摘自知乎文章：<https://www.zhihu.com/question/20070065/answer/117017972>

作者：杨晓辉  
链接：https://www.zhihu.com/question/20070065/answer/117017972  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

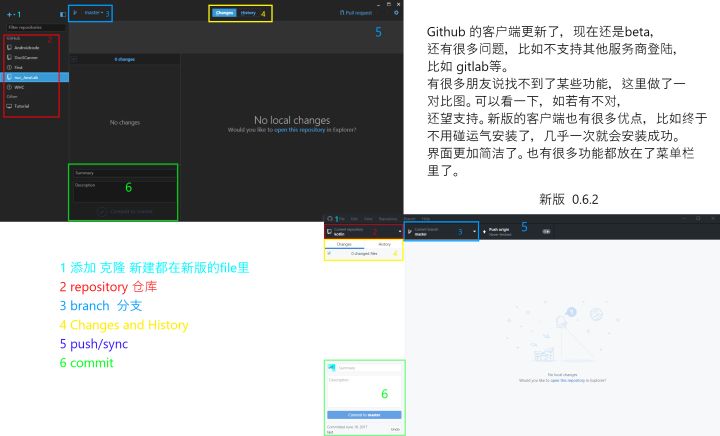
如果你还不知道什么是git，只知道github，但是还不会用，我想这个教程会帮助你。

**前言**

鉴于网上目前的教材都太落后，github for windows已经更新了多个版本，好多界面都发生了变化，所以来写这个教程。目的是为了帮助和我一样初学github，但是苦于找不到教程的同学，为了写最详细的教程。配备了大量的图文介绍。该教程是基于**GitHub for windows (3.0.17.0)**

**注：**

由于教程为 3.0.17.0 ，之后github对客户端进行了新版的更新，这里的图为新版与旧版对比。希望可以给大家带来帮助。

<img src="https://pic3.zhimg.com/50/v2-38f98b1a48dd602a3f19ea0638e348f9\_hd.jpg" data-rawwidth="2778" data-rawheight="1681" class="origin\_image zh-lightbox-thumb" width="2778" data-original="https://pic3.zhimg.com/v2-38f98b1a48dd602a3f19ea0638e348f9\_r.jpg">

**什么是Github**

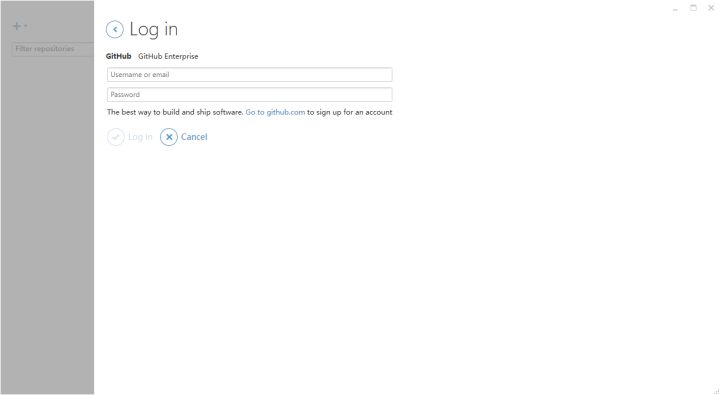
说到什么事github，我们先看wikipedia的描述“[GitHub是一个利用Git进行版本控制、专门用于存放软件代码与内容的共享虚拟主机服务。它由GitHub公司（曾称Logical Awesome）的开发者Chris Wanstrath、PJ Hyett和Tom Preston-Werner使用Ruby on Rails编写而成。](https://link.zhihu.com/?target=https%3A//zh.wikipedia.org/wiki/GitHub)”

准备工作

1. 下载[github for windows](https://link.zhihu.com/?target=https%3A//desktop.github.com/)，安装这里不赘述。
2. [注册github账号](https://link.zhihu.com/?target=https%3A//github.com/)

<img src="https://pic4.zhimg.com/50/cbb9a0d32513fb0c843f35c92978cc77\_hd.jpg" data-rawwidth="1904" data-rawheight="956" class="origin\_image zh-lightbox-thumb" width="1904" data-original="https://pic4.zhimg.com/cbb9a0d32513fb0c843f35c92978cc77\_r.jpg">

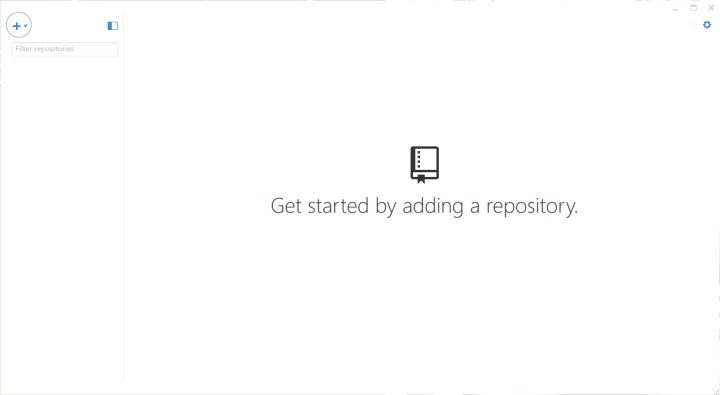
1. 登陆到github for windows。

<img src="https://pic2.zhimg.com/50/b69b539f7a331713c8a3f28e79803648\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic2.zhimg.com/b69b539f7a331713c8a3f28e79803648\_r.jpg">

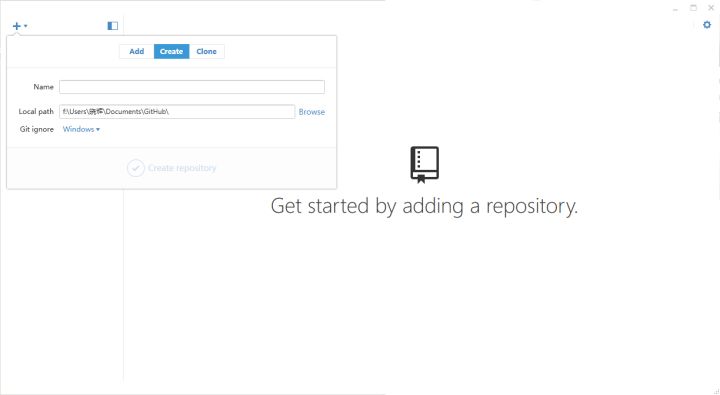
准备工作都完了，我们开始正式学习。^\_^

**创建第一个代码库**

**认识界面**

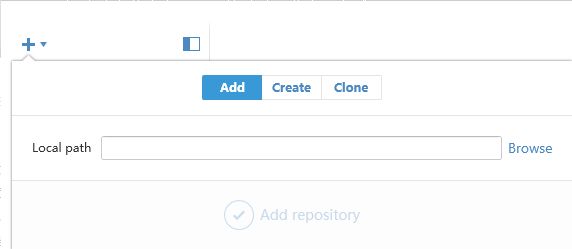
<img src="https://pic1.zhimg.com/50/dabdb03cc2c79ad5fa1629b18d0bc6c4\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic1.zhimg.com/dabdb03cc2c79ad5fa1629b18d0bc6c4\_r.jpg">

github for windows的界面非常清爽，的确符合geek的性质，个人表示非常喜欢。 我们来建立第一个仓库，点击左上角的**+号**，初次建立他会有一圈圈的涟漪，非常漂亮哦。 打开之后有三个选项，**Add**，**Create**，**Clone**。 我们分别来介绍一下这三个功能。

<img src="https://pic4.zhimg.com/50/9ada359dcf0b5eb355dfb4cf4de6cb84\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic4.zhimg.com/9ada359dcf0b5eb355dfb4cf4de6cb84\_r.jpg">

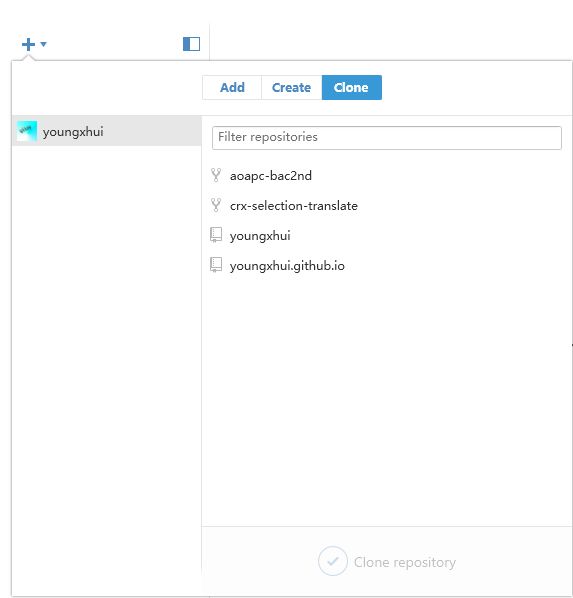
**Add功能**

如果本地有工程，就可以使用Add添加

<img src="https://pic1.zhimg.com/50/28af723c9462ab2173dcb1ac3092985c\_hd.jpg" data-rawwidth="572" data-rawheight="249" class="origin\_image zh-lightbox-thumb" width="572" data-original="https://pic1.zhimg.com/28af723c9462ab2173dcb1ac3092985c\_r.jpg">

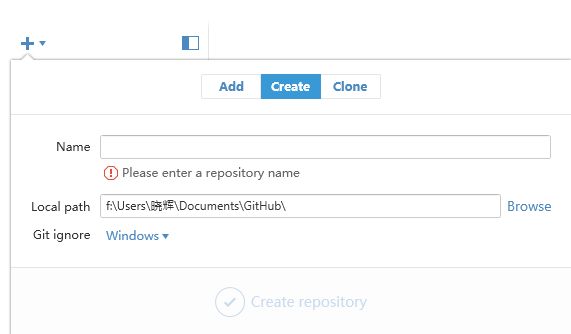
**Clone功能**

这个功能其实最好理解了，克隆这名字通俗易懂好理解。 如何使用Clone功能呢？ 就是将在浏览器上已经创建好的项目导入到本地，换句话说就是下载到本地。

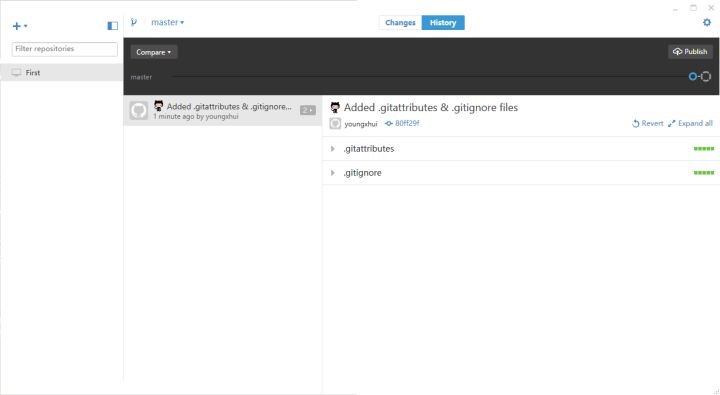
<img src="https://pic3.zhimg.com/50/a653022f1f8a6aa46acc72e2c5d0649e\_hd.jpg" data-rawwidth="573" data-rawheight="598" class="origin\_image zh-lightbox-thumb" width="573" data-original="https://pic3.zhimg.com/a653022f1f8a6aa46acc72e2c5d0649e\_r.jpg">

**Create功能**

创建一个代码库， Name填写你的仓库名字。Local path写你将要保存在本地路径。

<img src="https://pic4.zhimg.com/50/9357be0cae8004ec7718464b7f6aad90\_hd.jpg" data-rawwidth="571" data-rawheight="334" class="origin\_image zh-lightbox-thumb" width="571" data-original="https://pic4.zhimg.com/9357be0cae8004ec7718464b7f6aad90\_r.jpg">

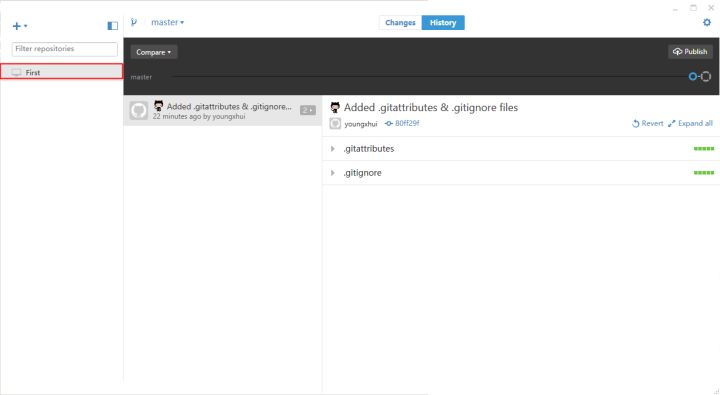
我们主要从这个功能开始github之旅。 我们在这里填写First，来创建第一个我们自己的repoeitory。

<img src="https://pic2.zhimg.com/50/1165fd380931e0e795ffecb609702305\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic2.zhimg.com/1165fd380931e0e795ffecb609702305\_r.jpg">

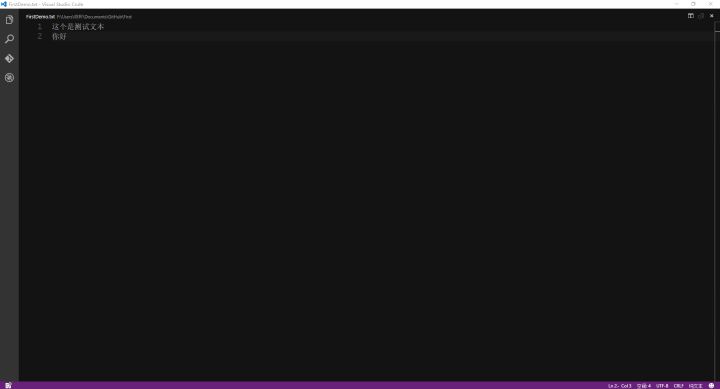
**开始使用第一个代码库**

**修改第一个代码库中内容**

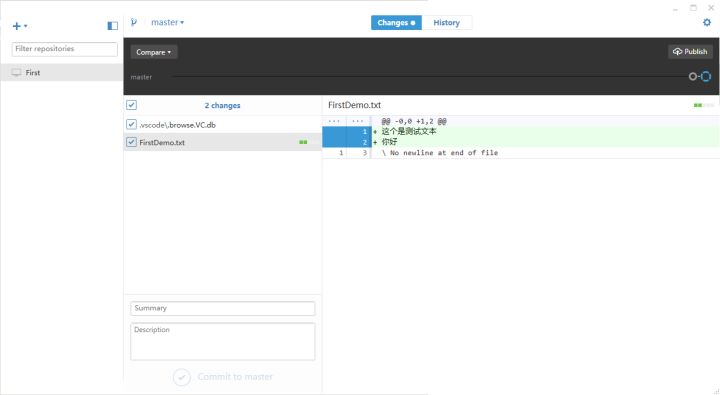
我们来找到刚刚创建的代码库在本地的位置。就是刚刚在local path的地址路径，当然如果你忘了，请右键点击First。

<img src="https://pic1.zhimg.com/50/ece15b960e76cd4dde9227136156ef02\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic1.zhimg.com/ece15b960e76cd4dde9227136156ef02\_r.jpg">

选择Open in Explorer。这样我们就可以转到刚刚的路径下。 我们新建一个文本文档。在里面编辑。 如下

<img src="https://pic4.zhimg.com/50/42c0bbcbce9841870311a76d12bfd792\_hd.jpg" data-rawwidth="1920" data-rawheight="1038" class="origin\_image zh-lightbox-thumb" width="1920" data-original="https://pic4.zhimg.com/42c0bbcbce9841870311a76d12bfd792\_r.jpg">

此时的github就会变成这个样子(Changs)：

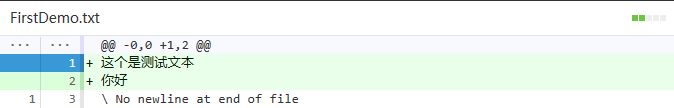
<img src="https://pic3.zhimg.com/50/61be45812f8f72a1976e598659568f6c\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic3.zhimg.com/61be45812f8f72a1976e598659568f6c\_r.jpg">

你会发现此时github会出现刚刚编辑的内容。

1. 这个是测试文本
2. 你好

并且前面会有**蓝色标识**，那么这个**蓝色标识**是什么用呢？

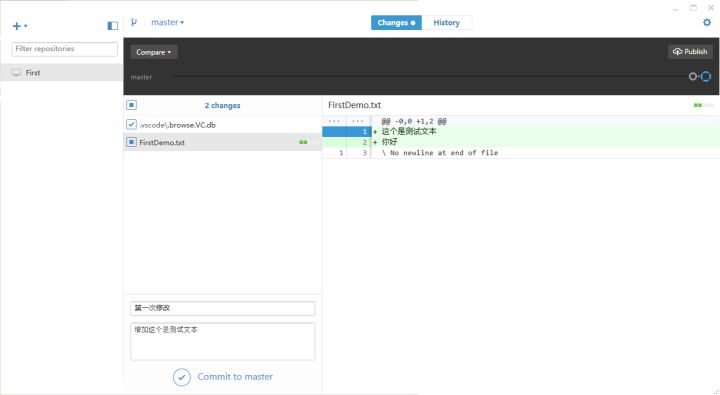
其实这个蓝色标识是提示你会上改变的文本。比如我第一次只想改变 **这个是测试文本**并不想把**你好**上传。 这时我们点击一下**你好**的前面的**蓝色标识**。

<img src="https://pic3.zhimg.com/50/afc0da3743ac94b06a610b77e2a92c51\_hd.jpg" data-rawwidth="674" data-rawheight="108" class="origin\_image zh-lightbox-thumb" width="674" data-original="https://pic3.zhimg.com/afc0da3743ac94b06a610b77e2a92c51\_r.jpg">

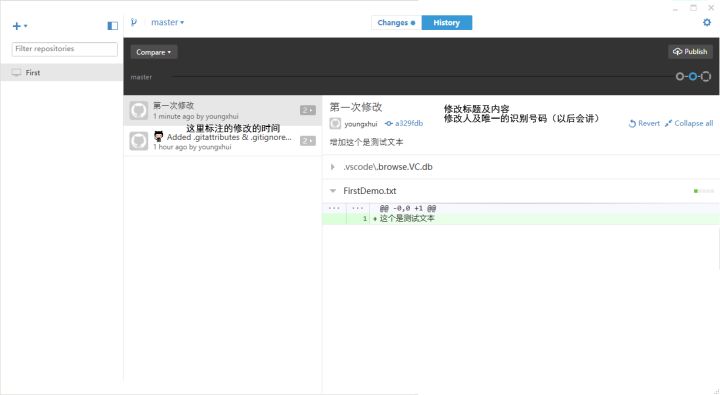
你会发现**你好**前面的蓝色标识没有了。

我们填写好**Summer**和**Description** Summer就是这次改动的总结，我们也可以理解为标题\*（必填）*，而Description可以理解为详细概况*（选填）\*

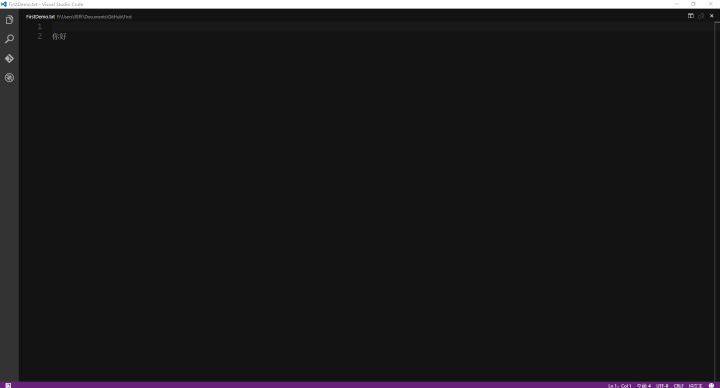
我们这里只选择第一个修改对象，也就是**这个是测试文本**就行修改。summer我们填写为*第一次修改*，Description我们填写为*增加了这个是测试文本*的内容，之后点击**Commit to master**。

<img src="https://pic4.zhimg.com/50/acce3c7a64257cc07be8476369bd5f6c\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic4.zhimg.com/acce3c7a64257cc07be8476369bd5f6c\_r.jpg">

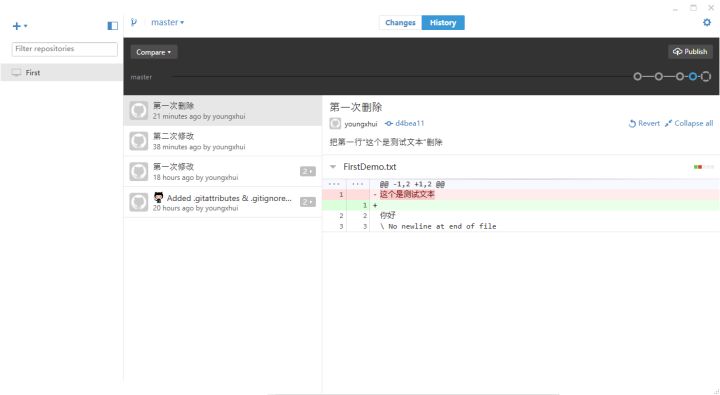
切换到**History**目录下 我们会发现他改变了。 这次我们把**你好**进行添加。

<img src="https://pic2.zhimg.com/50/6d0e86ae4e881fef4bf47fafa0a292cd\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic2.zhimg.com/6d0e86ae4e881fef4bf47fafa0a292cd\_r.jpg">

在**History**目录下发生了这样的改变。会在**History**目录下形成一天时间线，来指出每一次的修改标题和内容，同时会把修改的内容用**绿色标识**标出。 我们打开本地的文本，删除刚刚添加的第一行**这个是测试文本**。

<img src="https://pic3.zhimg.com/50/695dd572610b2831c914d1455e14e3a7\_hd.jpg" data-rawwidth="1920" data-rawheight="1035" class="origin\_image zh-lightbox-thumb" width="1920" data-original="https://pic3.zhimg.com/695dd572610b2831c914d1455e14e3a7\_r.jpg">

此时你就会发现github发生了变化。 此时的**红色标识**标识删除。我们写好Summer和Description并点击Commit to master。 这样我们就删除了第一行。同时在**History**目录下又多了一条时间轴。

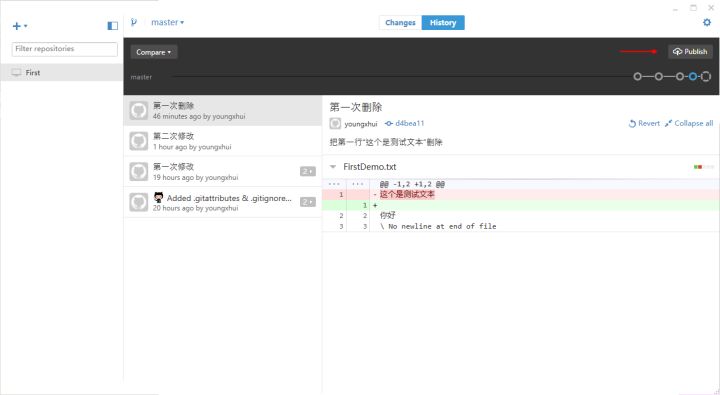
<img src="https://pic1.zhimg.com/50/420ec1f70c23f795ae0f03b3c0b95f03\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic1.zhimg.com/420ec1f70c23f795ae0f03b3c0b95f03\_r.jpg">

这样我们就完成了删除。

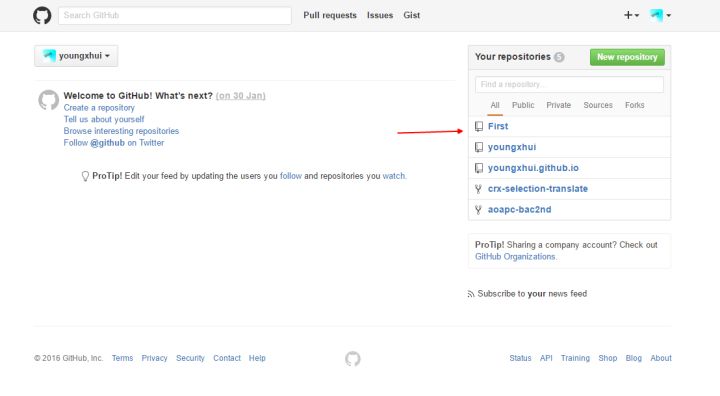
**上传与同步**

**上传**

此时，当我们打开github网页，就会发现此时你的修改的内容并没有出现在这里。这是因为你没有进行同步，仅仅是在本地就行了修改。此时我们仅仅需要点击右上角的**publish**

<img src="https://pic4.zhimg.com/50/90a5b3dca47f2dc2bd52f7fccbabe4f5\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic4.zhimg.com/90a5b3dca47f2dc2bd52f7fccbabe4f5\_r.jpg">

此时你就会本地内容已经上传到网页上。

<img src="https://pic2.zhimg.com/50/1ca0d556a0b27070abb2210b2452e74c\_hd.jpg" data-rawwidth="1108" data-rawheight="609" class="origin\_image zh-lightbox-thumb" width="1108" data-original="https://pic2.zhimg.com/1ca0d556a0b27070abb2210b2452e74c\_r.jpg">

**同步**

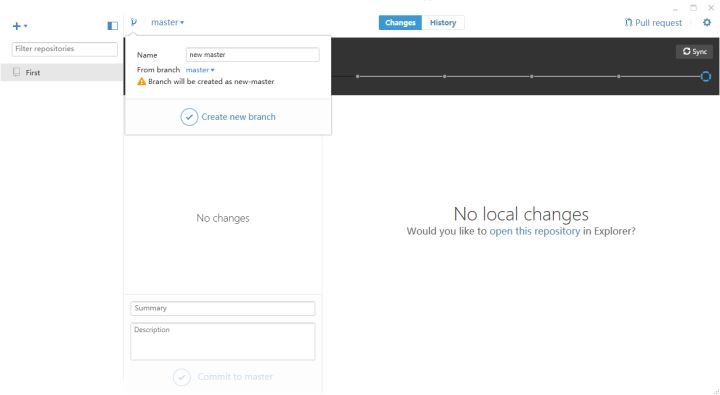
当你的代码库上传后就会发现，原来的**publish**以及变为了**Sync**。 点击**Sync**同步代码库！

<img src="https://pic3.zhimg.com/50/ad2db6627ad067becc9cdf4941523214\_hd.jpg" data-rawwidth="418" data-rawheight="96" class="content\_image" width="418">

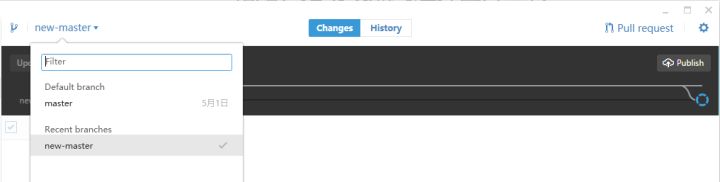
**分支的使用**

**创建分支**

我们创建第一个分支取名为**new masterh,**点击**Create new branch**创建第一个分支**。**

<img src="https://pic2.zhimg.com/50/6e539ed50af4c2aa33df4c487a45e462\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic2.zhimg.com/6e539ed50af4c2aa33df4c487a45e462\_r.jpg">

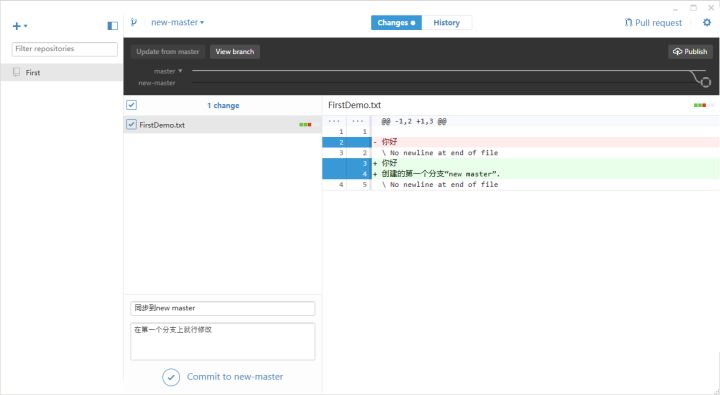
我们发现此时的分支已经切换到了我们刚刚创建的分支**new masterch** 。

<img src="https://pic1.zhimg.com/50/3e2f6ff6926c608cded2db22baf64ebe\_hd.jpg" data-rawwidth="1017" data-rawheight="257" class="origin\_image zh-lightbox-thumb" width="1017" data-original="https://pic1.zhimg.com/3e2f6ff6926c608cded2db22baf64ebe\_r.jpg">

我们来修改**new masterch**分支上的内容。 我们仍旧打开**FirstDemo.txt**进行编辑。输入以下内容

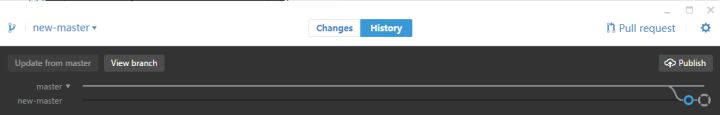
创建的第一个分支。

打开github进行，填写**Summary**和**Description**

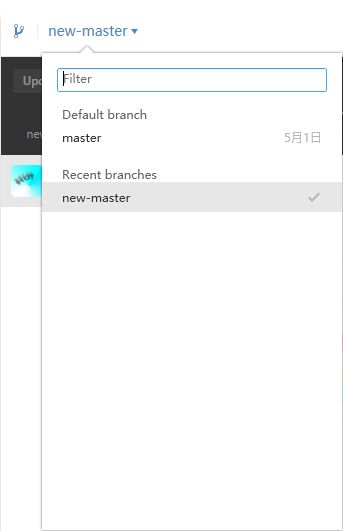
<img src="https://pic3.zhimg.com/50/b199254d043a50dfd07516236cbb083b\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic3.zhimg.com/b199254d043a50dfd07516236cbb083b\_r.jpg">

之后我们点击**Commit to new-master**在**History**目录下，我们可以看到会有两条主线，分别是**master**和**new-master**并且在**new-master**的分支下又一个蓝色的**实线空心圈**和一个**虚线空心圈**。

**实线圈**表示当前的节点，**空心圈**表示下一次修改时的节点。

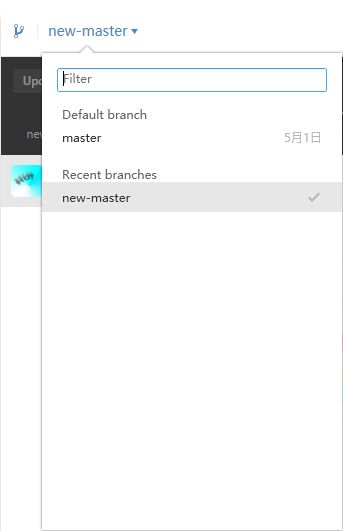
<img src="https://pic4.zhimg.com/50/c658345ff874e7a44a127cf88c808708\_hd.jpg" data-rawwidth="1011" data-rawheight="162" class="origin\_image zh-lightbox-thumb" width="1011" data-original="https://pic4.zhimg.com/c658345ff874e7a44a127cf88c808708\_r.jpg">

**红线**标示的部分就是当前的分支

<img src="https://pic2.zhimg.com/50/87fdd7675d9c09b750eec11d9d829f7e\_hd.jpg" data-rawwidth="343" data-rawheight="531" class="content\_image" width="343">

**切换分支**

点击**红色**划线部分就会出现分支的列表

<img src="https://pic2.zhimg.com/50/87fdd7675d9c09b750eec11d9d829f7e\_hd.jpg" data-rawwidth="343" data-rawheight="531" class="content\_image" width="343">

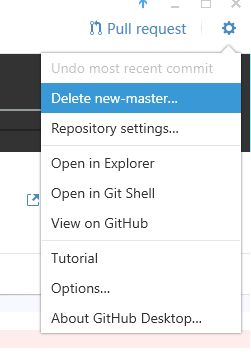
我们点击**master**就会切换到**master**分支。

**上传/同步分支**

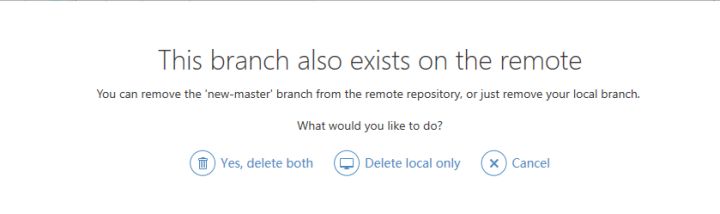
这个操作和同步仓库是一个操作，点击**Publish/Sync**上传或同步分支。

**删除分支**

首先要把分支切换到你要删除的分支下，如我们要删除**new master**，将分支切换到**new master**

<img src="https://pic2.zhimg.com/50/494470bea0eca387f7ca9407b5280d42\_hd.jpg" data-rawwidth="251" data-rawheight="352" class="content\_image" width="251">

点击右上角齿轮就会出现**Delete new master** 点击**Delete new master**就会弹出一个对话框，询问删除的内容。

<img src="https://pic1.zhimg.com/50/2753fc132872815dd8b1c43ca3507580\_hd.jpg" data-rawwidth="839" data-rawheight="249" class="origin\_image zh-lightbox-thumb" width="839" data-original="https://pic1.zhimg.com/2753fc132872815dd8b1c43ca3507580\_r.jpg">

第一个**yes ，Delete both**是将本地与网页全部删除；

第二个**Delete local only**仅仅是删除本地。

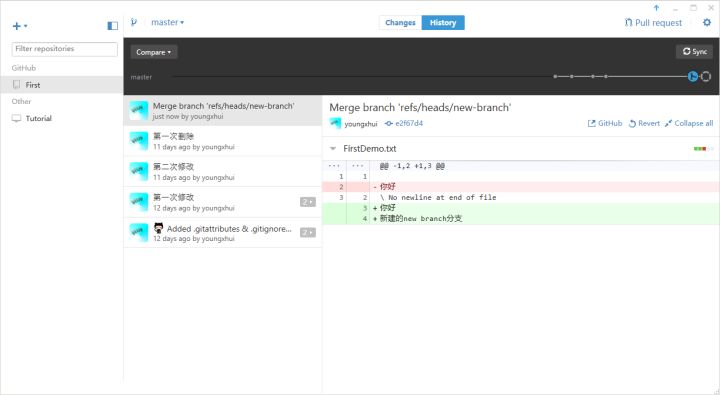
第三个是取消。

**合并两个分支**

将一个分支与**master**分支进行合并。 我们首先把分支切换到**master**下，点击**Update from new-branch**进行分支的合并。

<img src="https://pic1.zhimg.com/50/7c763cf71b32fa54cd9c4ad0ad2d340a\_hd.jpg" data-rawwidth="1011" data-rawheight="161" class="origin\_image zh-lightbox-thumb" width="1011" data-original="https://pic1.zhimg.com/7c763cf71b32fa54cd9c4ad0ad2d340a\_r.jpg">

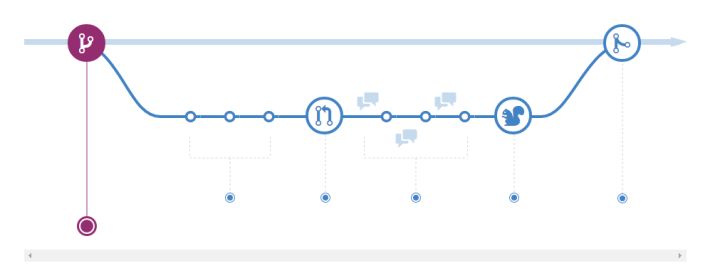
此时我们查看**history**目录下就会

<img src="https://pic2.zhimg.com/50/1374e10c54b4a09835d9d56e0cb0b302\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic2.zhimg.com/1374e10c54b4a09835d9d56e0cb0b302\_r.jpg">

**团队协作流程**

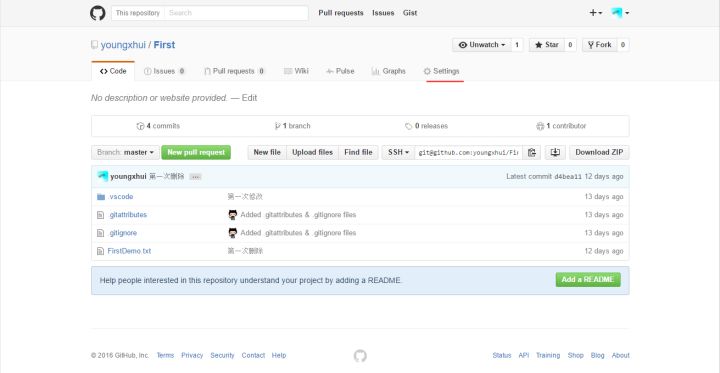
**认识Flow**

[**GitHub Flow**](https://link.zhihu.com/?target=https%3A//guides.github.com/introduction/flow/)是一个轻量级的，基于分支的工作流程，支持团队和部署在那里的定期做项目。

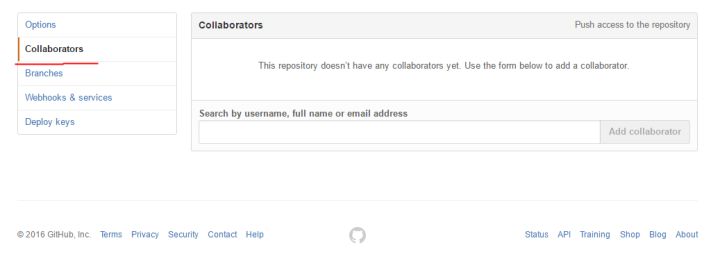
<img src="https://pic1.zhimg.com/50/cf9c119f8b0d6faa804a8ad44e9bae84\_hd.jpg" data-rawwidth="1011" data-rawheight="391" class="origin\_image zh-lightbox-thumb" width="1011" data-original="https://pic1.zhimg.com/cf9c119f8b0d6faa804a8ad44e9bae84\_r.jpg">

**为团队成员写入权限**

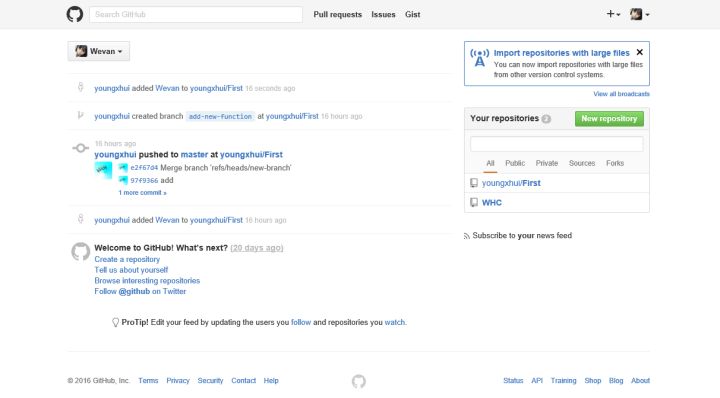
在我们的队友添加一个写的权限，这样我们的队友才能很好的修改代码。 我们打开网页上的[GitHub](https://link.zhihu.com/?target=http%3A//www.github.com/),点击**settings**,

<img src="https://pic1.zhimg.com/50/c96b3f7e7e09c01de3b7f7fc99842568\_hd.jpg" data-rawwidth="1311" data-rawheight="680" class="origin\_image zh-lightbox-thumb" width="1311" data-original="https://pic1.zhimg.com/c96b3f7e7e09c01de3b7f7fc99842568\_r.jpg">

之后我们找到**collaborators**，这里会让我们验证密码，之后就有添加合作者的选项。

<img src="https://pic1.zhimg.com/50/329402a87a0ebc9de0de35d7746b6fbb\_hd.jpg" data-rawwidth="1037" data-rawheight="376" class="origin\_image zh-lightbox-thumb" width="1037" data-original="https://pic1.zhimg.com/329402a87a0ebc9de0de35d7746b6fbb\_r.jpg">

这样我们就能添加我们的小伙伴了！ 这样我们就添加了新的小伙伴，新的小伙伴有着同样的权限去修改和管理代码。 此时我们就会看到我的小伙伴wevan的github主页上就会出现关于我创建的First的各种通知。

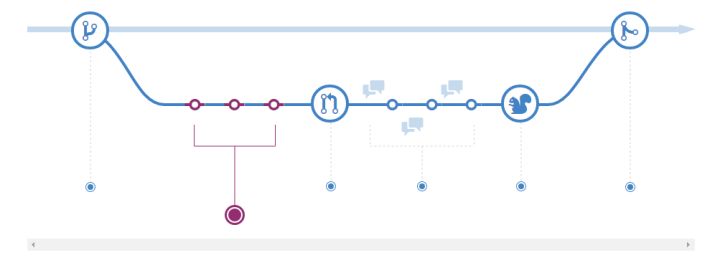
<img src="https://pic3.zhimg.com/50/cfbc506bd1b3a2e5d86938ea8e8d48f5\_hd.jpg" data-rawwidth="1212" data-rawheight="670" class="origin\_image zh-lightbox-thumb" width="1212" data-original="https://pic3.zhimg.com/cfbc506bd1b3a2e5d86938ea8e8d48f5\_r.jpg">

**创建分支**

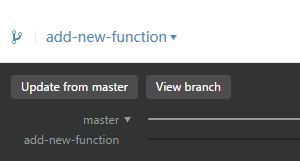
在我们创建一个叫**add new function**的分支。

创建一个分支 *Create a branch* 当你工作的一个项目，你会在任何给定的时间有一堆不同的功能或正在进行的想法 - 其中一些是蓄势待发，而另一些则不是。分支的存在是为了帮助你管理这个工作流程。 *When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.* 当您在项目中创建一个分支，你创造一个环境，在那里你可以尝试新的想法。你让一个分支的更改不会影响主分支，让你可以自由进行实验，并提交更改，在你的分支将不会被合并，直到它准备好知识安全的人所正在与合作进行审查。 *When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.* ProTip 分支在Git中是一个核心概念，整个GitHub的流量是基于它。这里只有一个规则：在任何主分支总是部署。 *Branching is a core concept in Git, and the entire GitHub Flow is based upon it. There's only one rule: anything in the master branch is always deployable.* 正因为如此，这是非常重要的一个功能或修复工作时，你的新分支关老爷的创建。您的分支名应该是描述（例如，重构的身份验证，用户的内容缓存键，使视网膜-化身），以便其他人可以看到正在处理。 *Because of this, it's extremely important that your new branch is created off of master when working on a feature or a fix. Your branch name should be descriptive (e.g., refactor-authentication, user-content-cache-key, make-retina-avatars), so that others can see what is being worked on.*[**来自GitHub Flow**](https://link.zhihu.com/?target=https%3A//guides.github.com/introduction/flow/)

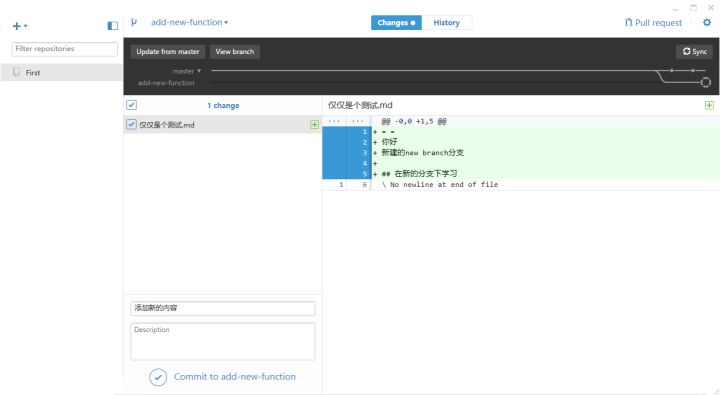
**添加提交**

<img src="https://pic4.zhimg.com/50/28d743aa072a6423fa4748f18be13eaf\_hd.jpg" data-rawwidth="1003" data-rawheight="374" class="origin\_image zh-lightbox-thumb" width="1003" data-original="https://pic4.zhimg.com/28d743aa072a6423fa4748f18be13eaf\_r.jpg">

我们首先把分支切换到新的分支上**add new function**

<img src="https://pic3.zhimg.com/50/fd8ba40905161453c4b150ae92c98bc7\_hd.jpg" data-rawwidth="300" data-rawheight="161" class="content\_image" width="300">

修改新的版本

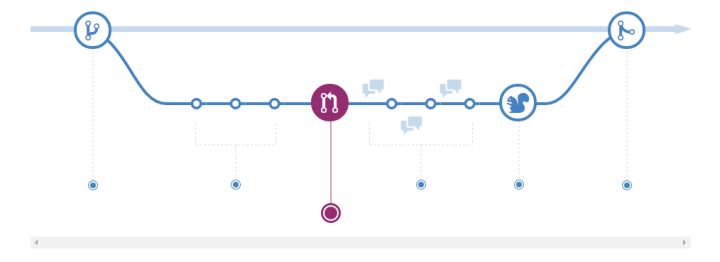
<img src="https://pic2.zhimg.com/50/719dde1748f8170e7191f8f5069ac375\_hd.jpg" data-rawwidth="1224" data-rawheight="672" class="origin\_image zh-lightbox-thumb" width="1224" data-original="https://pic2.zhimg.com/719dde1748f8170e7191f8f5069ac375\_r.jpg">

填写好新的**Summary**和**Description**，提交新的版本并同步。 这样小伙伴登陆到**GitHub**上就看到了就可以清楚的看到一切的修改。

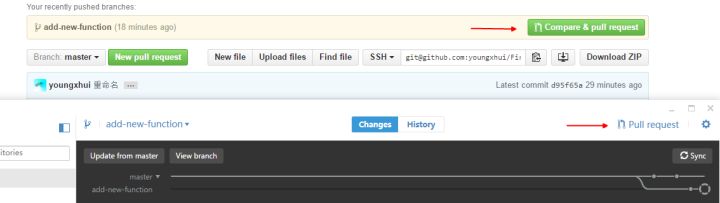
添加提交 *Add commits* 一旦你的分支已经建立，现在是时候开始进行更改。无论何时添加，编辑或删除一个文件，你作出承诺，并将其添加到您的分支。提交加入这一过程保持你的进步轨迹，你在一个特性分支工作。 *Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.* 还承诺创建工作的透明历史，其他人可以按照理解你做了什么，以及为什么。每次提交都有一个关联的提交信息，这是解释为什么一个特定的变化作出了说明。此外，每次提交被认为是变革的一个独立单元。这使您可以回滚的变化，如果发现错误，或者如果你决定在一个不同的方向前进。 *Commits also create a transparent history of your work that others can follow to understand what you've done and why. Each commit has an associated commit message, which is a description explaining why a particular change was made. Furthermore, each commit is considered a separate unit of change. This lets you roll back changes if a bug is found, or if you decide to head in a different direction.* ProTip

提交信息是重要的，特别是因为Git跟踪更改，然后将它们显示为承诺一旦他们推到服务器。通过字迹清晰提交信息，你可以更容易为其他人跟着，并提供反馈。 *Commit messages are important, especially since Git tracks your changes and then displays them as commits once they're pushed to the server. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.*[**来自GitHub Flow**](https://link.zhihu.com/?target=https%3A//guides.github.com/introduction/flow/)

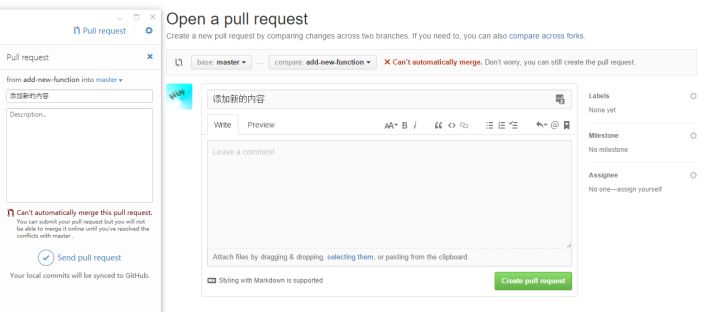
**打开一个pull请求**

<img src="https://pic2.zhimg.com/50/9c66d35939ee5274ec37a264f4f62580\_hd.jpg" data-rawwidth="1014" data-rawheight="369" class="origin\_image zh-lightbox-thumb" width="1014" data-original="https://pic2.zhimg.com/9c66d35939ee5274ec37a264f4f62580\_r.jpg">

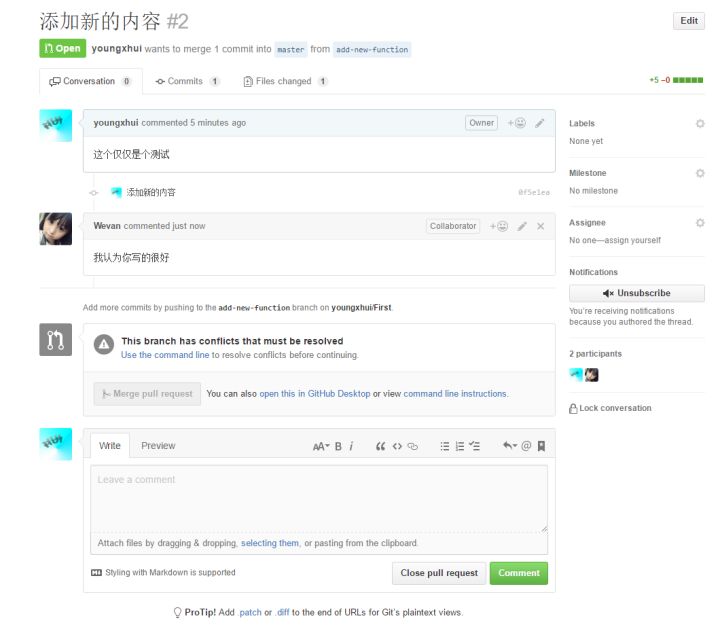
这个是整个流程中比较关键的一步，发布**Pull Request**。

<img src="https://pic3.zhimg.com/50/d6b4ace033d32b20fb2975b5c9b4c18e\_hd.jpg" data-rawwidth="1134" data-rawheight="319" class="origin\_image zh-lightbox-thumb" width="1134" data-original="https://pic3.zhimg.com/d6b4ace033d32b20fb2975b5c9b4c18e\_r.jpg">

点击客户端或者网页上的**Pull Request**发布。 我们这里点击**Pull Request**

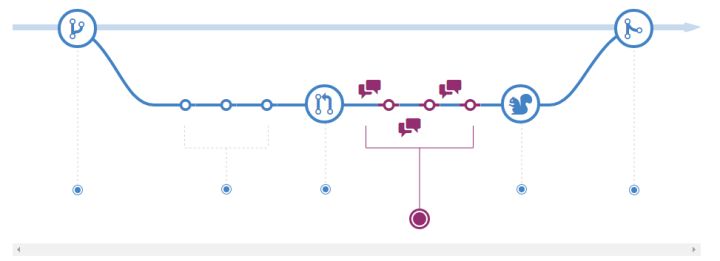
<img src="https://pic3.zhimg.com/50/235a55e9f0855426ab2e898ccdedb727\_hd.jpg" data-rawwidth="1329" data-rawheight="575" class="origin\_image zh-lightbox-thumb" width="1329" data-original="https://pic3.zhimg.com/235a55e9f0855426ab2e898ccdedb727\_r.jpg">

我们填写好必要的说明性文字 点击**Send Pull Request** 他既然让我们到GitHub上看，我们就听他的，点击，进入。

<img src="https://pic3.zhimg.com/50/1824a8ae56aa97ff1b02e0ab1f1a274f\_hd.jpg" data-rawwidth="1060" data-rawheight="923" class="origin\_image zh-lightbox-thumb" width="1060" data-original="https://pic3.zhimg.com/1824a8ae56aa97ff1b02e0ab1f1a274f\_r.jpg">

我们发现小伙伴已经在下面留言了！

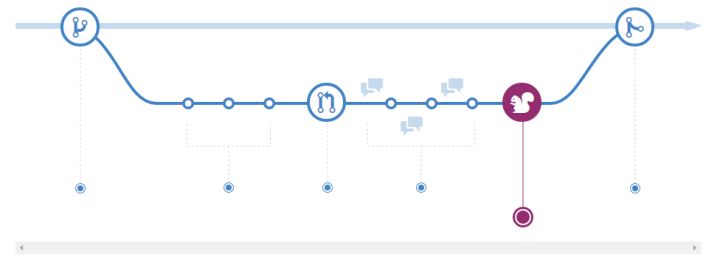
**讨论和审核你的代码**

<img src="https://pic3.zhimg.com/50/d85ebb2cb8eab013c3025e4370262e94\_hd.jpg" data-rawwidth="973" data-rawheight="353" class="origin\_image zh-lightbox-thumb" width="973" data-original="https://pic3.zhimg.com/d85ebb2cb8eab013c3025e4370262e94\_r.jpg">

你的小伙伴开始对你的代码讨论，修改，迭代。

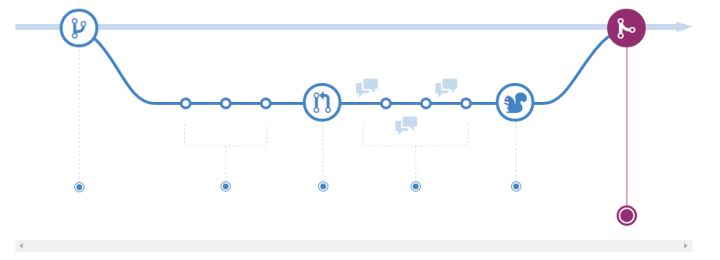
讨论和审查你的代码 Discuss and review your code 一旦拉入请求已被打开，人或团队审查您的变化可能有疑问或意见。也许编码风格不匹配项目的指导方针，改变缺少单元测试，或者也许一切看起来不错，道具都是为了。引入请求旨在鼓励并捕获这种类型的对话。 Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and capture this type of conversation. 您还可以继续推送到你的分支在你提交的讨论和反馈光。如果有人评论说，你忘了做某件事，或者如果在代码中的错误，你可以在你的分支修复它，推高的变化。GitHub上会显示新的提交和其他任何意见，你可能会收到统一拉请求视图。 You can also continue to push to your branch in light of discussion and feedback about your commits. If someone comments that you forgot to do something or if there is a bug in the code, you can fix it in your branch and push up the change. GitHub will show your new commits and any additional feedback you may receive in the unified Pull Request view.  
ProTip 拉请求的意见都写在降价，所以你可以插入图片和表情符，使用预先格式化的文本块，等轻质格式。 Pull Request comments are written in Markdown, so you can embed images and emoji, use pre-formatted text blocks, and other lightweight formatting.

**部署**

<img src="https://pic1.zhimg.com/50/e953313d37c6d634cef64375324cc15c\_hd.jpg" data-rawwidth="976" data-rawheight="348" class="origin\_image zh-lightbox-thumb" width="976" data-original="https://pic1.zhimg.com/e953313d37c6d634cef64375324cc15c\_r.jpg">

部署 Deploy 一旦你拉的请求进行了审查和部门通过你的测试，您可以部署您的更改，以验证他们的生产。如果你的分支造成的问题，您可以通过部署现有的主投产回滚 Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them in production. If your branch causes issues, you can roll it back by deploying the existing master into production.

**合并**

<img src="https://pic4.zhimg.com/50/c0fa9ed539774519b044402d090bb936\_hd.jpg" data-rawwidth="989" data-rawheight="357" class="origin\_image zh-lightbox-thumb" width="989" data-original="https://pic4.zhimg.com/c0fa9ed539774519b044402d090bb936\_r.jpg">

合并分支我们之前已经说过，这里就不再赘述。

合并 Merge 现在，您的更改在生产中得到了验证，现在是时候你的代码合并到主分支。 Now that your changes have been verified in production, it is time to merge your code into the master branch. 合并后，引入请求保护的历史变迁到您的代码记录。因为他们是搜索的，他们不让任何人回去的时间理解为什么以及如何决定了。 Once merged, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.  
ProTip 通过将某些关键字到您的拉请求的文本，你可以用代码相关联的问题。当你拉入请求合并，相关问题也将被关闭。例如，输入短语关闭＃32将关闭在仓库中发行数量32。欲了解更多信息，请查看我们的帮助文章。 By incorporating certain keywords into the text of your Pull Request, you can associate issues with code. When your Pull Request is merged, the related issues are also closed. For example, entering the phrase Closes #32 would close issue number 32 in the repository. For more information, check out our help article.

基本的一些用法就完成了。看着这个操作一遍基础就差不多了。

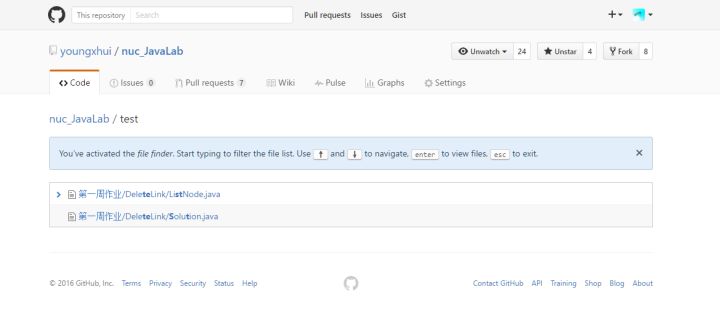
这个知乎的编辑啊…………好累

----------------------------------------------------------------------------------------------------------------

在上述的几个教程里讲解了一些Github的基础使用，现在开始讲解一些使用技巧。

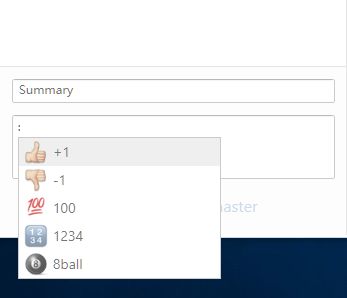
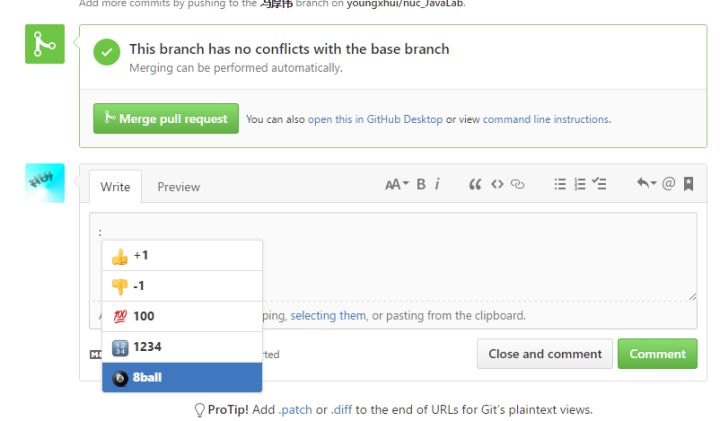
**查找内容**

在github页面上是没有搜索的按钮，如何搜索呢。 在网页上按**T**就会出现。 这样我们就能很方便的查找到我们需要的代码了。

<img src="https://pic4.zhimg.com/50/ce653075b54f0fe5c51c33eabce6a4e7\_hd.jpg" data-rawwidth="1169" data-rawheight="513" class="origin\_image zh-lightbox-thumb" width="1169" data-original="https://pic4.zhimg.com/ce653075b54f0fe5c51c33eabce6a4e7\_r.jpg">

**评论小表情**

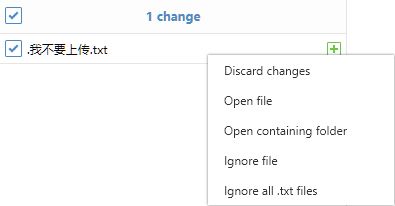
常常在**版本描述**或者**pull request**时我们需要对伙伴的代码进行一下评论与说明，光是文字有点很死板，其实github给我有**emoji**，如何使用呢？其实很简单，只需要**冒号**就可以 ：

<img src="https://pic2.zhimg.com/50/0eef2008c96d58863ee3567d7853eeb7\_hd.jpg" data-rawwidth="347" data-rawheight="298" class="content\_image" width="347"><img src="https://pic4.zhimg.com/50/a5e608990731897e51be019da5fab544\_hd.jpg" data-rawwidth="802" data-rawheight="469" class="origin\_image zh-lightbox-thumb" width="802" data-original="https://pic4.zhimg.com/a5e608990731897e51be019da5fab544\_r.jpg">

这样我们就可以看到emoji表情，当然默认会显示五个常用的，你也可以继续敲下emoji的名字，出现更多（[这里有所有的表情](https://link.zhihu.com/?target=http%3A//www.webpagefx.com/tools/emoji-cheat-sheet/)）。

**忽略不想上传的文件**

有些在github中的文件我们是不想上传的，我们如何过滤掉它们呢？ 在github中对不想上传的文件点击右键。就会出现下面选项。 **Ignore file**忽略这个文件 **Ignore all.txt files** 忽略所有的以.txt结尾的文件 这样就可以过滤掉你不想上传的文件

<img src="https://pic3.zhimg.com/50/811644c7f4b016ba326942185a4f10fe\_hd.jpg" data-rawwidth="395" data-rawheight="206" class="content\_image" width="395">

**搜索项目**

如何高效的搜索一个你想要的库呢？ 我们常常评判一个项目的标准有star数目，fork数目和跟新时间。 通过搜索命令

**stars**

stars:>1000

表示star数目大于1000。

**fork**

fork:>1000

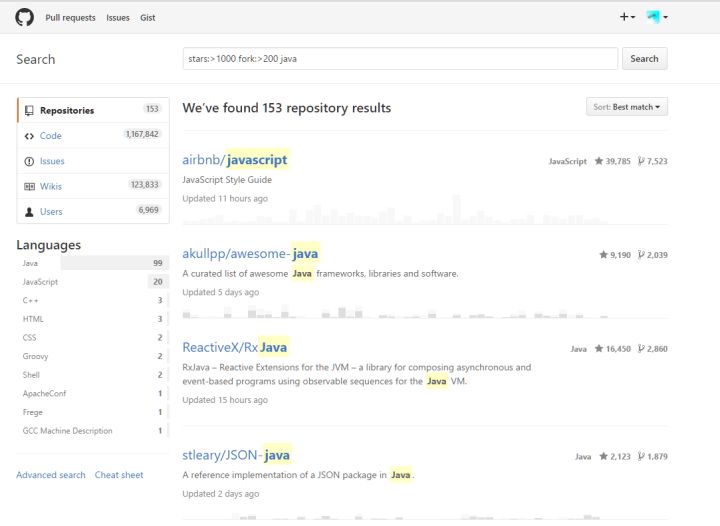
表示fork数目大于1000。

**语言搜索**

java，html等等

综合一下就是，比如你要查找一个stars大于1000的，fork大于200的java代码。

stars:>1000 fork:>200 java

<img src="https://pic4.zhimg.com/50/c3e978786fbe2e21e559b43f08a15714\_hd.jpg" data-rawwidth="1083" data-rawheight="791" class="origin\_image zh-lightbox-thumb" width="1083" data-original="https://pic4.zhimg.com/c3e978786fbe2e21e559b43f08a15714\_r.jpg">

就是这样。

**查看项目中的语言类型**

一个项目中，可能使用了多种语言，我们如何一下子就能看到一个项目使用了什么语言？其实很简单，Github已经为我们统计好了。

<img src="https://pic1.zhimg.com/50/38c5118a5159fa56101d6ce14aa70adc\_hd.jpg" data-rawwidth="1003" data-rawheight="76" class="origin\_image zh-lightbox-thumb" width="1003" data-original="https://pic1.zhimg.com/38c5118a5159fa56101d6ce14aa70adc\_r.jpg">https://pic1.zhimg.com/80/38c5118a5159fa56101d6ce14aa70adc_hd.jpg

也行你注意过，但是没有发现它有什么用。

点击下面的彩条

<img src="https://pic3.zhimg.com/50/fa476a57295ab2e4484aa7c6090c85fa\_hd.jpg" data-rawwidth="1008" data-rawheight="77" class="origin\_image zh-lightbox-thumb" width="1008" data-original="https://pic3.zhimg.com/fa476a57295ab2e4484aa7c6090c85fa\_r.jpg">https://pic3.zhimg.com/80/fa476a57295ab2e4484aa7c6090c85fa_hd.jpg

github已经为我们统计好这个项目所有的语言及其比例。

<img src="https://pic1.zhimg.com/50/75cd02b8fcd34975eb34ee357bfc273e\_hd.jpg" data-rawwidth="1004" data-rawheight="77" class="origin\_image zh-lightbox-thumb" width="1004" data-original="https://pic1.zhimg.com/75cd02b8fcd34975eb34ee357bfc273e\_r.jpg">https://pic1.zhimg.com/80/75cd02b8fcd34975eb34ee357bfc273e_hd.jpg

**总结**

码字不易，终于写完了，如果觉得对你有帮助，我的目的就达到了。

谢谢

如有错误，还望指正。