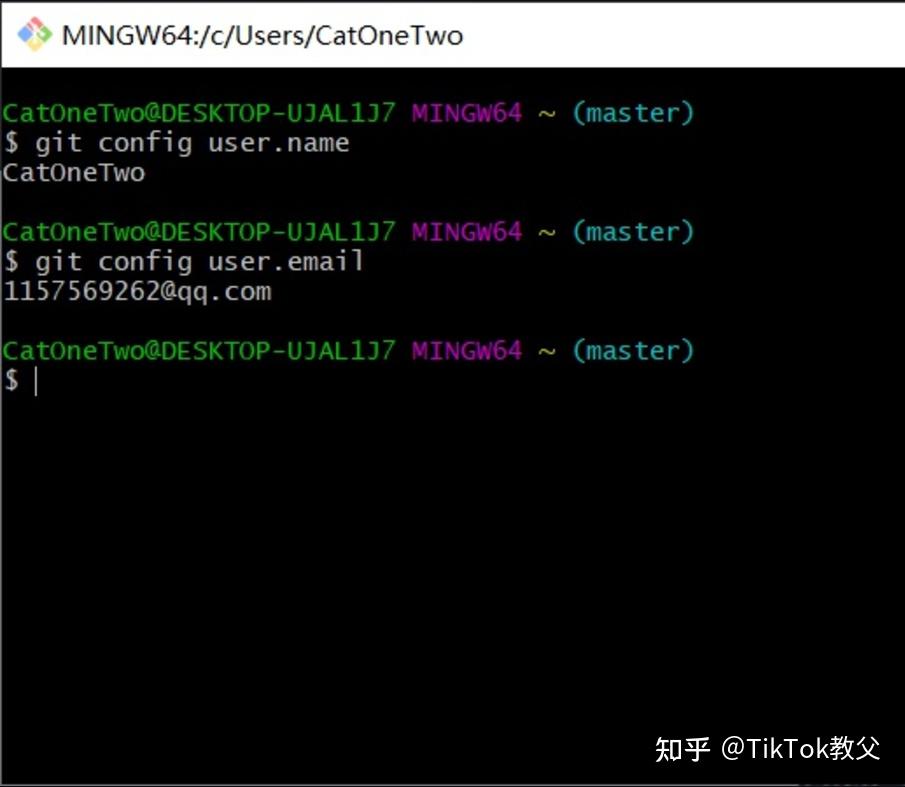
再运行检查配置的命令，可以看到最底下显示用户信息：



你也可以使用下面命令单独查看用户配置：

git config user.name

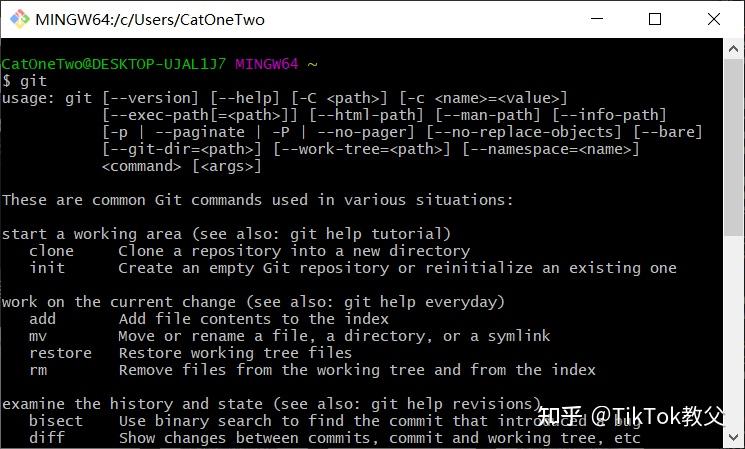
git config user.email



再次强调，如果使用了--global选项，那么该命令只需要运行一次，因为之后无论你在该系统上做任何事情， Git 都会使用那些信息。**当你想针对特定项目使用不同的用户名称与邮件地址时，可以在那个项目目录下运行没有 --global 选项的命令来配置。**

配置好后，下面开始学习 git 命令。

打开 GitBash，我们先输入 git 进行测试：



所有的操作命令都是以 git 开头的，可以看到上面列举了一些常用的命令，比如 clone，init，add 等。

为了学习这些命令，我们首先要新建一个文件夹，我选择在 E 盘下建立一个叫 Git Demo 的文件夹。

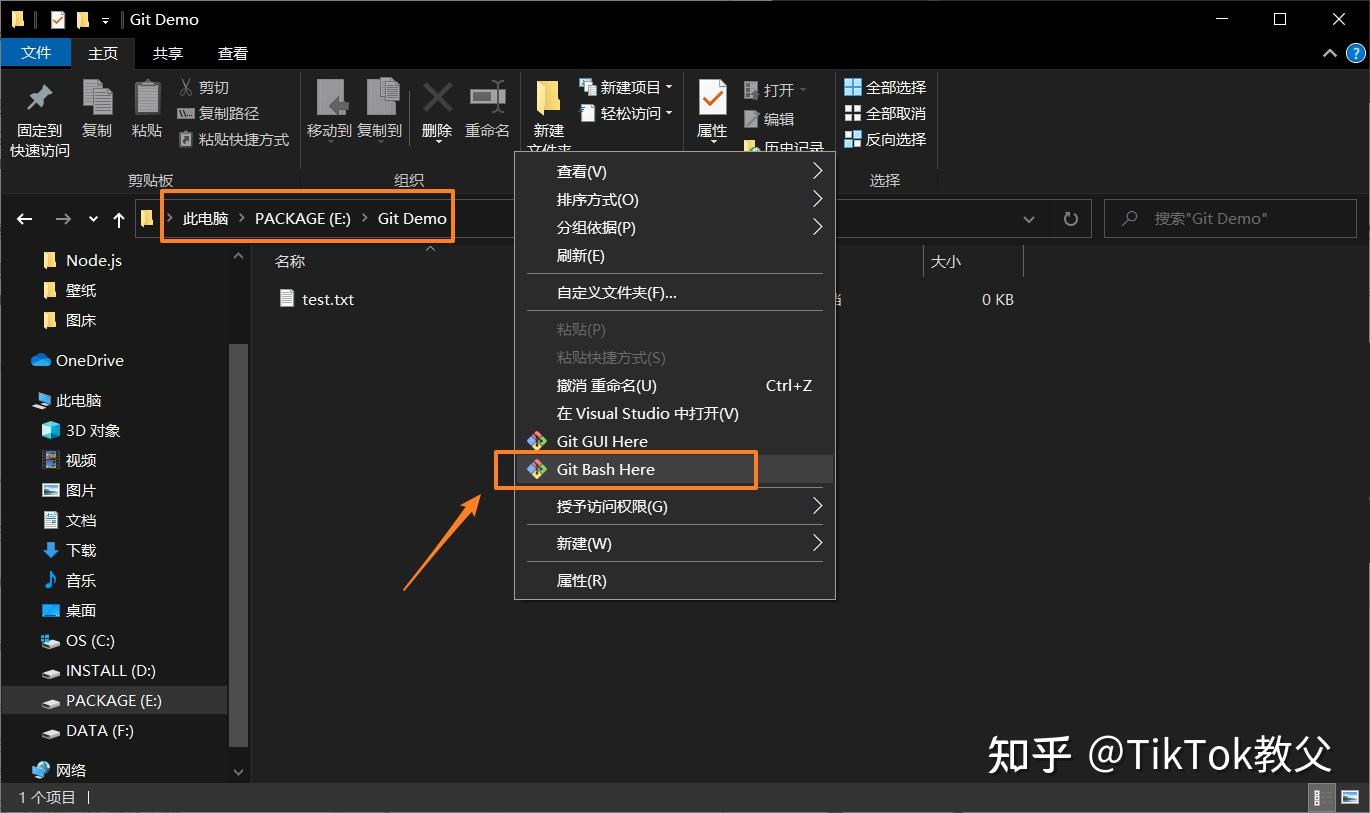
注意：在进行任何 Git 操作之前，都要先切换到 Git 仓库目录，也就是先要先切换到项目的文件夹目录下。



并在文件夹夹里创建一个文件如 test.txt。



然后我们进入这个文件夹，把它当作我们的 Git 仓库，鼠标右击选择**Git Bash Here**：



打开的 GitBash 可以看到自动定位到我们的仓库位置，如果在其他位置打开，还得再次定位，这就是我们在仓库/文件夹打开的原因。



下面我们开始 Git 操作的速成学习，一共涉及九个操作——

**2.1 git status**

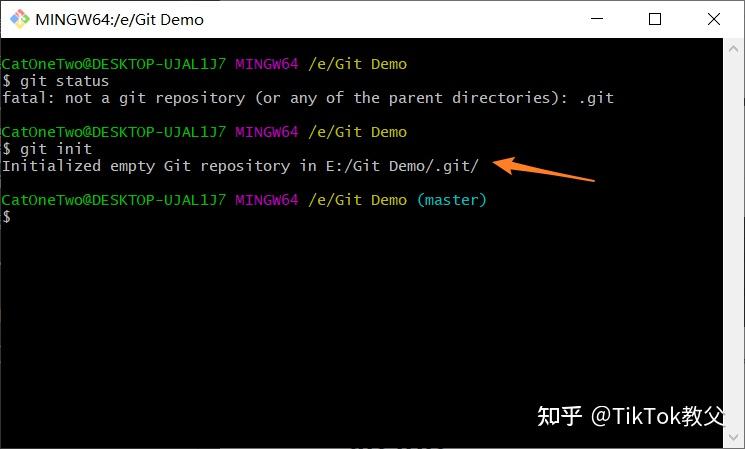
首先输入 **git status** 命令，查看仓库状态。可以看到提示，当前目录还不是一个 Git 仓库，它还只是个普通的文件夹。

以后我们可以经常使用**git status**命令查看仓库状态



**2.2 git init**

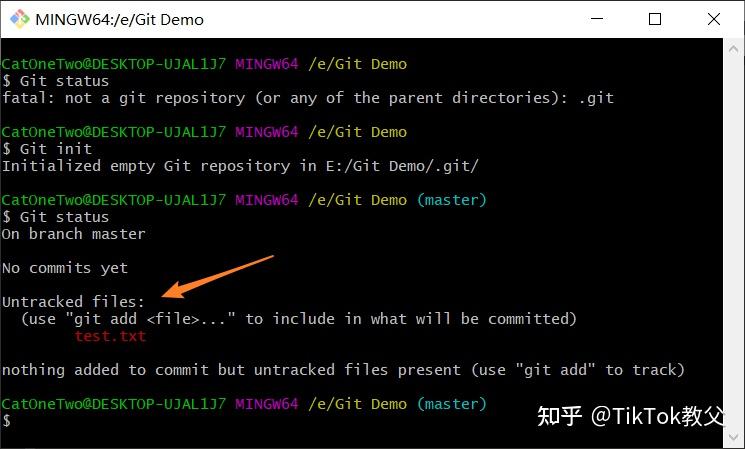
既然它还不是 Git 仓库，我们就用 **git init** 初始化，可以看到初始化完成，它是一个空的 Git 仓库。



我们再用 **git status** 命令检查一下仓库，发现 Git Demo 目录已经成为一个 Git 仓库了，并且默认进入 Git 仓库的 master分支，即主分支。

返回的三条语句：

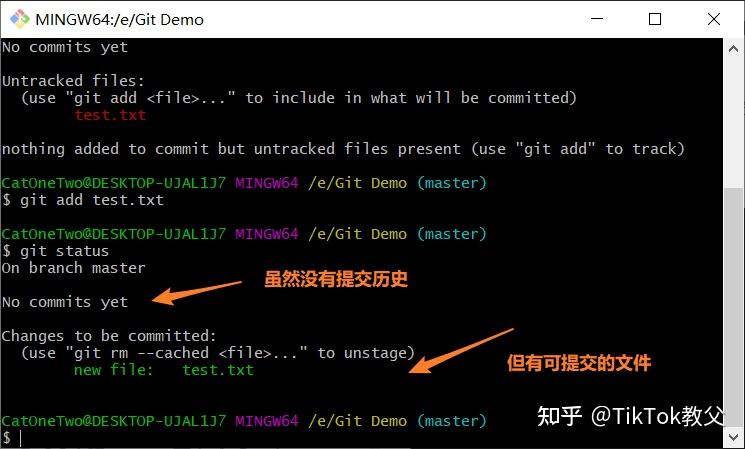
* 进入 Git 仓库的主分支
* 目前还没有提交过文件（git 是版本控制工具，要记录提交历史的以及每一个版本）
* 这时最主要的是提示 test.txt 文件是 Untracked files ，就是说 test.txt 这个文件还没有被跟踪，还没有提交在 git 仓库里呢，而且提示你可以使用 git add 去操作你想要提交的文件。



**2.3 git add**

接下来我们往仓库里添加文件，根据提示，输入 **git add test.txt**，再输入 **git status** 看看仓库状态：

虽然仓库没有提交历史，但提示以下文件 Changes to be committed ，意思就是 test.txt 文件等待被提交。



可以看到刚刚 git “不承认” 我们的文件，现在通过 **git add** 命令”承认“了，并且把它加到一个可提交的列表（临时缓冲区）。

当然你可以使用 git rm --cached 这个命令去移除这个缓存。

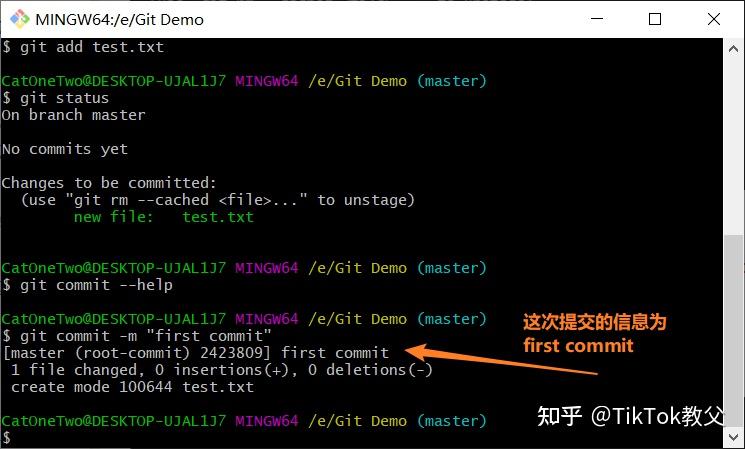
**2.4 git commit**

刚刚我们把文件送到缓冲区，接下来用 **git commit** 命令将缓存区里的改动给提交到本地的版本库。

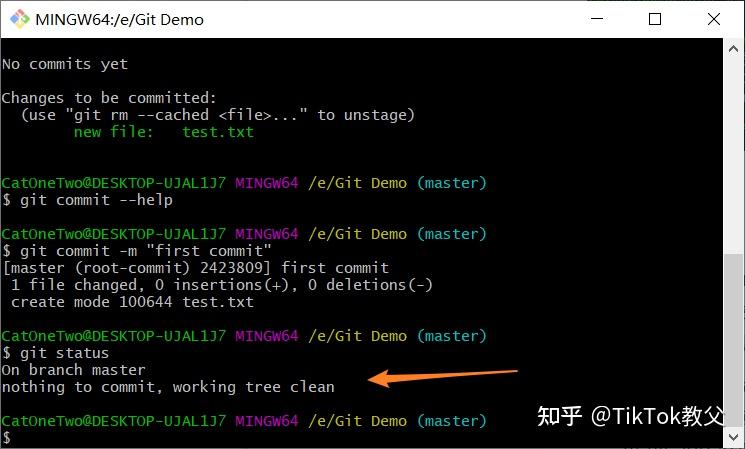
每次使用 **git commit** 命令我们都会在本地版本库生成一个 40 位的哈希值，这个哈希值也叫 commit-id（这个版本的编号），commit-id 在版本回退的时候是非常有用的，它相当于一个索引，可以在未来的任何时候通过与 git reset 的组合命令回到这里。

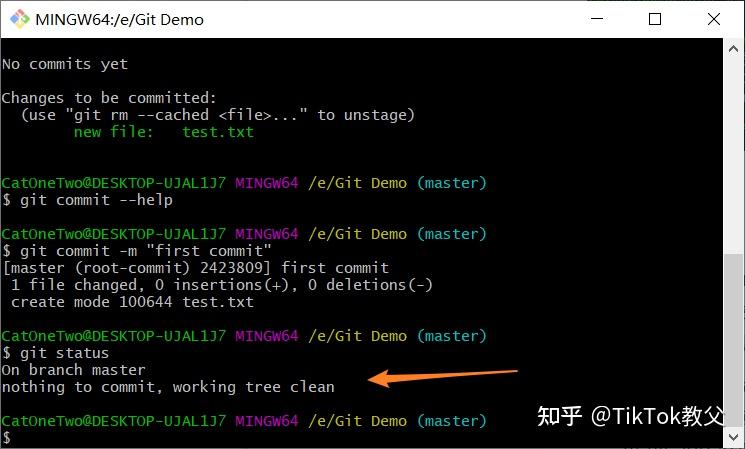
**git commit** 不是单独使用的，一般需要参数指定提交方式，我们可以输入 **git commit --help** 查看使用帮助（会跳转到相关网页）。

输入 **git commit -m ”first commit“**，我们成功将文件 text.txt 提交到了 Git 仓库。其中， -m 代表提交信息， first commit 是本次提交的信息，需要写在双引号内，提交信息你也可以写成 my commit，text commit 等任意句子。



我们再看看仓库状态，结果显示 nothing to commit, working tree clean，这表示已经没有内容可以提交了，即全部内容已经提交完毕。





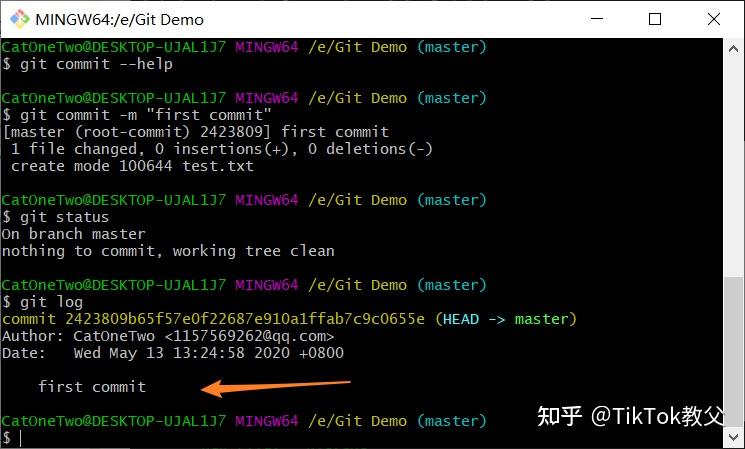
为什么不把文件直接提交到仓库呢因为我们可能提交一个错误的文件，先 add 再 commit 能有效防止错误提交。就像坐火车需要到候车室（缓冲区）等待一样，直接走到火车群里可能上错车。当然也有办法把这两步合并成一步，不过后面再介绍，建议新手先按部就班的一步步来。

补充**git commit**常见的用法有下面三个：

* **git commit -m “message”**，-m 参数表示可以直接输入后面的 “message” ，如果不加 -m 参数，那么是不能直接输入 message的，而是会调用一个编辑器一般是 vim 来让你输入这个 message。
* **git commit -a -m “massage”**，加的 -a 参数可以将**所有**已跟踪文件中的执行修改或删除操作的文件都提交到本地仓库，即使它们没有经过 git add 添加到缓存区，注意，新加的文件（即没有被 git 系统管理的文件）是不能被提交到本地仓库的。建议一般不要使用 -a 参数，正常的提交还是使用 git add 先将要改动的文件添加到暂存区，再用 git commit 提交到本地版本库。
* **git commit --amend**，追加提交，它可以在不增加一个新的commit-id的情况下将新修改的代码追加到前一次的commit-id中

**2.5 git log**

输入**git log**命令，打印 Git 仓库提交日志，会显示作者、时间和你写的提交信息：

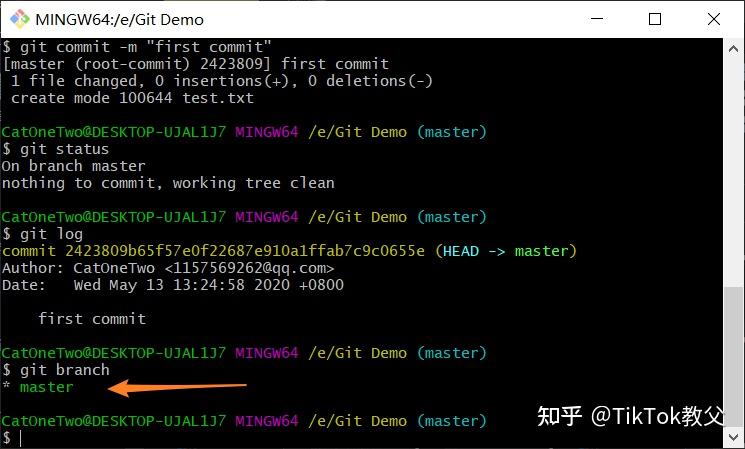


**2.6 git branch**

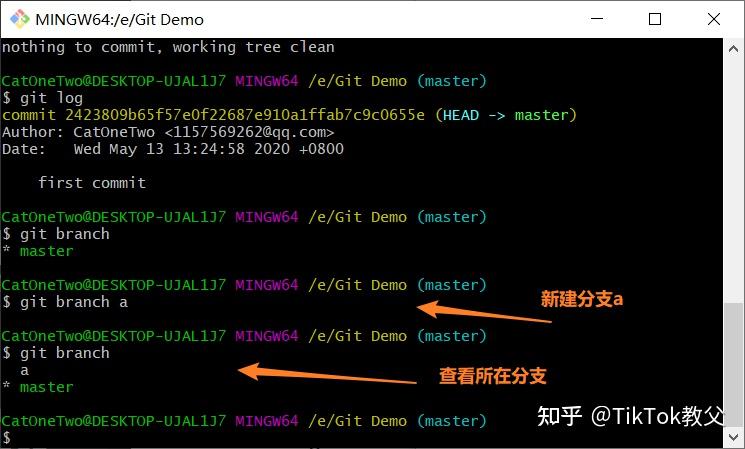
branch 是分支的意思，多人合作时，你一个模块，别人一个模块，你们各建一个分支就能保证改动互不干扰，等最后做完，把你们的分支合并起来就好。

git init 会初始化一个主分支 master，也是我们现在的默认分支。

输入 **git branch**，可以看到我们就在 master 这里：



在此基础上新建分支用**git branch a**，其中 a 是我起的分支名字。然后再用**git branch**查看，可以看到 master 有了一个分支，这时候分支 a 跟分支 master 是一模一样的内容。



**2.7 git checkout**

虽然建立分支，但可以看到 \* 还在 master 上，说明我们还处于主分支。这时候，输入 **git checkout a** 命令，切换到分支 a 上，再次查看分支，\* 在 a 前，说明我们处于分支 a ，可以修改分支 a 的内容了。



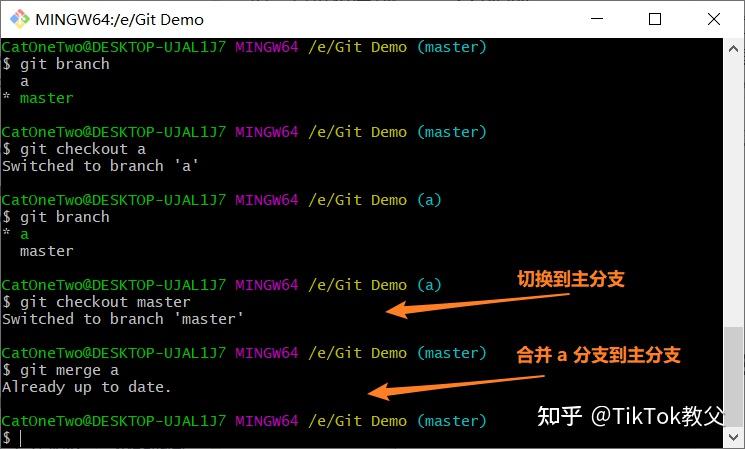
先新建再切换很麻烦，有一步到位的方法**git checkout -b a**，意思是建立分支后自动切换到该分支。

**2.8 git merge**

你在 a 分支完成了你的部分，但是主分支还是原来那样，这个时候就需要把你的代码合并到主分支 master上来，**git merge** 就是合并分支用到的命令。

**注意，合并分两步**

1. 切换到 master 分支，如果你已经在了就不用切换了;
2. 执行 git merge a ，意思就是把 a 分支的代码合并过来，不出意外，  
   这个时候 a 分支的代码就顺利合并到 master 分支来了。

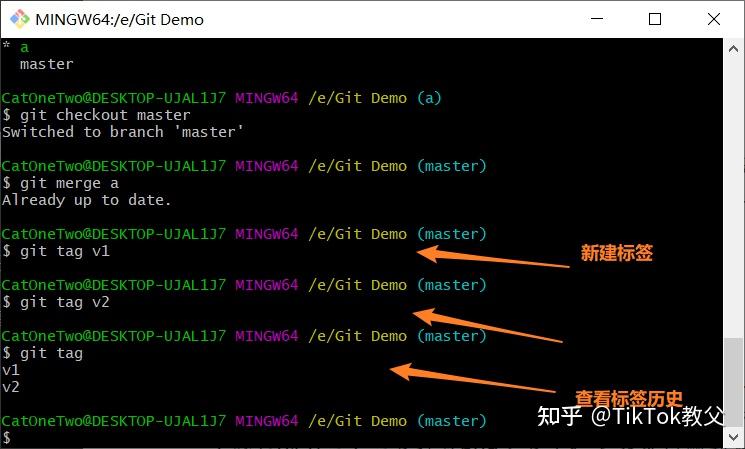


**删除分支**：有添加就有删除，假如分支新建错了，或者 a 分支的代码已经**顺利合并**到 master 分支来了，那么 a 分支没用了，用 **git branch -d** 删除。删除分支 a 的语句为 git branch -d a，这里不再演示。  
**强制删除分支**：上面说了 **git branch -d** 是删除分支的意思，有些时候可能会删除失败，比如如果 a 分支的代码还**没有合并**到master，你执行 git branch -d a 是删除不了的，它会智能的提示你a分支还有未合并的代码，但是如果你非要删除，那就执行 **git branch -D a** 就可以强制删除 a 分支。

**2.9 git tag**

这是今天学习的最后一个操作，tag 是标签的意思，写报告、写代码都有版本的迭代，比如 1.0，1.1 。git 可以给每个版本打上这样的标签，这样你忽然想看看第一版的报告是什么内容，就可以通过标签切换回去。

**git tag v1** 代表新建 v1 tag，你可以换成其他名字，**git tag** 代表查看标签。



如果想切换至某一标签，还是用刚刚学的**git checkout v1**即可，v1 是某版本标签名。

**3. 总结**

以上全是一些最基本的 git 操作，而且全是在本地环境进行操作的，完全没有涉及到远程仓库，最后总结一下基本 Git 操作：

* git status ：查看仓库状态
* git init：创建一个空仓库，或者重新初始化一个已有仓库
* git add：把文件添加到可提交列表（临时缓冲区）
* git commit：提交改动（增删改）至仓库
* git log：打印提交日志
* git branch：查看、添加、删除分支
* git checkout：切换分支、标签
* git merge：合并分支
* git tag：新建、查看标签

**四、本地与远程仓库的交互**

导读：上一篇文章介绍了 git 的安装及本地的几个基本操作，这篇文章介绍如何使用 git 这个工具，完成本地与远程仓库的交互。  
这篇文章将告诉你，如何使用 git  
下载仓库文件  
把本地的文件上传到远程仓库  
仓库文件有更新时保持本地与远程一致  
让我们开始吧！

补充：现在的主分支是 **main**，不是 **master**，下面的语句中要把 master 换成 main。

**1. SSH 配置**

要想本地与远程仓库交互，必须有一个安全机制防止数据泄漏，这个安全机制就是 SSH，因此远程交互之前需要进行 SSH 配置。

**1.1 生成 SSH key**

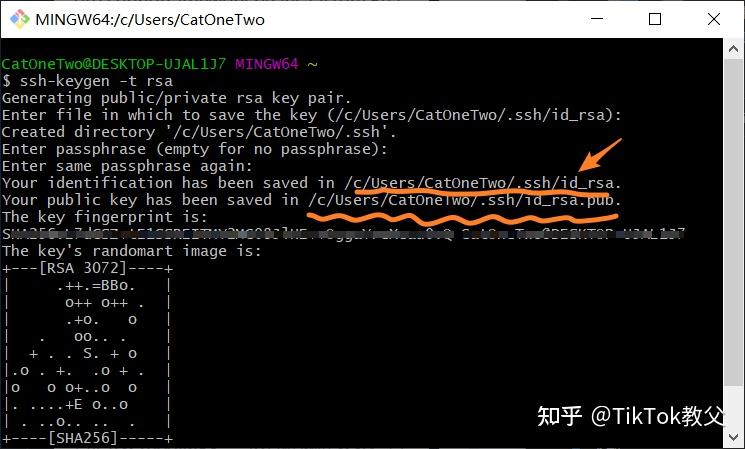
想要生成 SSH KEY 需要先安装 SSH，不过我们之前已经安装的 **GitBash** 自带 SSH（我的操作系统是 Windows 10）

打开 GitBash ，输入 ssh，如果返回如下界面说明 SSH 已安装，否则得自己安装呀！



紧接着输入 **ssh-keygen -t rsa** （指定 rsa 算法生成密钥），接着连续三个回  
车键（不需要输入密码），然后就会生成两个文件 id\_rsa （密钥）和 id\_rsa.pub （公钥）。

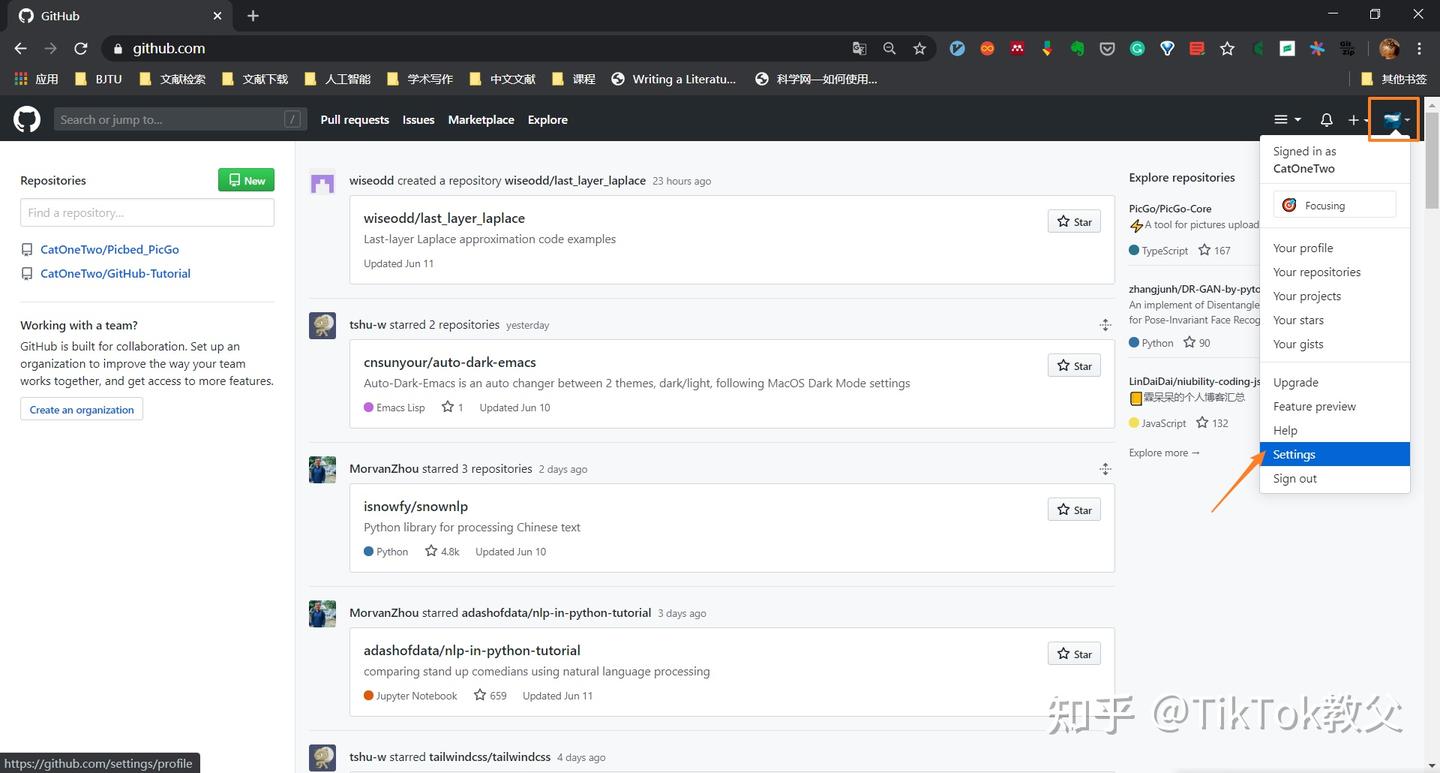
我划线的地方就是密钥和公钥存储的位置，等会儿我们需要打开文件复制密钥。



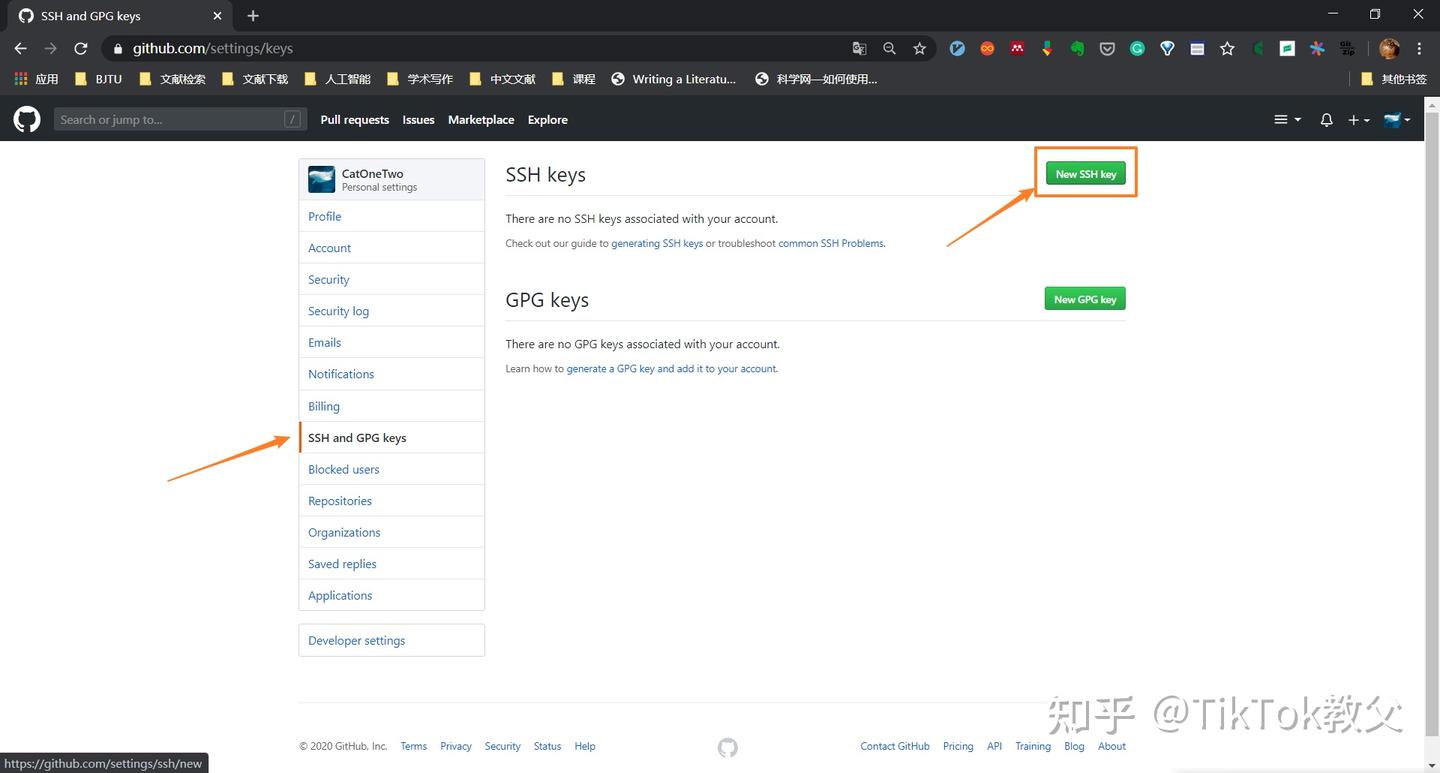
**1.2 添加 SSH key**

接下来要做的是把 id\_rsa.pub 的内容添加到 GitHub 上，这样你本地的 id\_rsa 密钥跟 GitHub 上的 id\_rsa.pub 公钥进行配对，授权成功才可以提交代码。

打开 GitHub，点击右上角的你的头像，点击设置 **settings**：

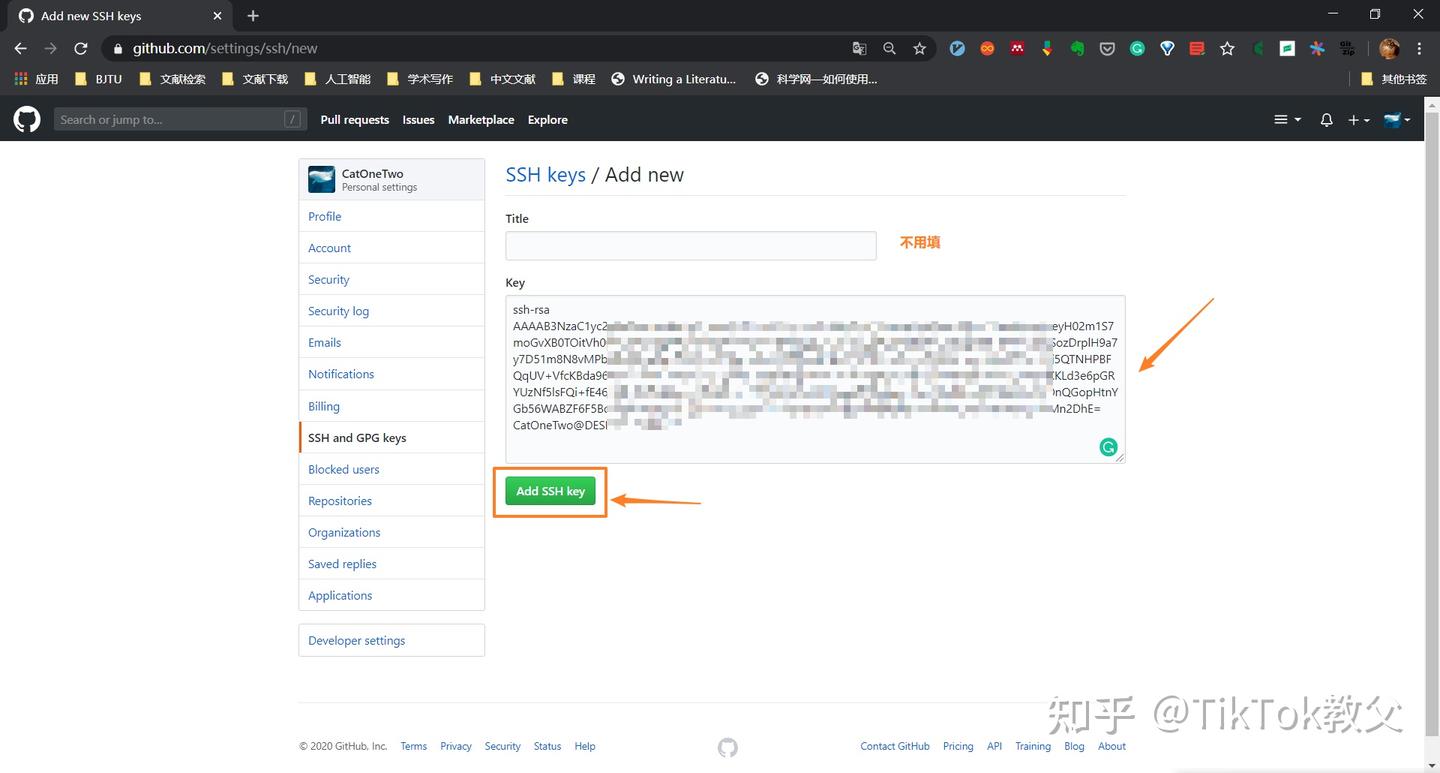


点击左侧的**SSH and GPG keys**，点击右上角的**New SSH key**：

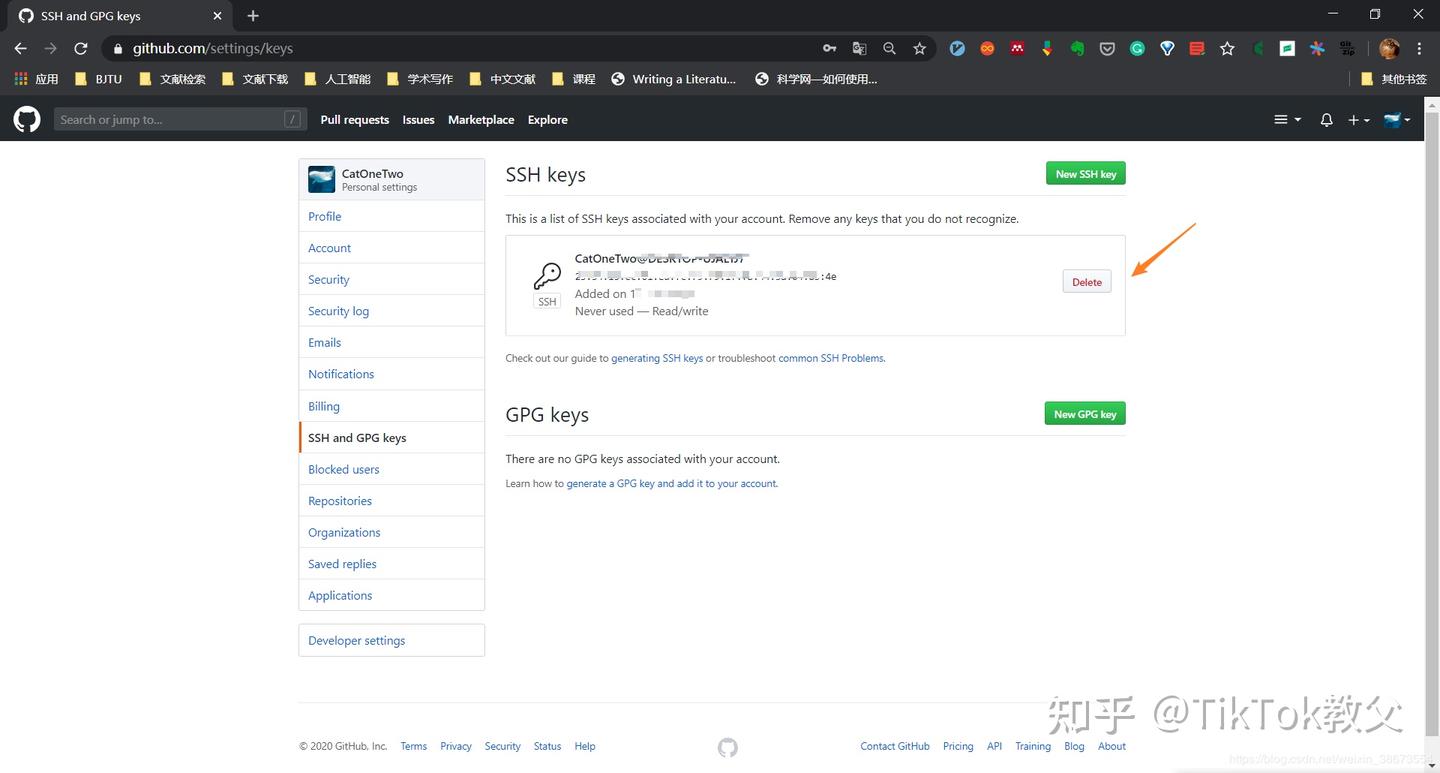


进入到下面的界面，把刚刚生成的公钥 （id\_rsa.pub 文件中的内容）复制到 Key 所在框， Title 不用填，复制好点击下方的 **Add SSH key** 按钮即可。

提示：id\_rsa.pub 可以选择用 Notepad++ 或 Sublime 打开~

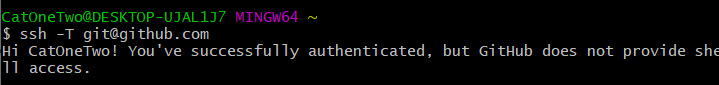


添加完返回 SSH 页面就会出现你的本地信息，如下图所示：



**1.3 验证绑定**

最后我们验证一下本地的 git 与 GitHub 是否绑定成功，在 GitBash 输入**ssh -Tgit@github.com**，如果返回下面的提示，说明绑定成功，不过有时候返回有些慢呀



到此为止，我们的 SSH 配置完成，接下来我们可以与远程仓库交互啦！

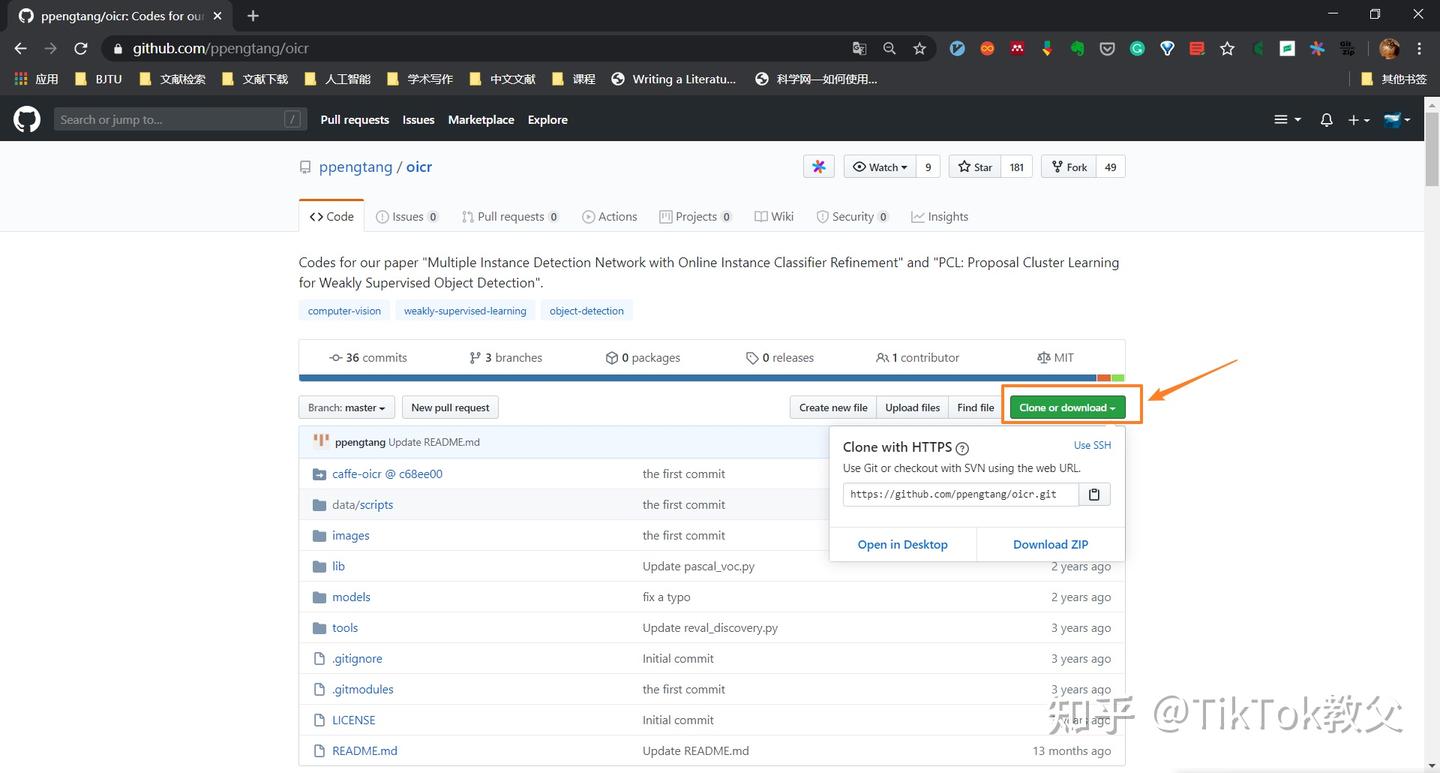
**2. 通过 Git 下载代码**

看过第一篇教程的同学一定知道，某个项目的界面的 **Clone or download** 按钮可以下载整个仓库/项目的文件。

最简单的就是点击 **Download ZIP**，这是常规的下载操作，下载一个压缩包到指定目录。

另一种方法就是使用 git clone，这种方法又分为 cloning with HTTPS 和 clone with SSH，下面我们分别演示一下。

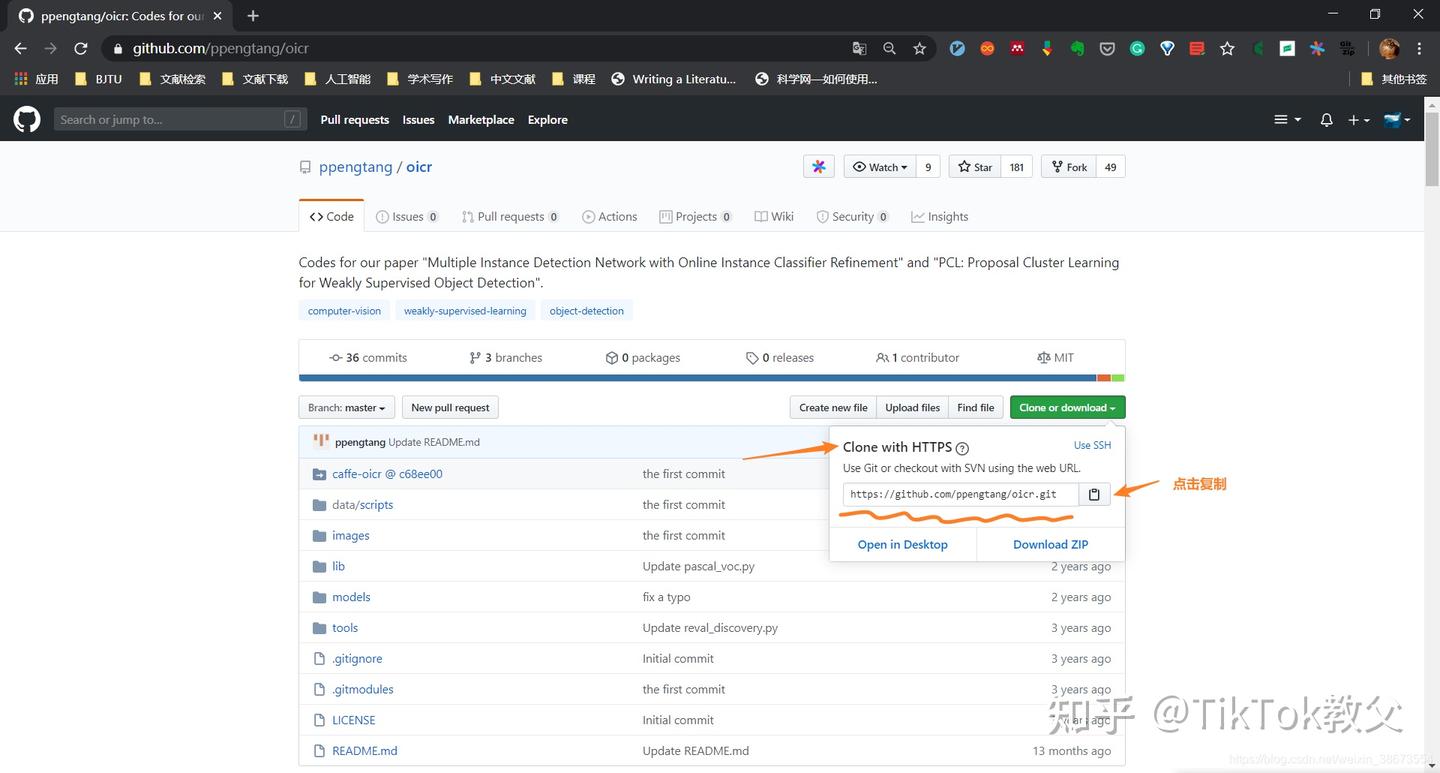
到这里你会有疑问，我能直接点击下载，为啥要学 git clone。  
还记得第一篇教程就说过，git 是个版本管理工具，你用它就是为了记录提交历史，写错了能够回滚到历史文件等。  
如果你下载的是你自己的文件，你直接 download，就完成不了上面的功能，你改了文件也没法在原来的基础上提交。  
当然如果你只是为了下载别人的仓库看看，不需要版本控制这些功能，直接download 就行；但如果需要用到版本管理，就得学习 git clone



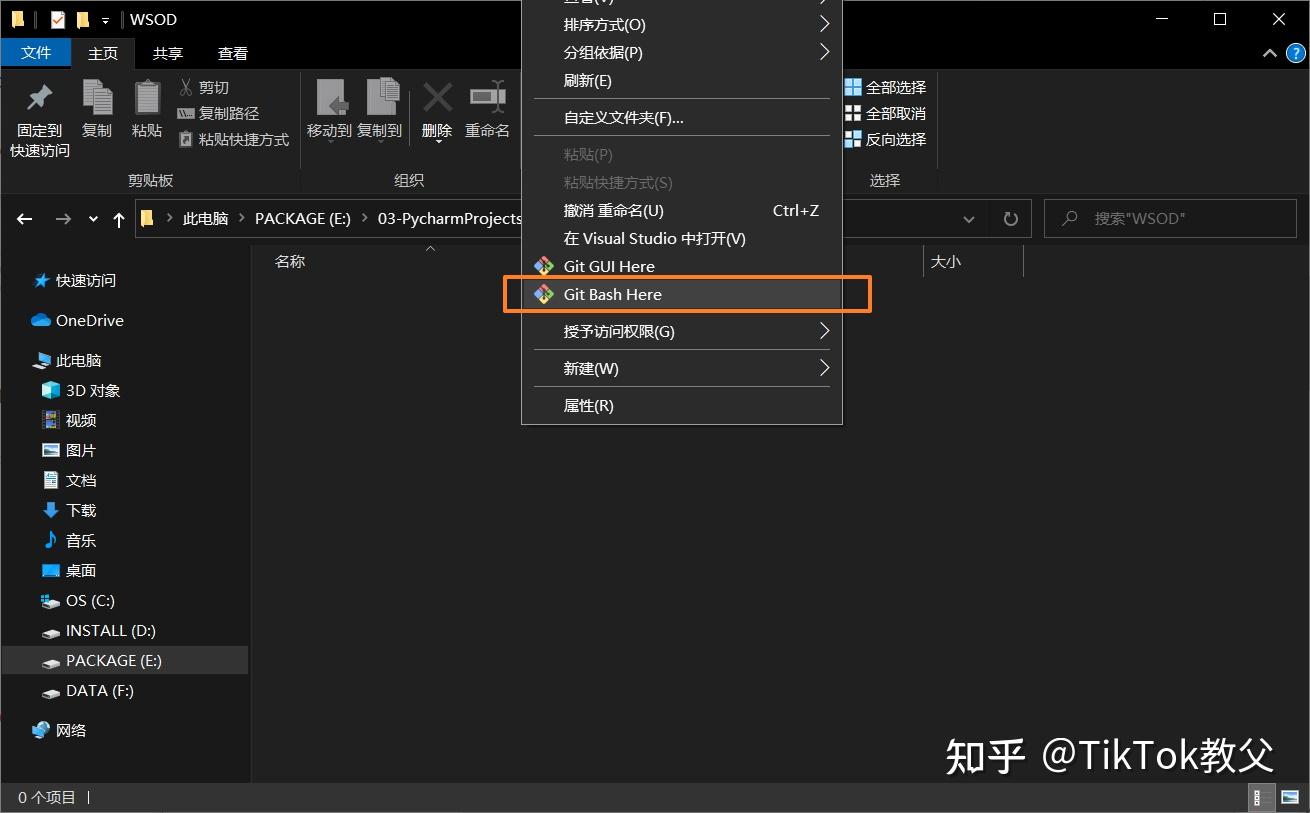
**2.1 git clone with HTTPS**

这种方法是官方推荐的，它不用 SSH 就可以完成。

我以一个仓库为例，注意上面写的是 Clone with HTTPS，复制下面的 URL：



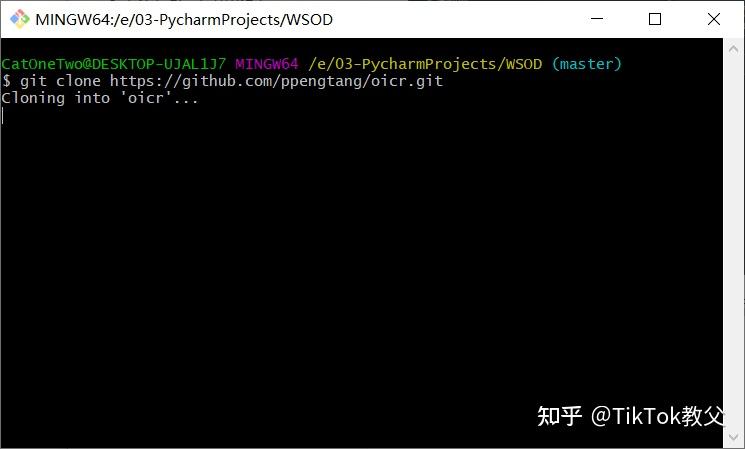
打开文件要 clone 的目录，右击选择Git BashHere:



打开 GitBash 窗口，输入 **git clone**，后面跟刚刚复制的 URL，回车，等它下载完，你会看到你的文件夹里直接就是仓库里的文件。

语句长这样：

git clone https://github.com/project/repo.git

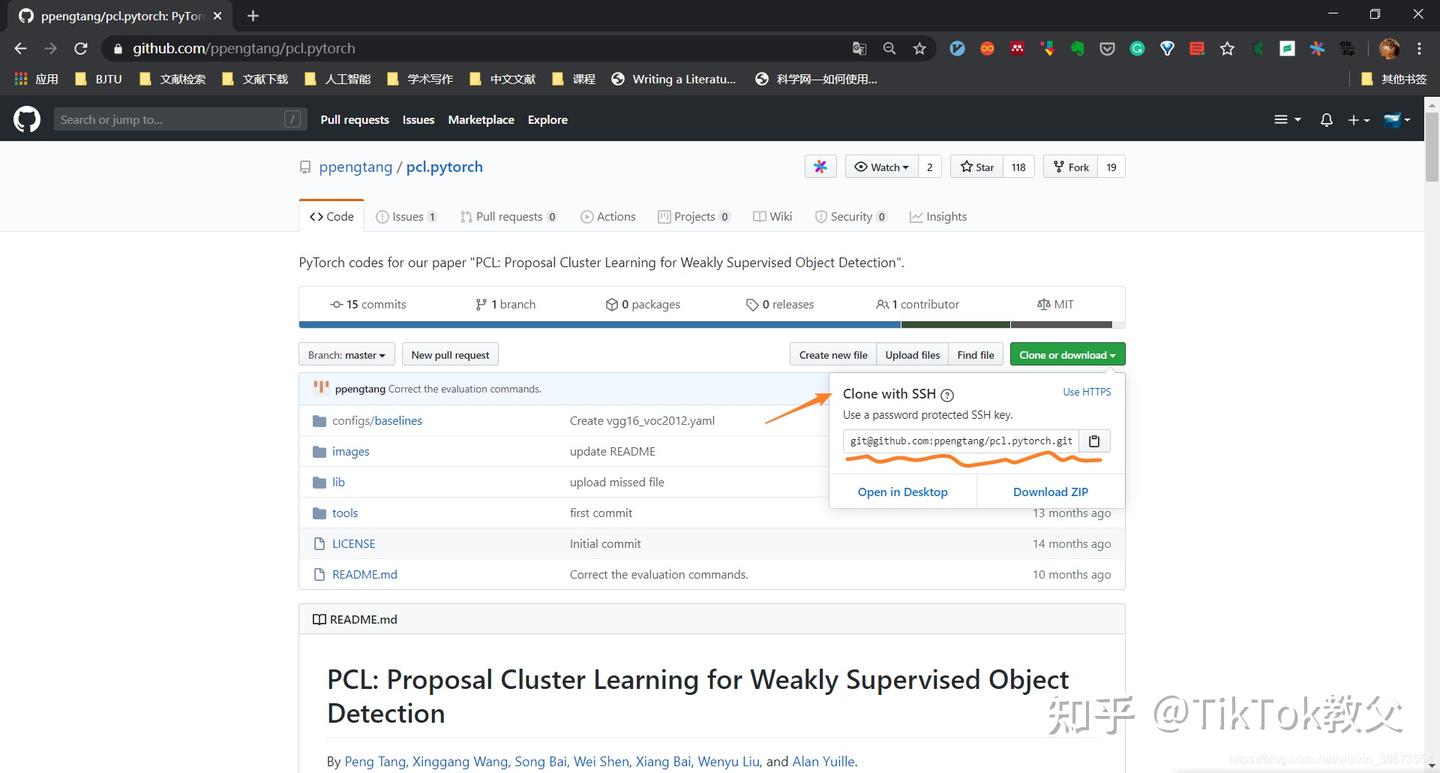


**2.2 git clone with SSH**

在 Clone with HTTPS那里，点击 Use SSH ，就会切换成第二种 git clone 方式。

第二种方式需要用到 SSH，没有配置是不能用的，所以我们放到这一节介绍。

其实步骤与上面的步骤一样，只不过复制的 URL 是 SSH 机制下的：



接着也是在目录下打开 Git Bash，输入 **git clone**， 后面是刚刚复制的 URL。

语句长这样，可以看出 SSH 下的 URL 和 HTTPs 下的 URL 不一样。

git clone git@github.com:project/repo.git

**2.3 异同**

再说这两种方法的异同。

同：HTTPs 与 SSH 下的 git 都可以直接进行 git clone 操作

异：

* HTTPs git clone 到本地，进行了一些文件的修改，当再次提交到 GitHub远程服务器的时候，都会进行账号与密码的输入
* SSH git clone 到本地之后，由于已有 SSH Keys 授权，就不需要用户名和密码进行授权了。

**3. 通过 Git 提交代码**

**3.1 两个命令**

上一节我们讲了一些基本的 git 命令，不过都只涉及本地，现在我们想本地与远程仓库交互，就需要学习两个新命令：

* git push：翻译为推，当你的代码更新，需要把本地的推到远程仓库
* git pull：翻译为拉，当远程仓库有更新，你需要把远程的拉到本地进行合并

可以看出这两个命令都是为了保证本地与远程的同步。

我们需要区分 git clone 和 git pull，虽然它们都是从远程仓库到本地的更新，但前者在本地无仓库时使用，后者是本地已有仓库时使用。

使用举例：

git push origin master # 把本地代码推到远程 master 分支

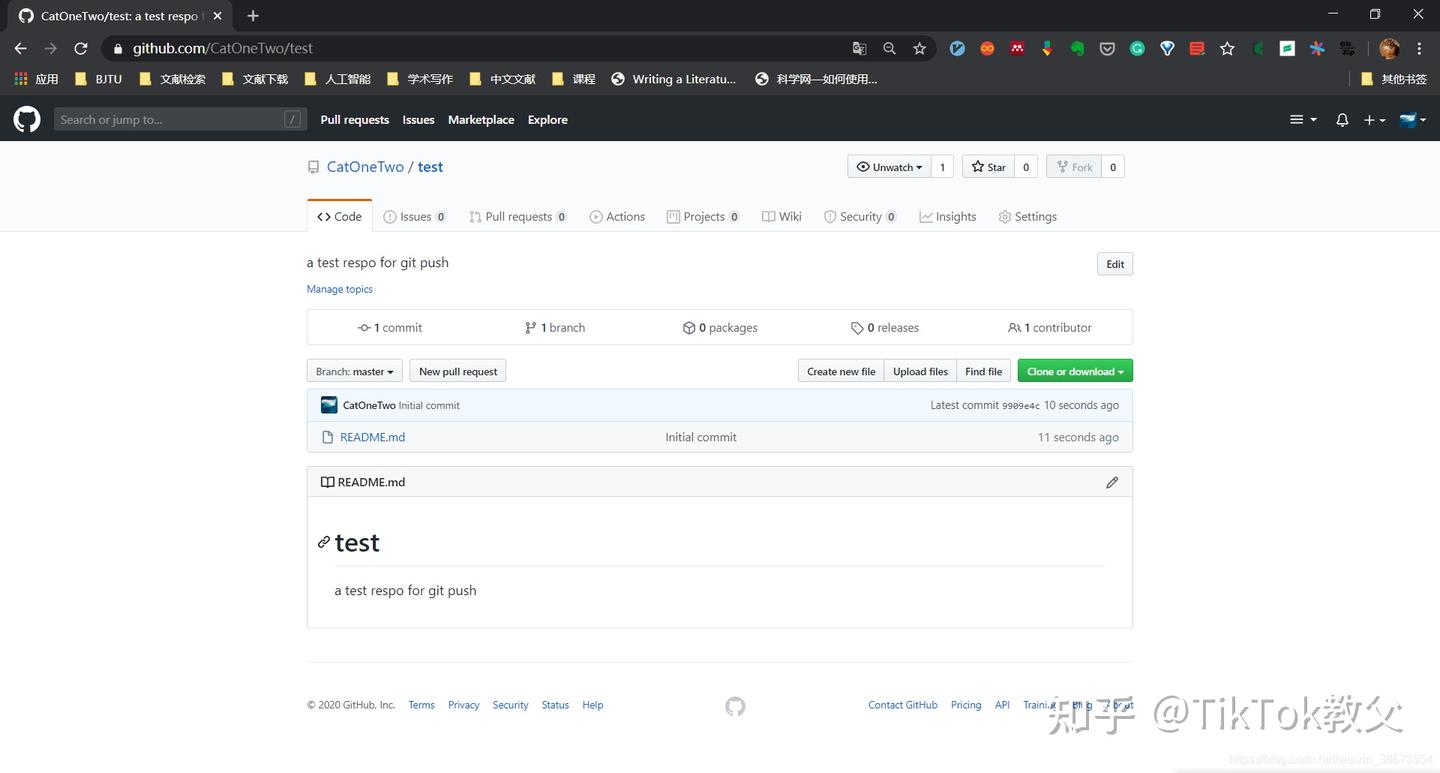
git pull origin master # 把远程最新的代码更新到本地

**一般我们在 push 之前都会先 pull ，这样不容易冲突。**

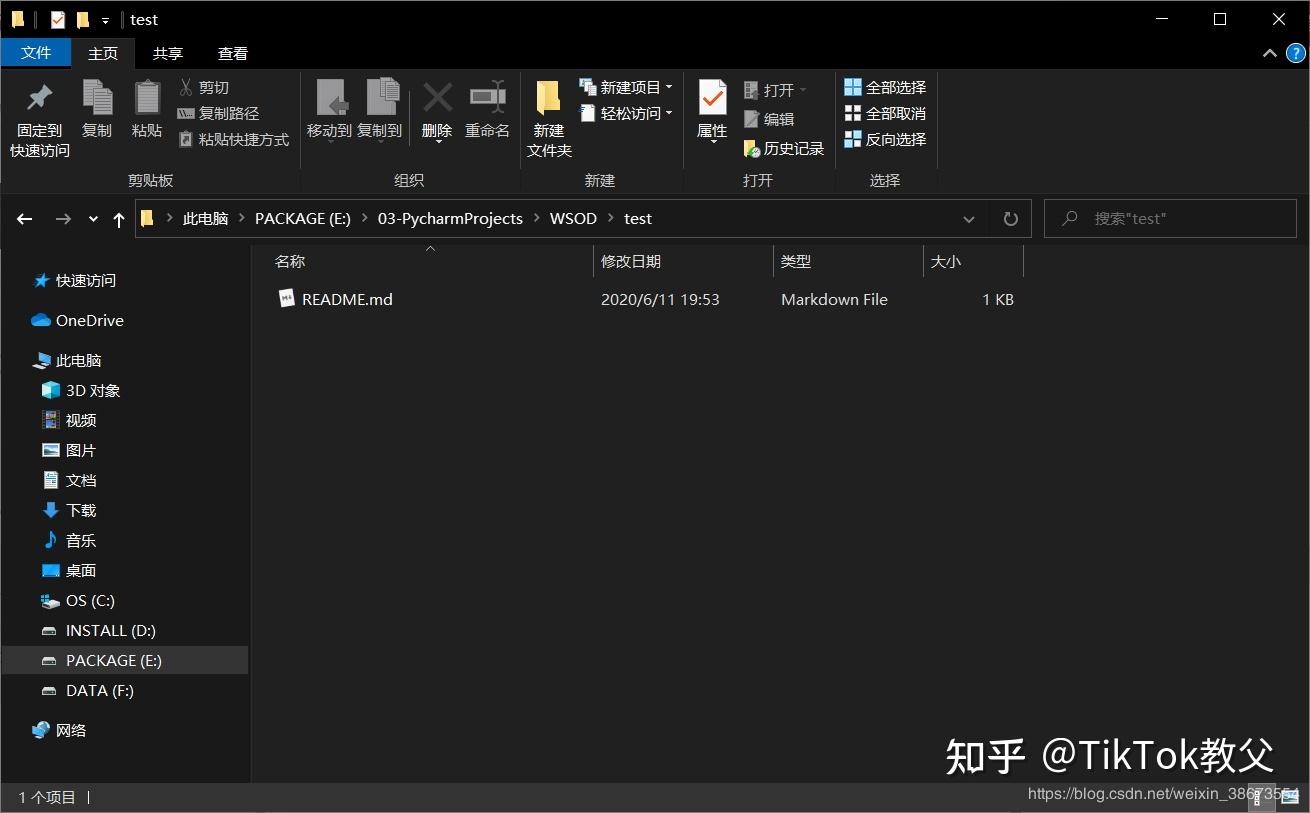
**3.2 提交代码**

通过 git 提交代码的前提是**已有仓库**，然后把仓库 clone 到本地，你修改后再pull。

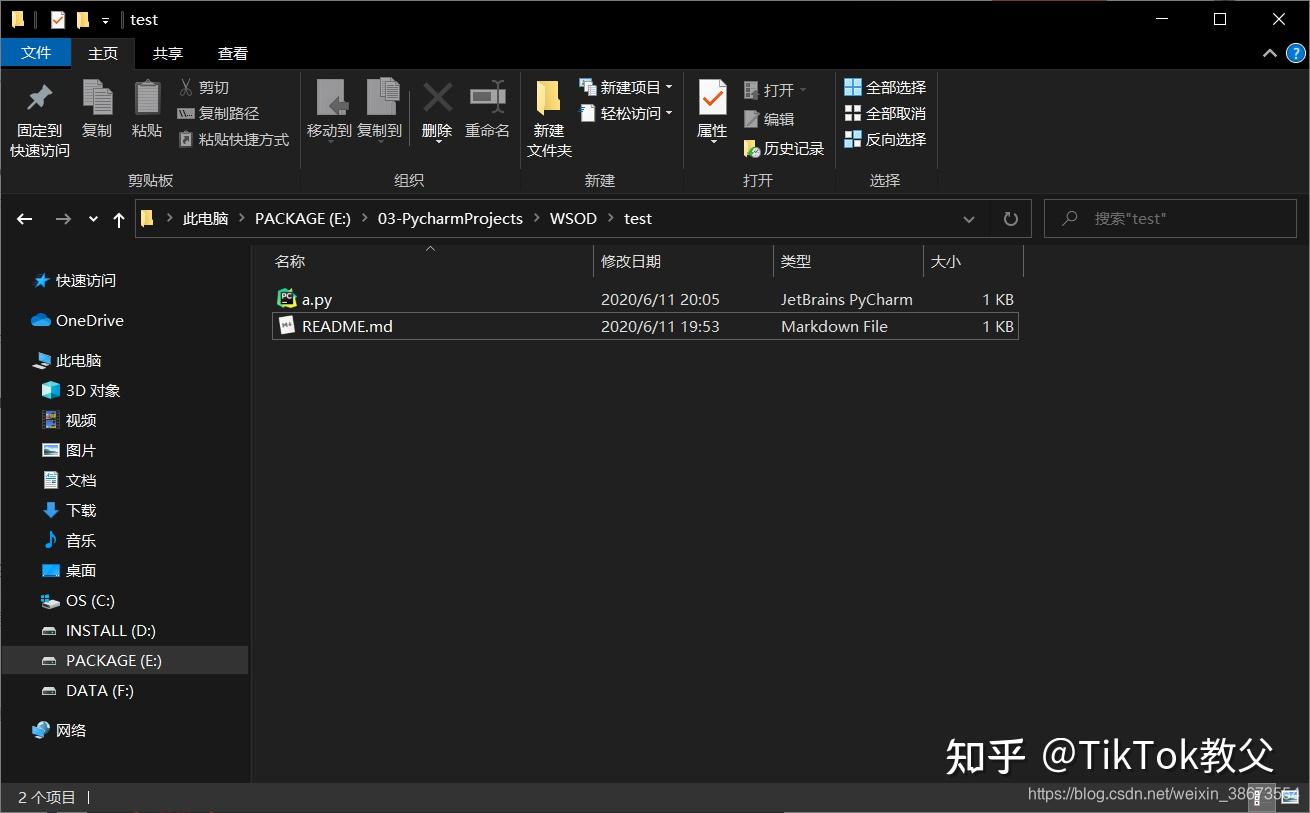
相信通过前面的教程，你已经学会如何新建和删除一个仓库，我这里用一个临时建的仓库演示一下提交功能。



然后我们把仓库 clone 到本地，为了不输入账号密码，我这里用 SSH 的方法 clone。



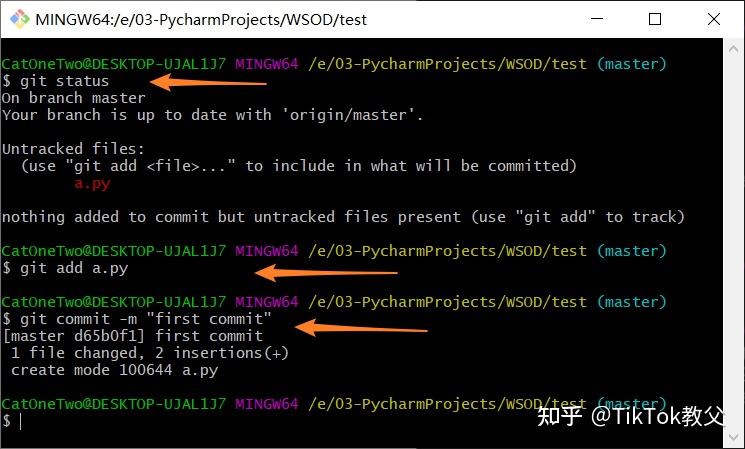
接下来我就要在更新本地仓库， 我只是新加一个 python 文件，你也可以修改一个已有文件。



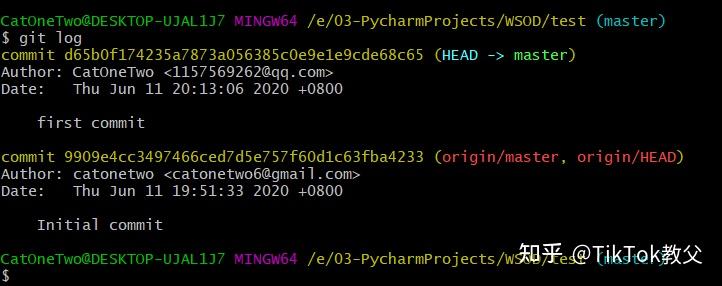
看过上一篇 git 教程，我们就知道所有的改动都需要通过 git add和git commit 提交到本地的仓库。

**在这个文件夹/本地仓库**，右击打开 **Git Bash**，输入 git status 查看状态，再用 git add和git commit 提交新加的文件。

注意 first commit 是我为这次 commit 起的名字，你可以写成其他名字。



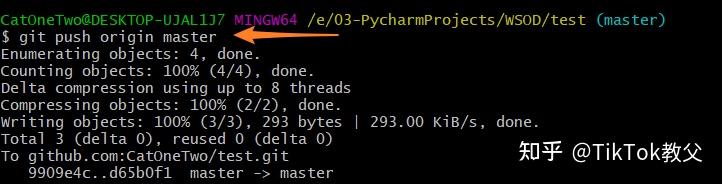
然后输入git log查看提交历史，可以看到一共有两次 commit，第一次是在 GitHub 上新建仓库，第二次是刚刚在本地新提交文件。



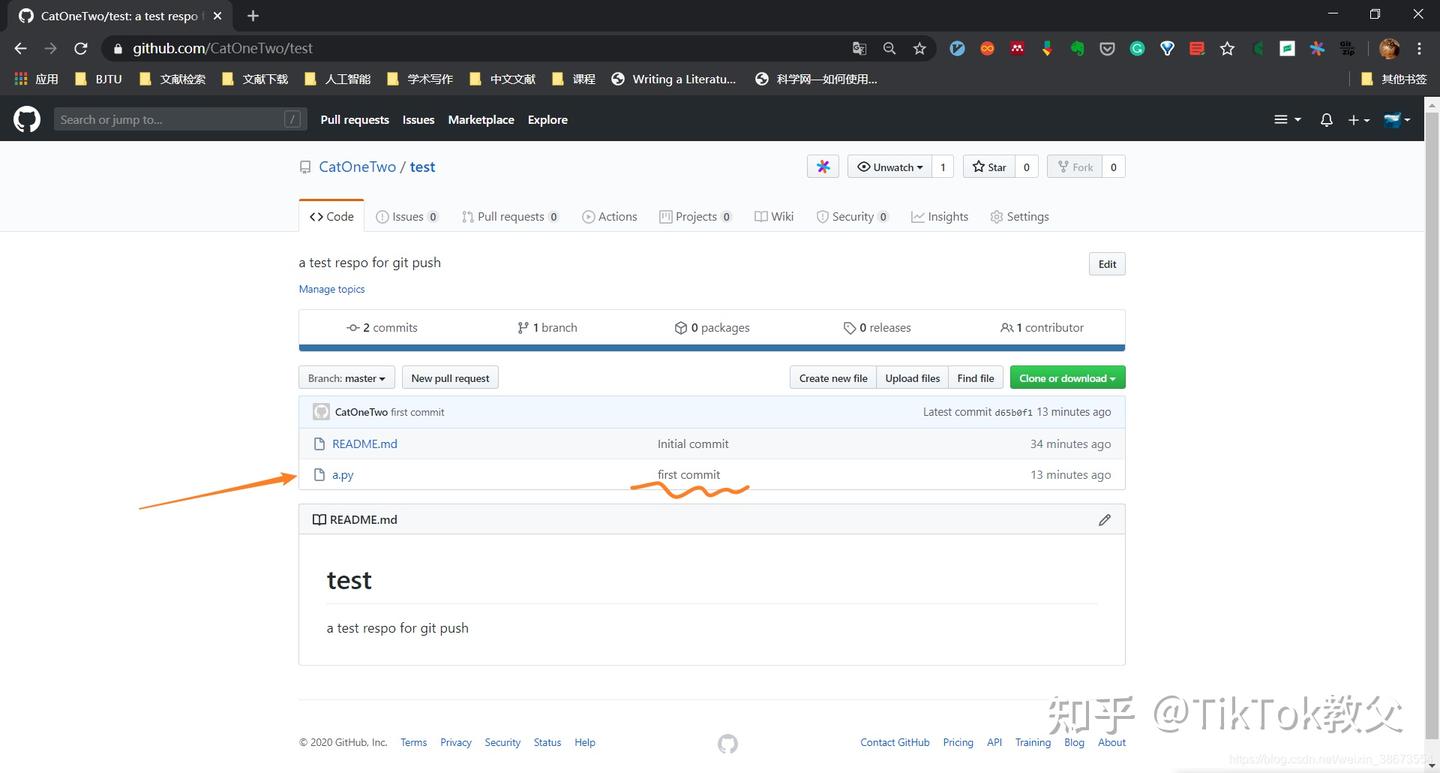
**到现在我们才完成本地更新的工作，接下来要把本地仓库 push 到远程仓库**

在 **Git Bash** 输入 git push origin master 即可

默认向 GitHub 上的 test 目录提交了代码，而这个代码是在 master 分支。



然后我们刷新远程仓库，可以看到有更新：



总结：这篇文章介绍了 SSH 的配置，通过 SSH 完成 git clone 和 git pull 等操作，完成本地仓库与远程仓库的交互，并记录每次的提交历史。

**五、Git 进阶**

之前的文章已经学习过 git 的基础用法，这篇文章再介绍几个进阶用法

**1. Git 基础**

首先我们回顾一下 Git 基础用法。

**1.1 基本配置**

* git --version：返回版本号
* git --help：帮助文档
* git config --list：查看所有配置
* git config --list --show-origin：查看所有配置以及它们所在的文件
* git config --global user.name "用户名"：全局配置用户名
* git config --global user.email 邮箱地址：全局配置邮箱
* git config user.name：查看用户名
* git config user.email：查看邮箱

我们在 GitHub 的每次提交理论上都会在 主页的下面产生一条绿色小方块的记录，如果你确认你提交了，但是没有绿色方块显示，那肯定是你提交代码配置的邮箱跟你 GitHub 上的邮箱不一致，GitHub 上的邮箱可以到 Setting -> Emails里查看。

**1.2 基本操作**

* git status ：查看仓库状态
* git init：创建一个空仓库，或者重新初始化一个已有仓库
* git add：把文件添加到可提交列表（临时缓冲区）
* git commit：提交改动（增删改）至仓库
* git log：打印提交日志
* git branch：查看、添加、删除分支
* git checkout：切换分支、标签
* git merge：合并分支
* git tag：新建、查看标签
* git clone：下载仓库

**2. Git 进阶**

下面介绍几个有用的进阶知识。

**2.1 设置别名**

上面提到的基本操作，有几个使用很频繁，每次都要输入完整的单词。

使用 `alias`` 可以给这些操作七个简单的别名，就能简化输入。

基本语法是：

git config --global alias.别名 git命令

比如给 commit 和 status 起别名：

git config --global alias.ci commit

git config --global alias.st status

下次再用这两个命令就可以写：

git ci

git st

除了简单命令还可以设置组合命令，给组合命令加引号即可：

git config --global alias.pullom 'pull origin master'

git config --global alias.pushom 'push origin master'



当然别名不是固定的，你可以根据习惯设定。

起别名后，你可以看到 .gitconfig 文件里有 alias 的配置，如果不需要某个别名，删掉即可。

此外，有一个厉害的 git log 命令，可以把常规的日志按结构化显示。

git log --graph --pretty=format:'%Cred%h%Creset - %C(yellow)%d%Creset %s %Cgreen(%cr)

%C(bold blue)<%an>%Creset' --abbrev-commit --date=relative

你可以给它起名为lg，以后打印结构化日志直接输入git lg就行：

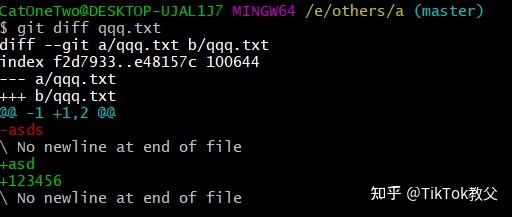
git config --global alias.lg "git log --graph --pretty=format:'%Cred%h%Creset - %C(yellow)%d%Creset %s %Cgreen(%cr)

%C(bold blue)<%an>%Creset' --abbrev-commit --date=relative"

**2.2 显示改动**

还记得 git add 会把文件放入暂存区吧，**在git add** **之前**使用 git diff 可以显示你对某一文件的改动。

红色的部分前面有个 - 代表我删除的，绿色的部分前面有个 + 代表我增加的，所以从这里你们很一目了然的知道我到底对这个文件做了哪些改动。



除此之外，git diff还有其他用法：

git diff <$id1> <$id2> # 比较两次提交之间的差异

git diff <branch1>..<branch2> # 在两个分支之间比较

git diff --staged # 比较暂存区和版本库差异

**2.3 版本回滚**

在我们使用 Git 的时候，有时候会遇到想要回滚到某次提交之前的场景。

首先用 git log 查看版本号（commit 后面那一长串就是版本号）：



找到想要回退的版本号之后，在本地 Git 仓库执行如下命令，选一个即可：

git reset --hard 版本号 (抛弃当前工作区的修改)

git reset --soft 版本号 (回退到之前的版本，但保留当前工作区的修改，可以重新提交)

如果你的文件只在本地，执行上面的步骤就可以啦。

如果你的文件提交到 GitHub 上，还需要执行如下命令同步到远端：

git push origin 分支名

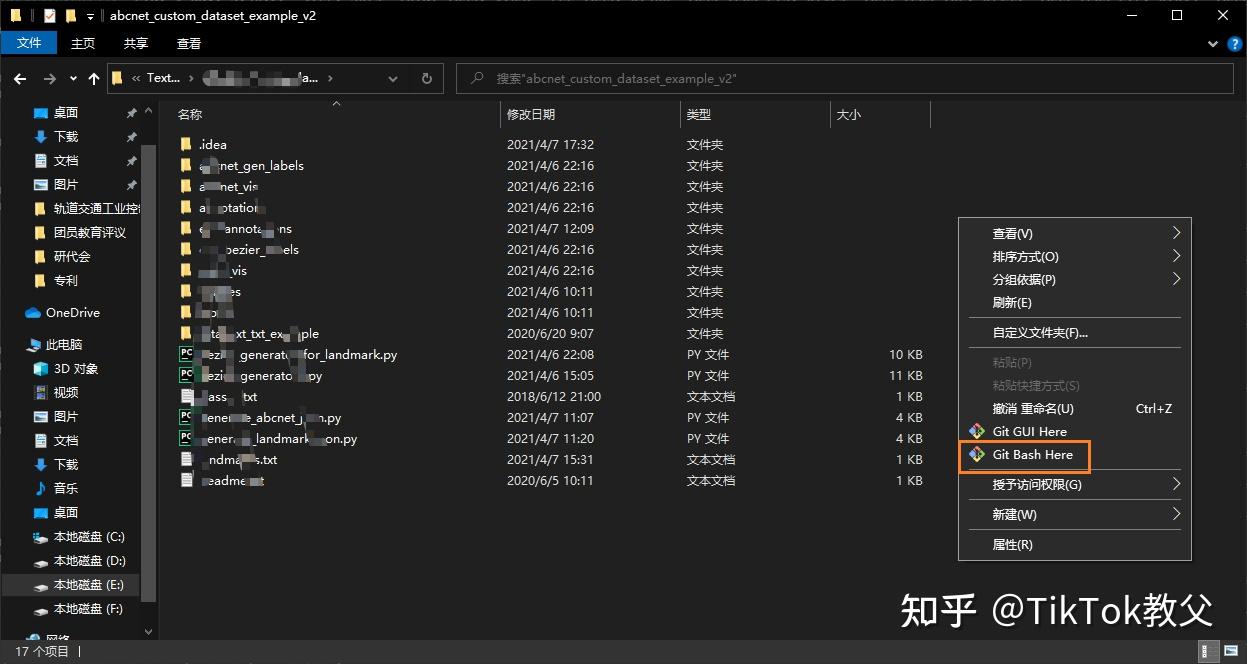
示本地的版本落后于远端的版本，因此我们还需要在上述命令中加上--force参数：

git push origin 分支名 --force

**六、GitHub 提交文件夹快速版**

导读：**四、本地与远程仓库的交互**这篇介绍比较详细，如果你现在已经新建了仓库（最好设成私有的），本地有个文件夹，如何快速上传文件夹至仓库。

首先进入这个文件夹，右击点击 Git Bash Here。



依次输入下面的语句：

git init // 初始化版本库

git add . // 添加文件到版本库（只是添加到缓存区），.代表添加文件夹下所有文件

git commit -m "first commit" // 把添加的文件提交到版本库，并填写提交备注

git remote add origin 你的远程库地址 // 把本地库与远程库关联

git push -u origin main // 第一次推送时

git push origin main // 第一次推送后，直接使用该命令即可推送修改

然后更新你的仓库，文件夹上传成功！

编辑于 2024-12-22 20:50・IP 属地河南