# Github协作开发文档

开发准备手册第二版

## GitHub多人协作

GitHub常见的多人协作开发分为三种模式（详细请看附件2）：

1. **Fork模式**
2. **合作者模式**
3. **组织模式**

在企业中开发，建议使用组织模式；如果是小规模团队（2-10人）开发，建议使用Fork和合作者模式协同进行开发。

### 分支模型及详解：

我们为git定下一种分支模型，在这种模型中，分支有三种：

**master branch：主分支**

**develop branch：开发分支**

**feature branch：功能分支**

**master：**主分支从项目一开始便存在，它用于存放经过测试，已经完全稳定代码；在项目开发以后的任何时刻当中，master存放的代码应该是可作为产品供用户使用的代码。每一次master更新的时候都应该用git打上tag，说明你的产品有新版本发布了。

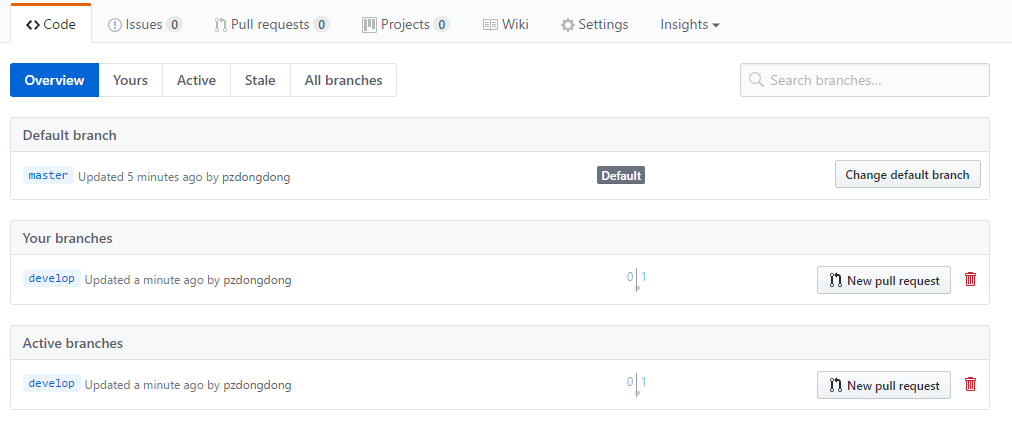
**develop：**开发分支，一开始从master分支中分离出来，用于开发者存放基本稳定代码。开发者把功能做好以后，是存放到自己的develop中，当测试完以后，可以向管理者发起一个pull request，请求把自己仓库的develop分支合并到源仓库的develop中。

**feature**：功能性分支，是用于开发项目的功能的分支，是开发者主要战斗阵地。开发者在本地仓库从develop分支分出功能分支，在该分支上进行功能的开发，开发完成以后再合并到develop分支上，这时候功能性分支已经完成任务，可以删除。功能性分支的命名一般为feature-\*，\*为需要开发的功能的名称。

**总结：**所有开发者开发好的功能会在源仓库的develop分支中进行汇总，当develop中的代码经过不断的测试，已经逐渐趋于稳定了，接近产品目标了。这时候，我们就可以把develop分支合并到master分支中，发布一个新版本。

分支建议按照**功能**进行划分，每一个开发中的功能都独立出一个分支，通过合作者模式进行功能代码编写、功能测试，测试无误后，可以通过Pull Request合并到开发（Develop）分支中，所有功能开发完毕，测试通过后，合并到主分支（Master）中。

**在现在的版本中，当我们生成一个仓库的时候，只会生成默认的master分支。**



### 流程展示：

**1) 源仓库的构建，创建一个项目，初始化了两个永久性分支master和develop.**

**2) 开发者Fork 库(会把master和develop分支一起Fork过去)。**

**3) 把自己开发者仓库clone到本地，命令：git clone**

**4）切换到develop分支，命令：git checkout develop**

**4) 构建功能分支进行开发、测试，完成后合并到自己的develop分支。**

**5) 向管理员提交pull request。**经过测试以后，觉得没问题，就可以请求管理员把自己仓库的develop分支合并到源仓库的develop分支中，这就是传说中的pull request。

**6) 管理员测试、合并**

6.1 对代码进行review，github提供非常强大的代码review功能

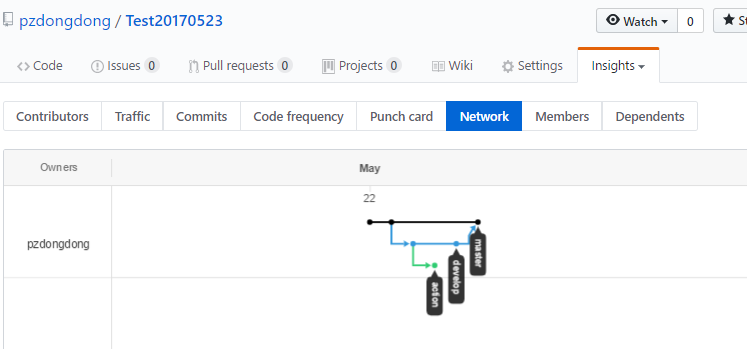
         6.2 在本地测试新建一个测试分支，测试pull request的代码

         6.3 判断是否同意合并到源仓库的develop中，如果经过测试没问题，可以把代码合并到源仓库的develop中

## Git使用规范要求：

### 分支结构

参考下图：



### 分支命名规范：

2.1、功能性分支建议以feature-xxx为主。xxx为功能名称。

### 3、Git使用规范：

3.1、每次推送（push）前，需先更新一下本地库

3.2、未经允许，不得直接push到develop和master分支

3.3、功能性分支需测试无误后方可提交到develop分支

3.4、develop分支需测试完整无Bug后，方可进行发布

3.5、每一次版本发布，必须打上版本号同时备注相应更新信息

### 4、包结构管理

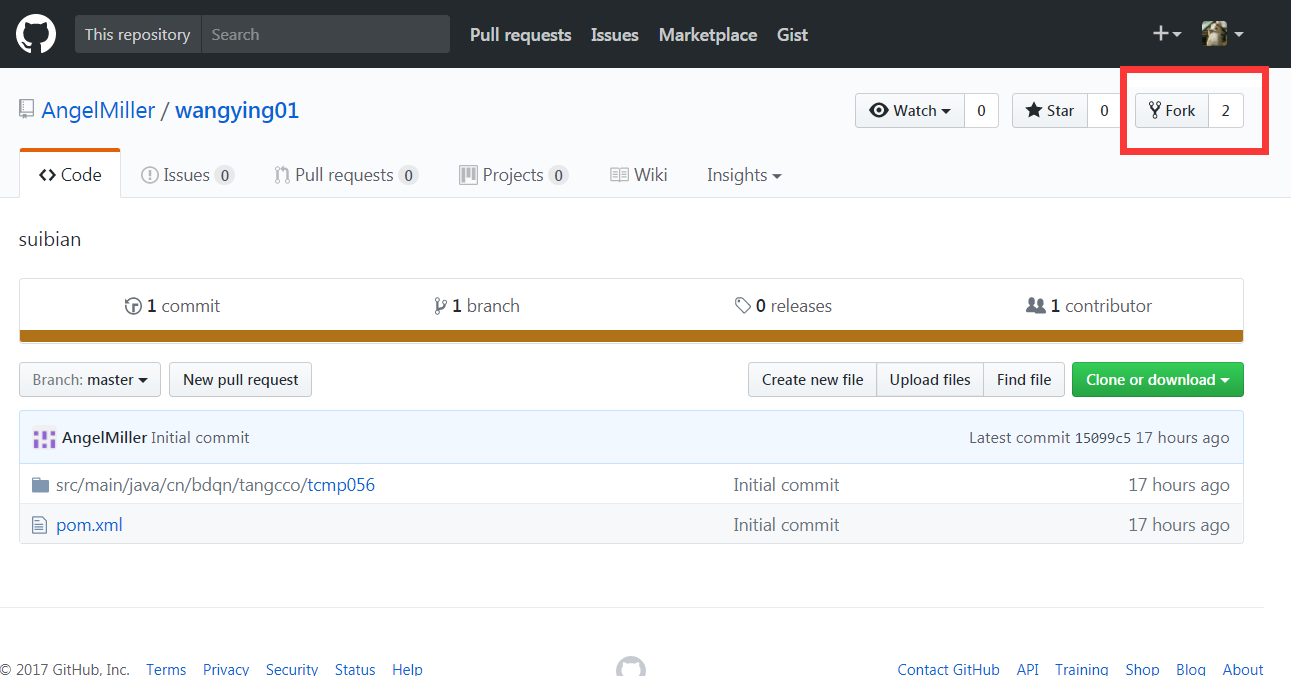
参考MAVEN包结构。

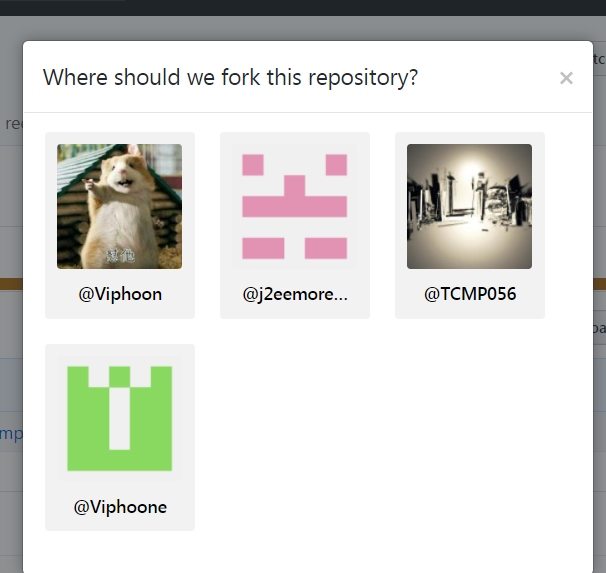
### Fork模式

参考https://github.com/oldratlee/translations/blob/master/git-workflows-and-tutorials/workflow-forking.md

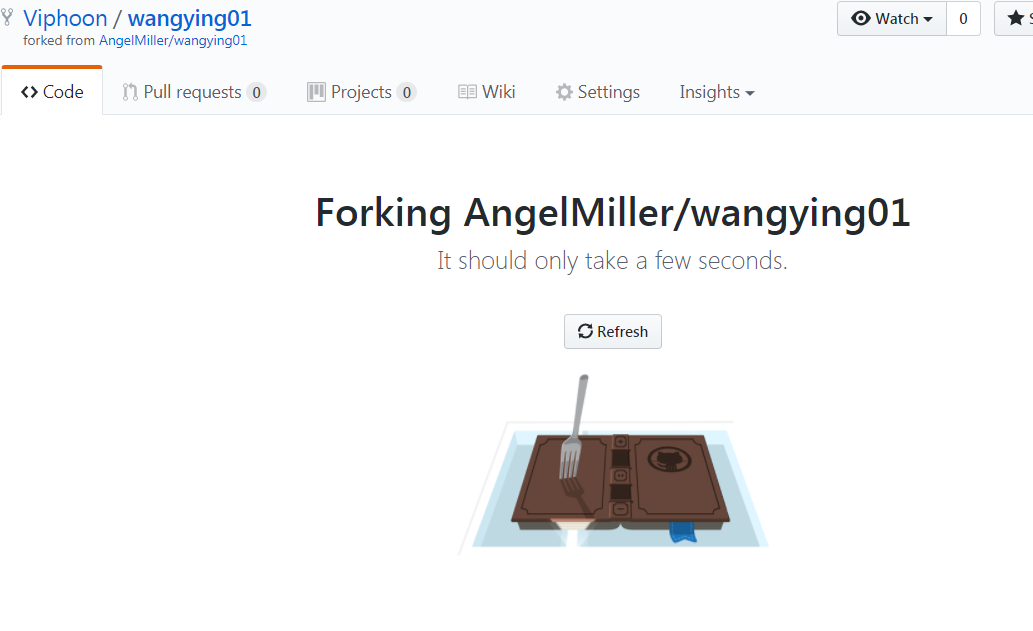
当我们发现比较好的代码的时候可以使用Fork模式来收藏到我们的仓库，代码拥有者每次跟新代码的时候我们都会得到提示。

##### 1.Fork一个项目

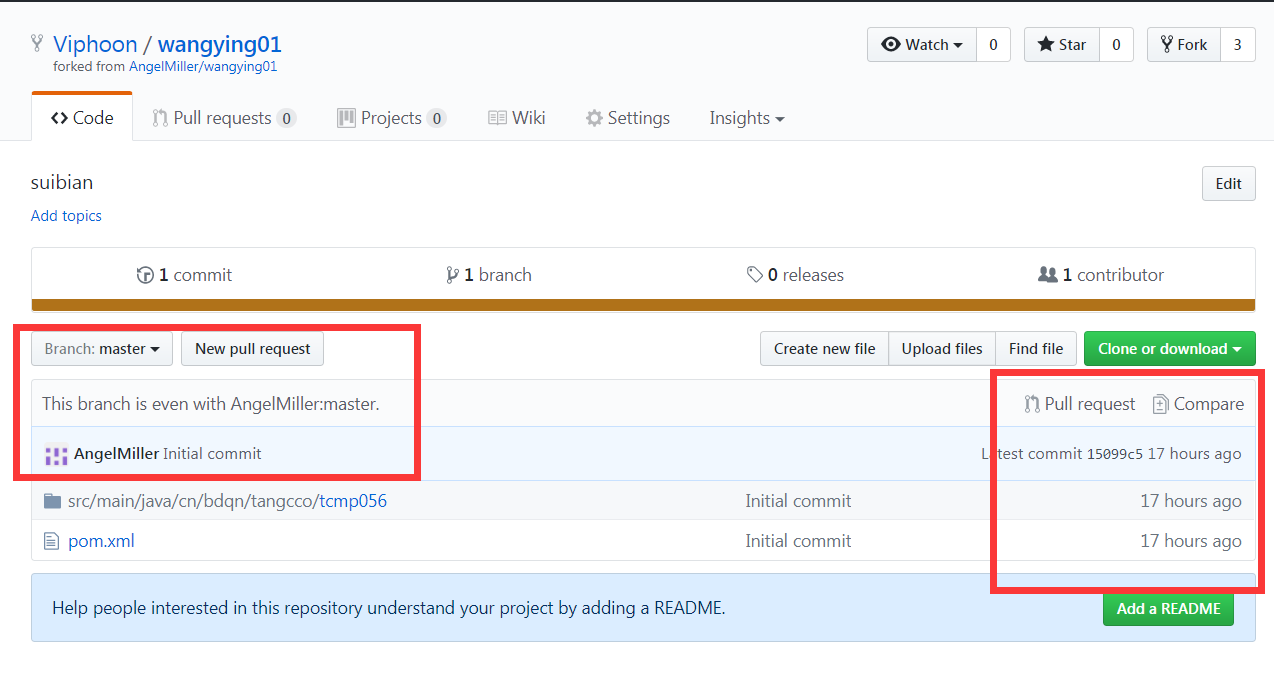




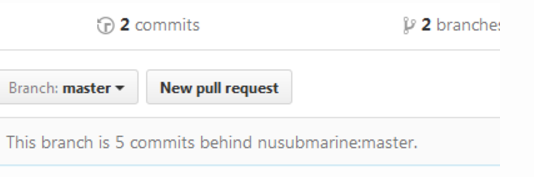
点击之后，就可以把项目克隆到我们自己的GitHub仓库，自己选择fork到哪里的仓库。



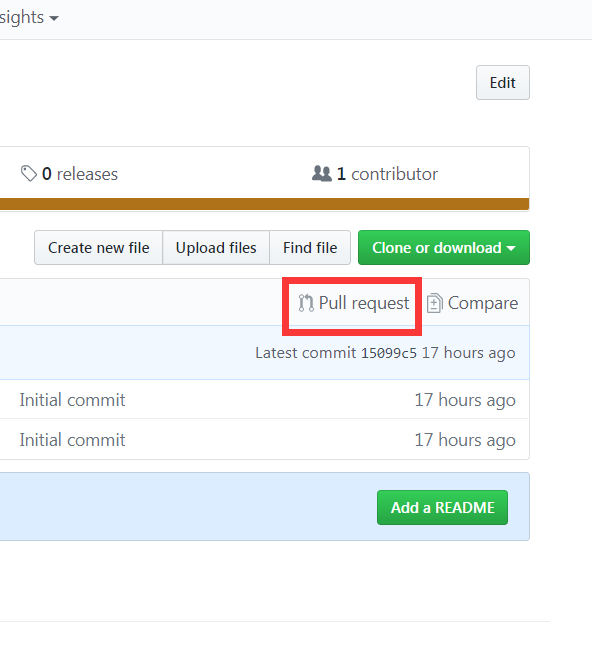
等待页面挑战。



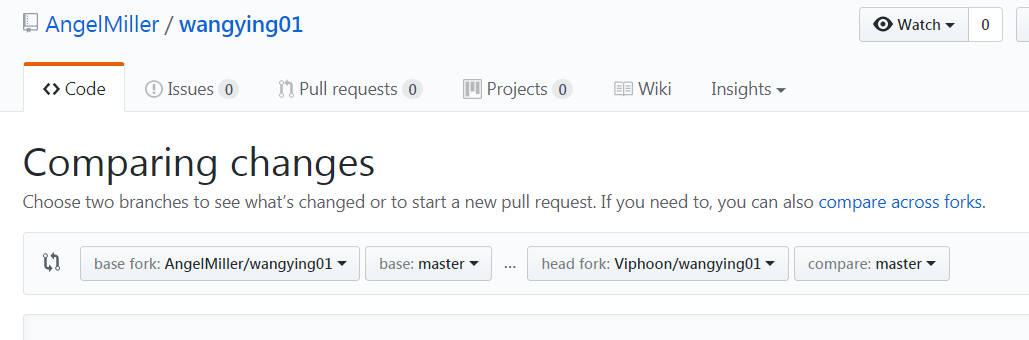
我们在自己的仓库就可以看到整个项目了，整个项目来自哪里，源代码拥有者的每一次具体操作修改记录，这个仓库的使用方法和我们自己的使用方法是一样的。

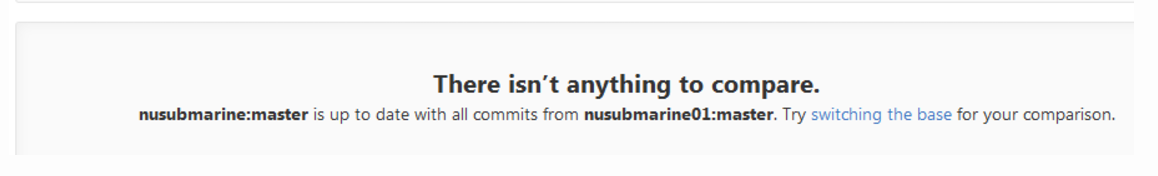


如上图所示，当源代码拥有者改变代码后，我们都会受到提醒，我们可以在仓库中看到下图中的提示：“This branch is 5 commits behind nusubmarine:master.”。表示我们fork的那个分支和我们仓库的当前分支比向前了5个提交。这个时候，我们想要和上游仓库保持同步更新的话，可以点击pull request。

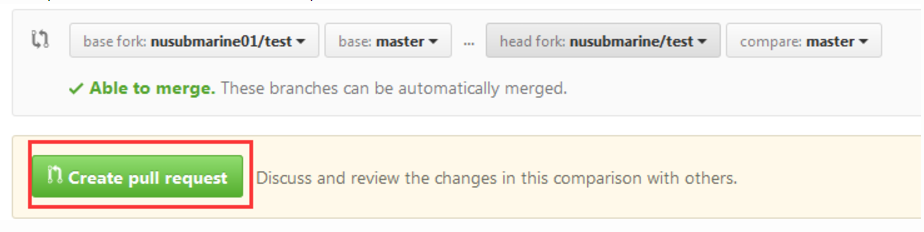


进入到下面的页面。如果我们没有要提交的内容，会直接提示：“Try switching the base for your comparison.”。如果因为自己仓库里有改动，没有这个提示，自己交换两个比较分支的位置。

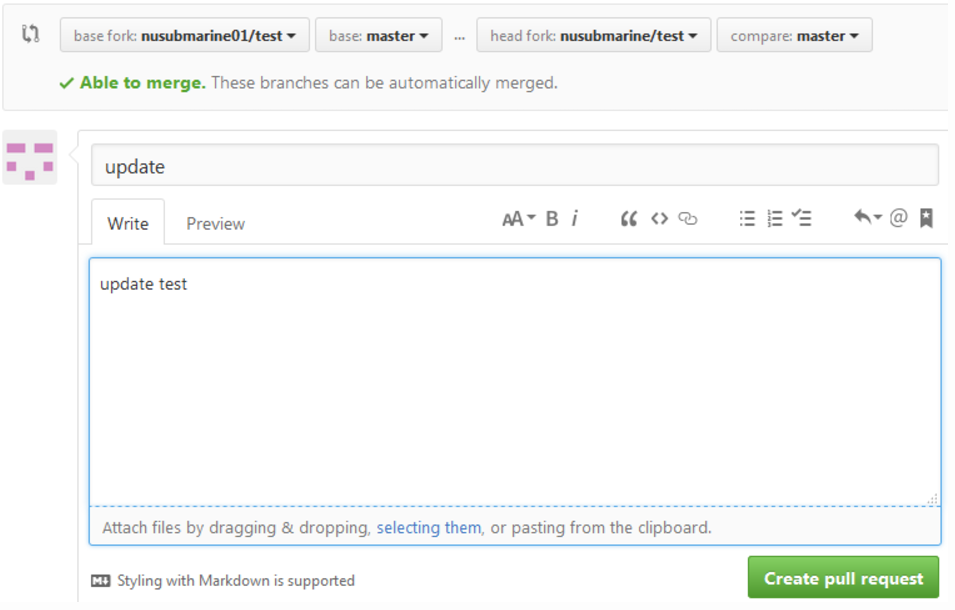




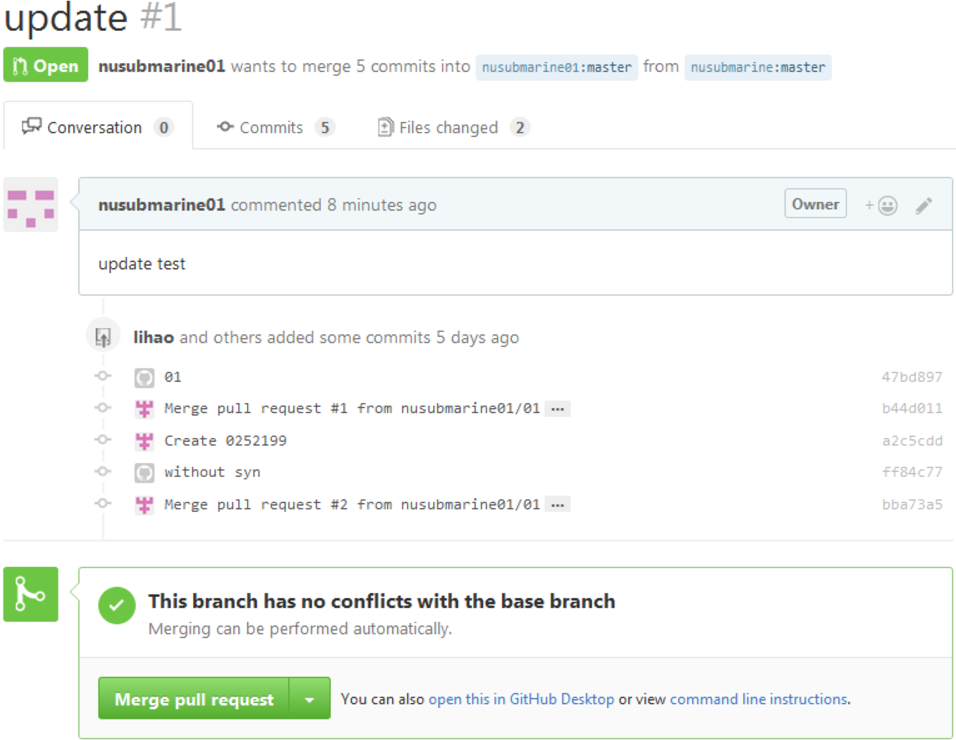
交换之后，就会变成我们拉取上游仓库的分支，点击红色框按钮新建请求。



填写提交的注释，自己在查看历史的时候会比较好理解。



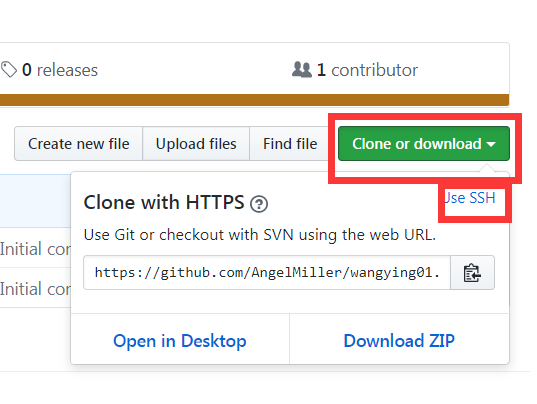
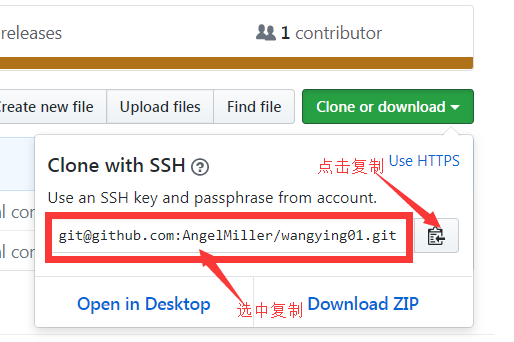
确认之后，提示可以同步，合并并确认。

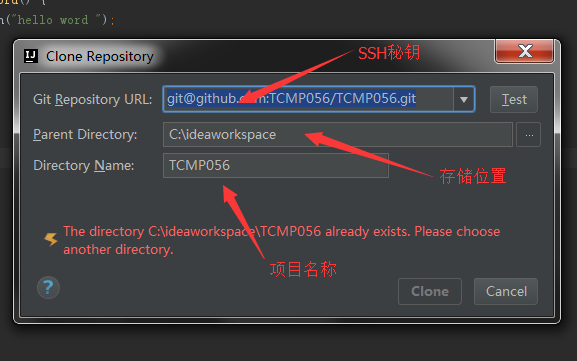
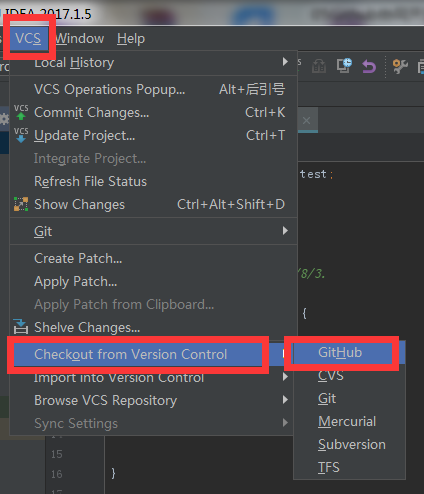


至此，我们就和上游仓库完成同步。

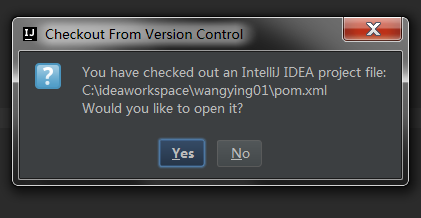
###### 2.下载项目到本地修改

打开idea ，选择如下图所示：填写ssh秘钥（在项目的clone or download处选择use ssh 复制即可）。



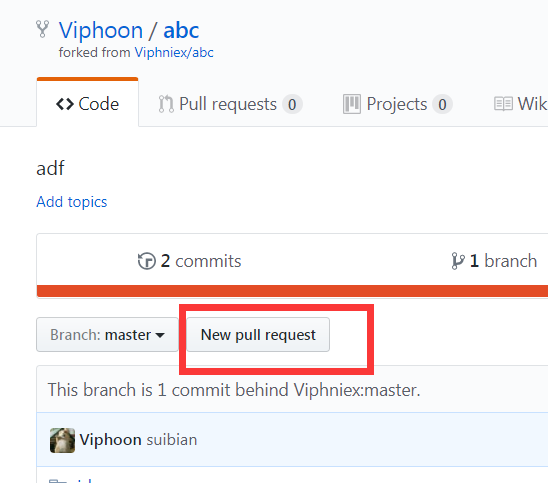
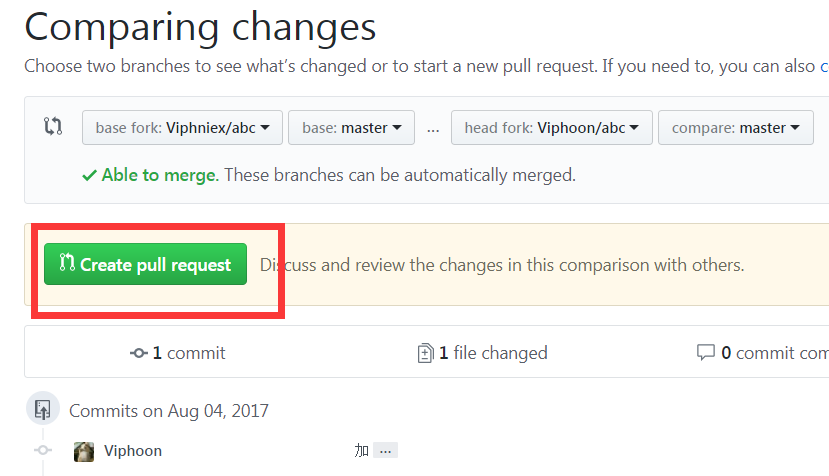
之后点击clone即可。弹出询问是否打开刚才克隆的项目。

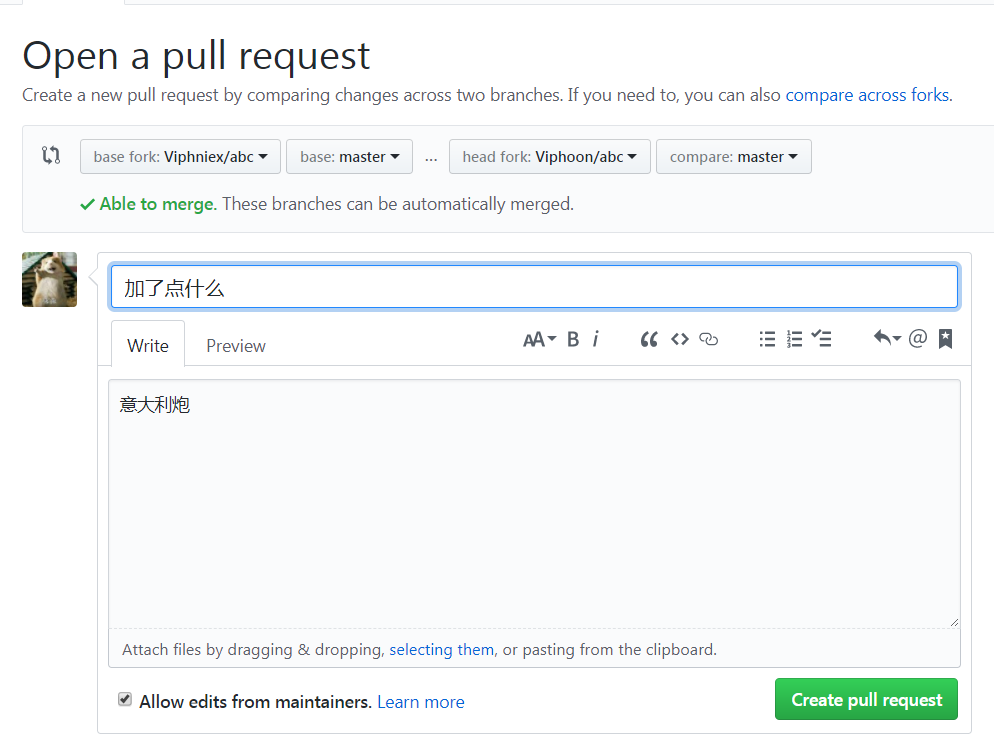


当选择yes的时候，建议使用this Windows 打开，就可以对项目代码进行修改了。修改完成之后直接commit changes 就可以了。

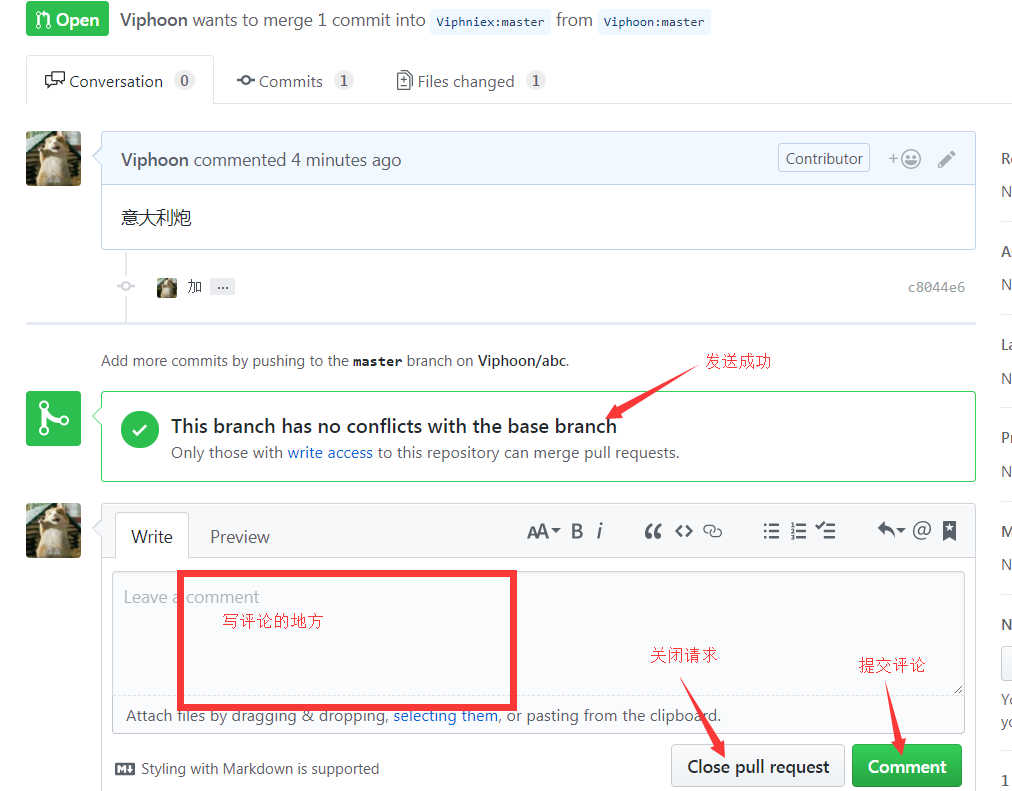
###### 3.发送一个pull request

当我们修改完代码上传到自己的仓库后，我们更新的只是自己的本地仓库，向上游仓库发送pull request如下图

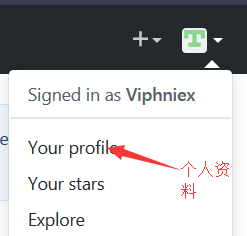
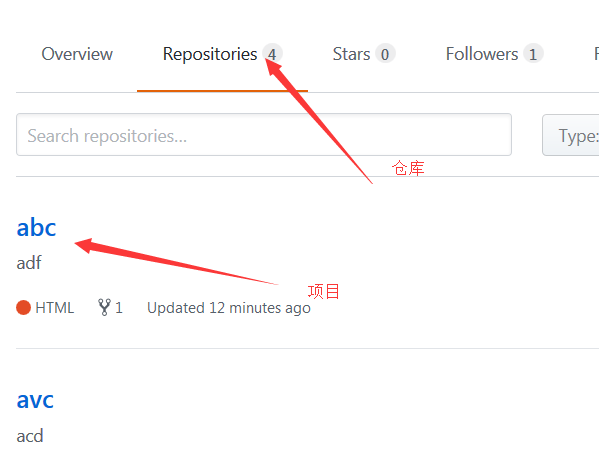
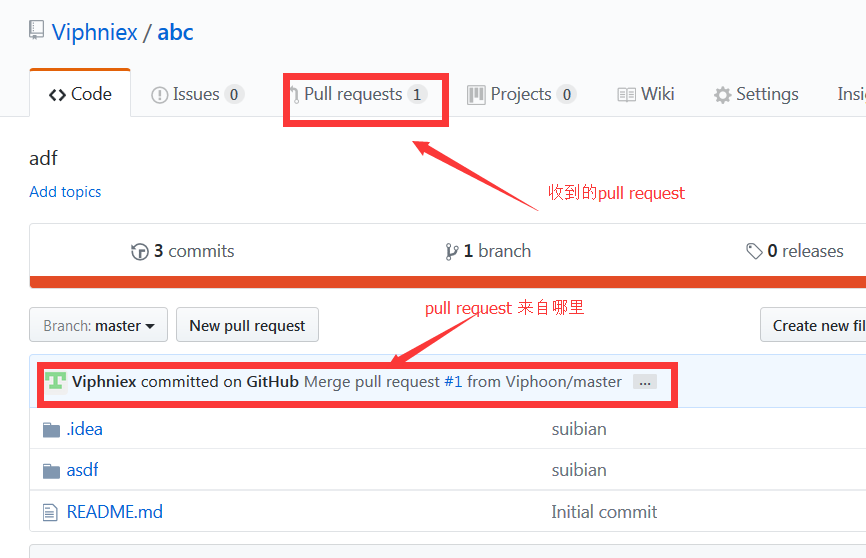
 



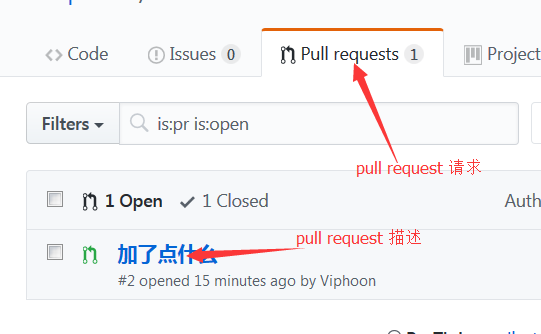
添加描述后点击确认发送就可以了。

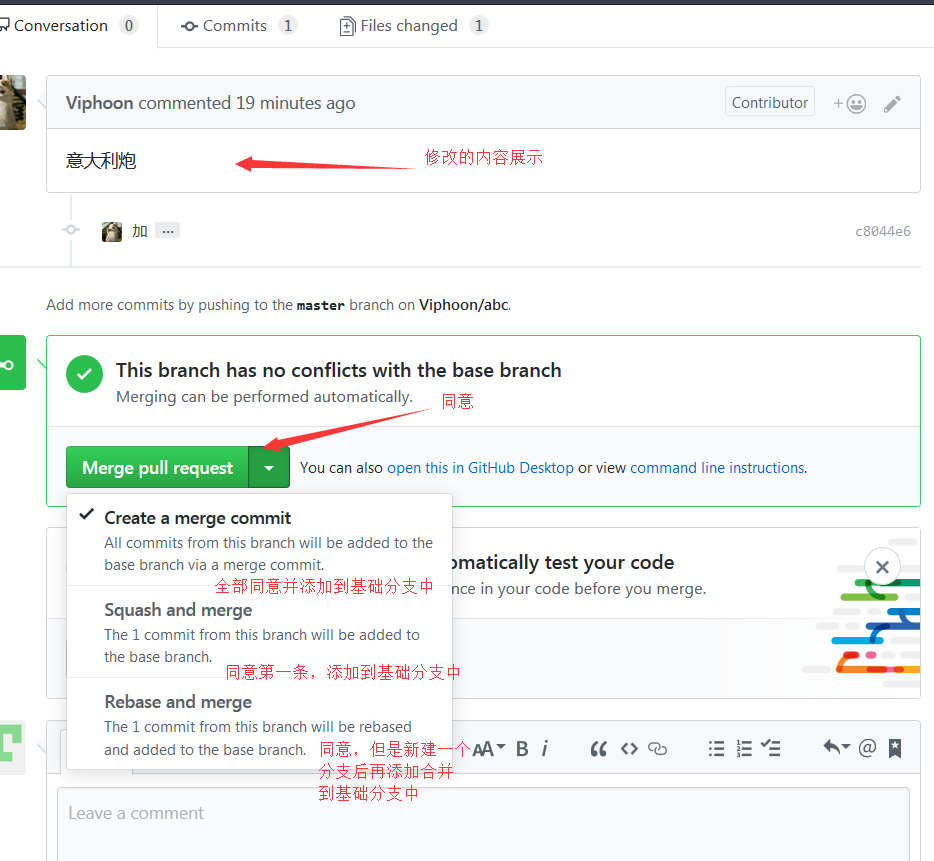
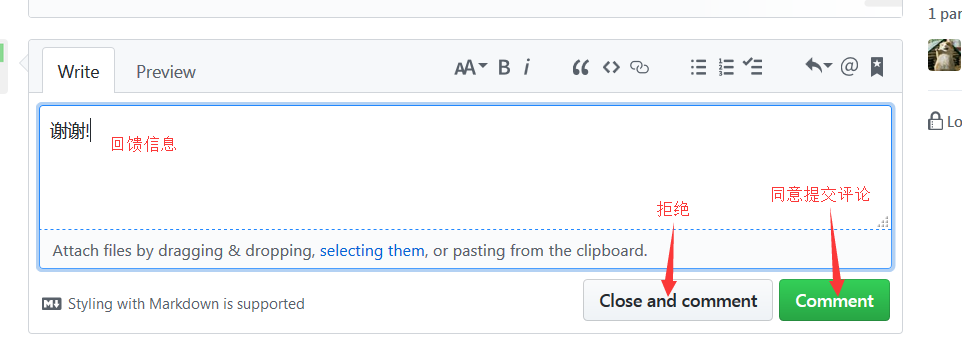


之后对方就会收到我们发送的请求了。其他人给我们发送的请求也是可以看到的，在个人资料中会有提示。

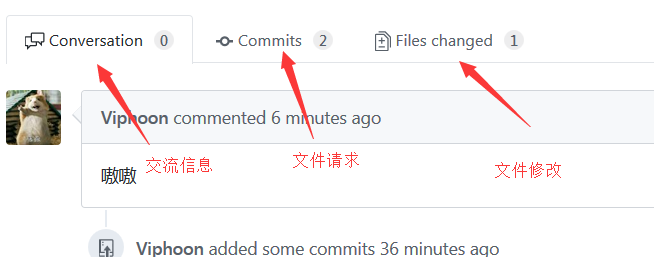
  

点开pull request，点击描述

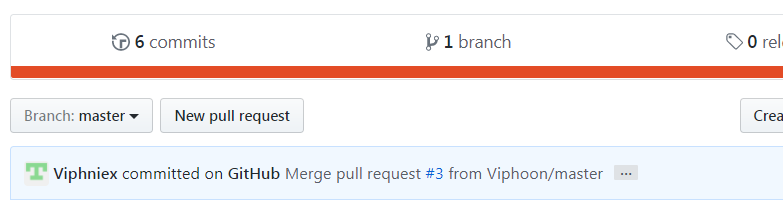


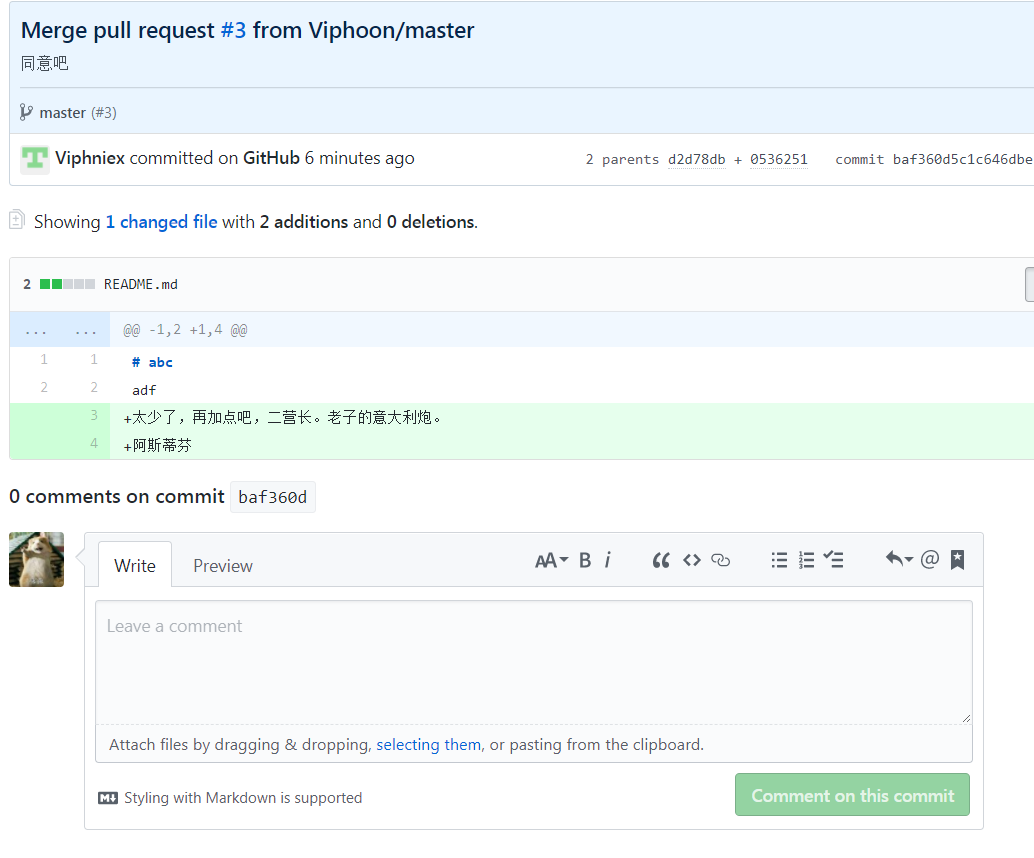
同意或者拒绝，都会给对方返回信息。



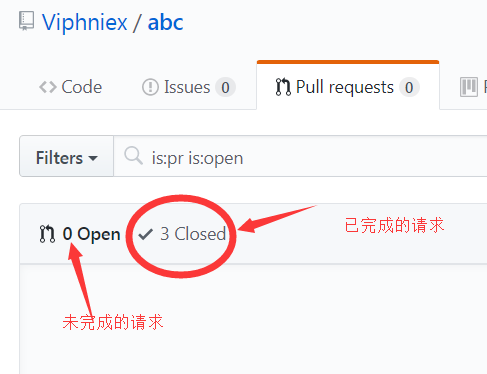
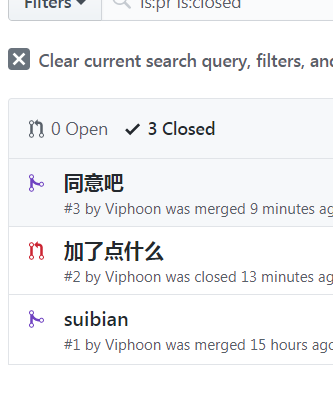
相同的方式，打开我们的仓库，可以看到对方对我们的请求所做的反馈。

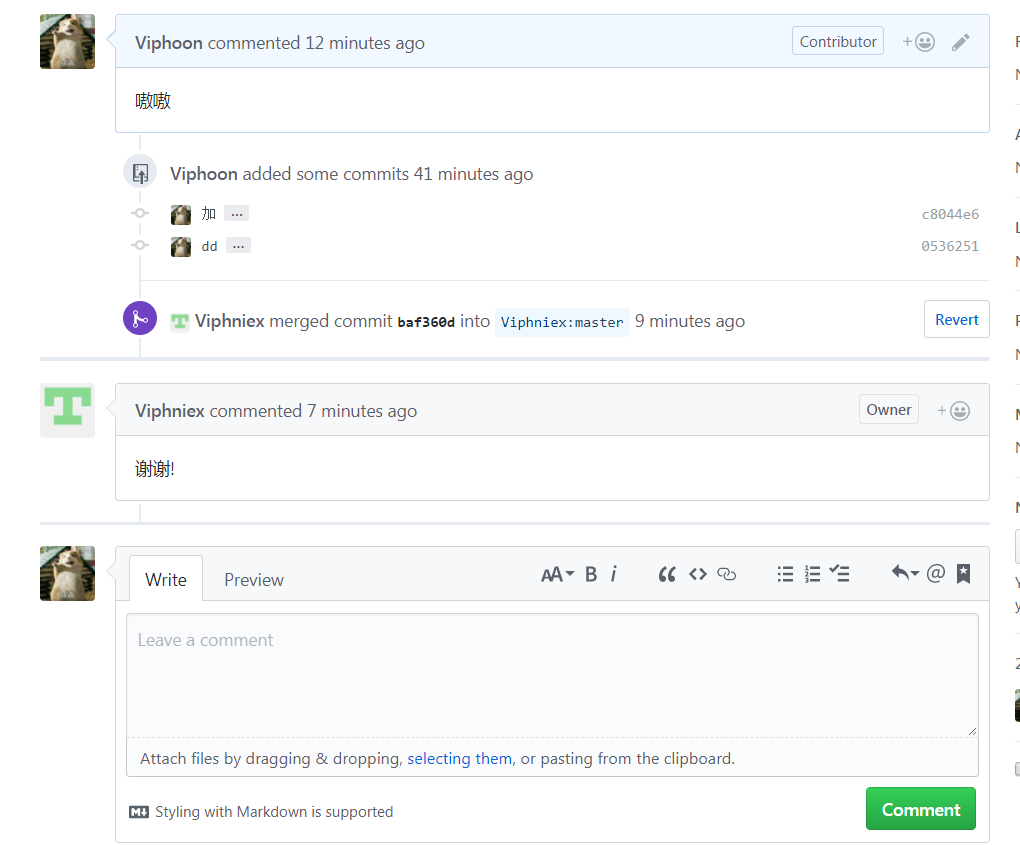


点击打开，，可以看到具体详情。



或者打开pull request，点击 closed查看。



至此，我们的一次完整的Fork模式操作完成了。

### 合作者模式

###### 创建仓库，寻找合作者。