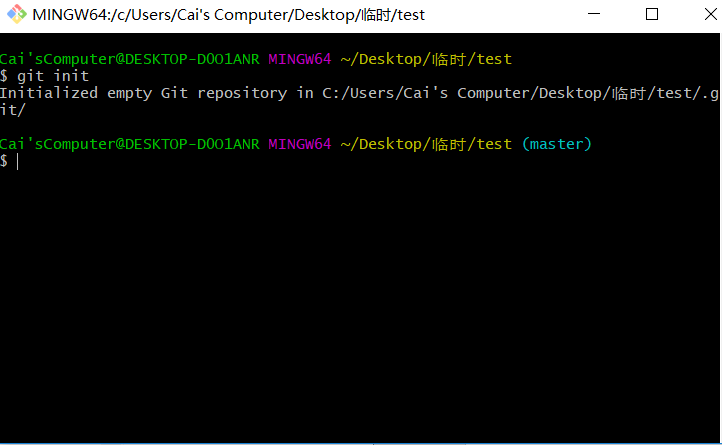
Git和GitHub的学习

1. Git windows应用：
2. 右键Git bash here
3. 输入命令Git init：初始化一个本地仓库，让git管理该目录



附：Git status：查看文件是否被接管，红色未被接管，绿色被接管（某个文件被修或者新添加文件都会变红）

1. Git add: 添加需要被托管的文件，即使使文件变绿（git add . ：托管所有文件，git add 文件名：托管某个文件）
2. Git commit: 将文件提交到本地仓库，即目前被git管理的目录（git commit -m “版本信息”）

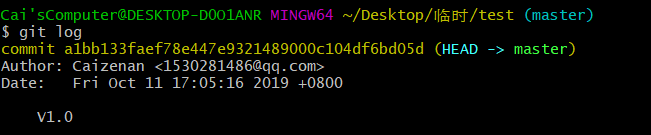
附：git commit 前需要告知git提交人信息：

git config –global user.email “[you@example.com](mailto:you@example.com)”

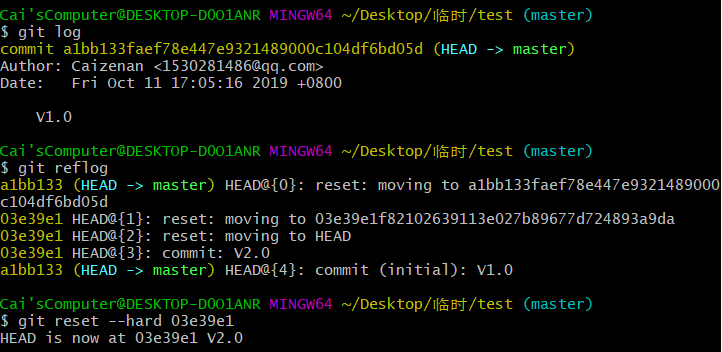
git config –global user.name “Your name”

（以上为git的账号邮箱以及用户名）

1. Git log: 查看commit信息



1. Git reset: 回滚操作(git reset –hard 版本ID)



附： commit 后一串即版本ID,用回滚操作返回某个版本后再用git log就没法查看到所有历史版本的ID,可用git reflog查看

黄色部分也是版本ID

1. Git checkout: 撤回修改：即在工作区内撤回，如修改了某个文件，用git status时被修改文件会显示未被git追踪，如果此时用git checkout – 文件名 来撤回操作
2. Git reset HEAD 文件名：由暂存区回到工作区,即由绿色变回红色
3. Git branch：创建分支，（Git branch 分支名），用Git branch查看master主分支和别的分支。（git checkout -b 分支名：创建并切换分支）

附：git branch -d 分支名：用于删除分支

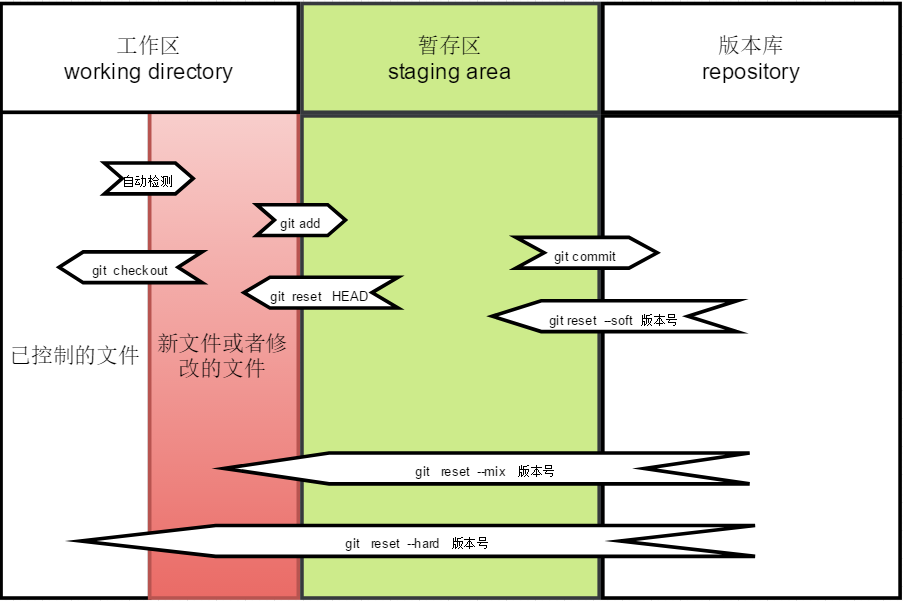
1. Git checkout 分支名：用于切换到别的分支下进行开发
2. Git merge：间分支合并到主分支，先git checkout master回到主分支，再git merge 分支名，将别的分支合并到主分支
3. 合并多个分支时出现冲突：比如不同版本更改了同一文件的同一行，合并到主分支时会出现冲突警告，此时git会把所有更改都增加到文件内，可以手动删掉不需要的更改。
4. 工作流：一般项目必须包括master分支用于储存绝对稳定的工程，创建Dev分支用于项目开发。
5. Github代码推送至远程仓库：
6. git remote add origin 仓库网址
7. git push -u origin 分支名

附：origin是仓库网址的别名，用于代替网址，push可以push各个分支

1. Git报错解决：fatal: remote origin already exists，执行git remote rm origin即可
2. Git克隆仓库：
3. 创建目录，打开终端，进入目录，git clone 远程仓库网址
4. Cd指令进入克隆到本地的仓库目录
5. git log即可看到各种版本，此时目录内包括所有分支
6. 如果需要切换分支，可用git checkout 分支名 进行切换
7. 本地仓库的更新：与克隆不一样，更新是因为本地仓库已经克隆了部分代码下来，此时只需要用git pull 远程仓库地址代号 分支名，即可把新的代码从远程仓库“拉”下来到本地仓库。（想拉代码到分支，最好用git fetch origin 分支名 再 git rebase origin/dev，若将master拉到分支进行合并时产生冲突，可以先手动解决冲突，在执行git add 产生冲突的文件，然后执行git rebase --continue）

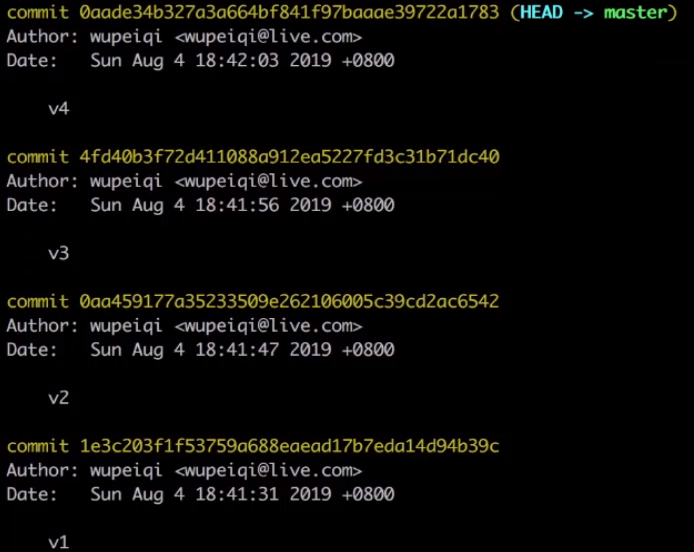






图片补充：从远程仓库到版本仓库：git fetch origin+git merge origin/分支名或git rebase origin/分支

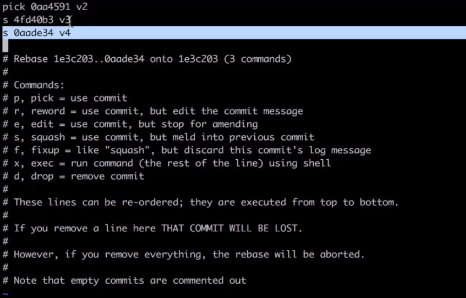
1. Git rebase应用场景1：提交版本记录合并：



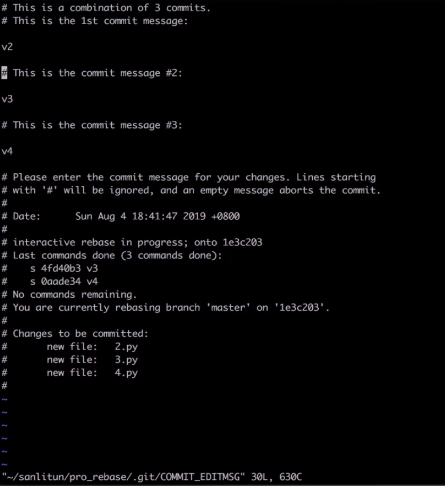
Git rebase -i 版本ID：从最新提交的版本到该版本ID合并到一起

Git rebase -i Head~数字N：从最新的提交的版本开始的前N个版本合并到一起

附:



手动将某个版本前的pick修改成s，即为s的开头的版本合并到上一个版本



保存上一个操作后弹出这个窗口，将#This is the commit message 都删掉，改成v2 & v3 & v4并保存退出即可将前几个合并

附：合并记录时最好把为提交到远程仓库的版本进行合并

1. Git rebase应用场景2:先切换到想合并分支，执行git rebase master，即把master分支所有代码添加到当前分支，然后切换回master分支，执行git merge,将该分支合并回master,以后，改分支就没了
2. Git rebase应用场景3:见17点
3. Git log –graph:用于查看分支，以图形方式查看

附：git log –graph –pretty=format:”%h,%s”，将上一条指令简化输出，%h为哈希值，即版本ID,%s为提交记录

1. 快速解决合并冲突：beyondcompare

Git config –local merge.tool bc3:将merge.tool改名为bc3

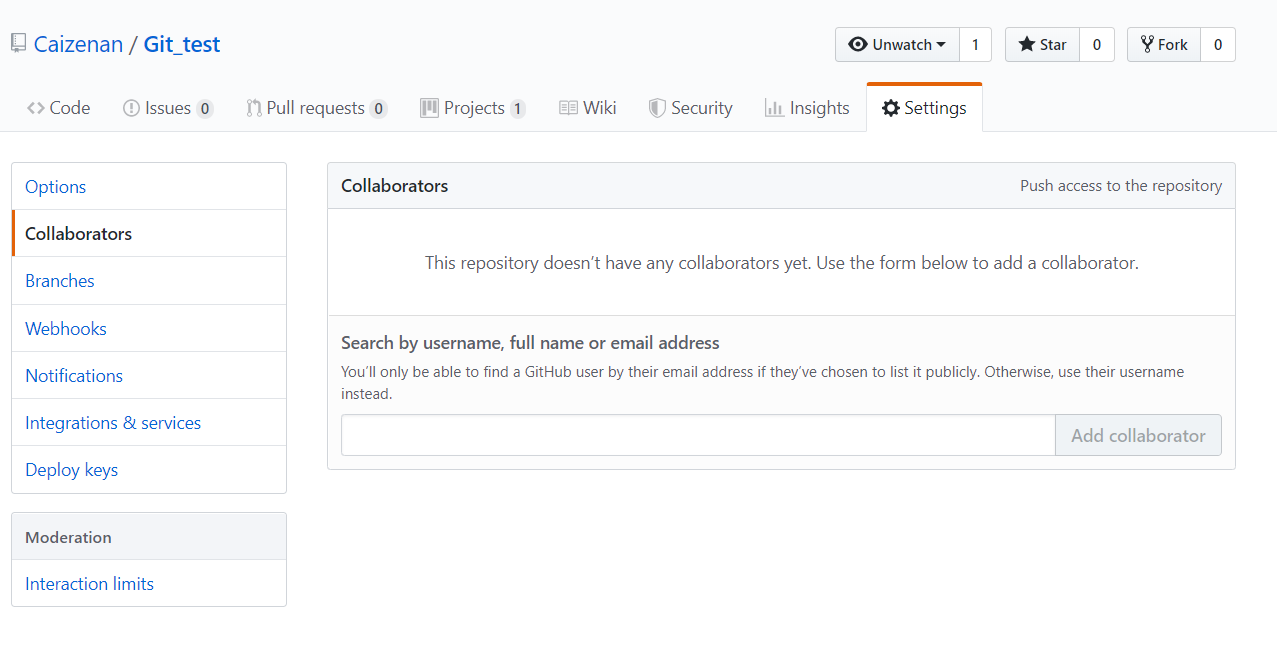
Git config –local mergetool.path ‘安装路径’:告诉git,beyondcompare安装路径

Git config –local mergetool.keepBackup false：解决冲突前不保留原文件

附：这样的配置只对git init的工作区有效，因为—local

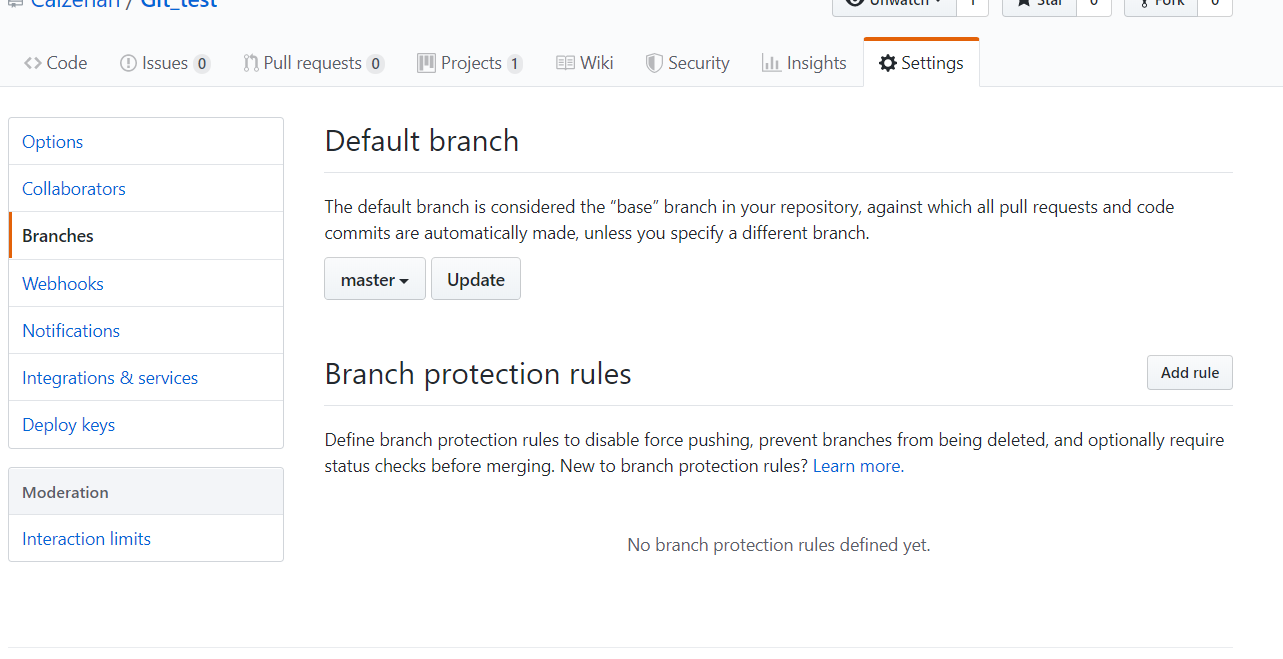
1. 创建项目合作开发：

1.创建项目并邀请：

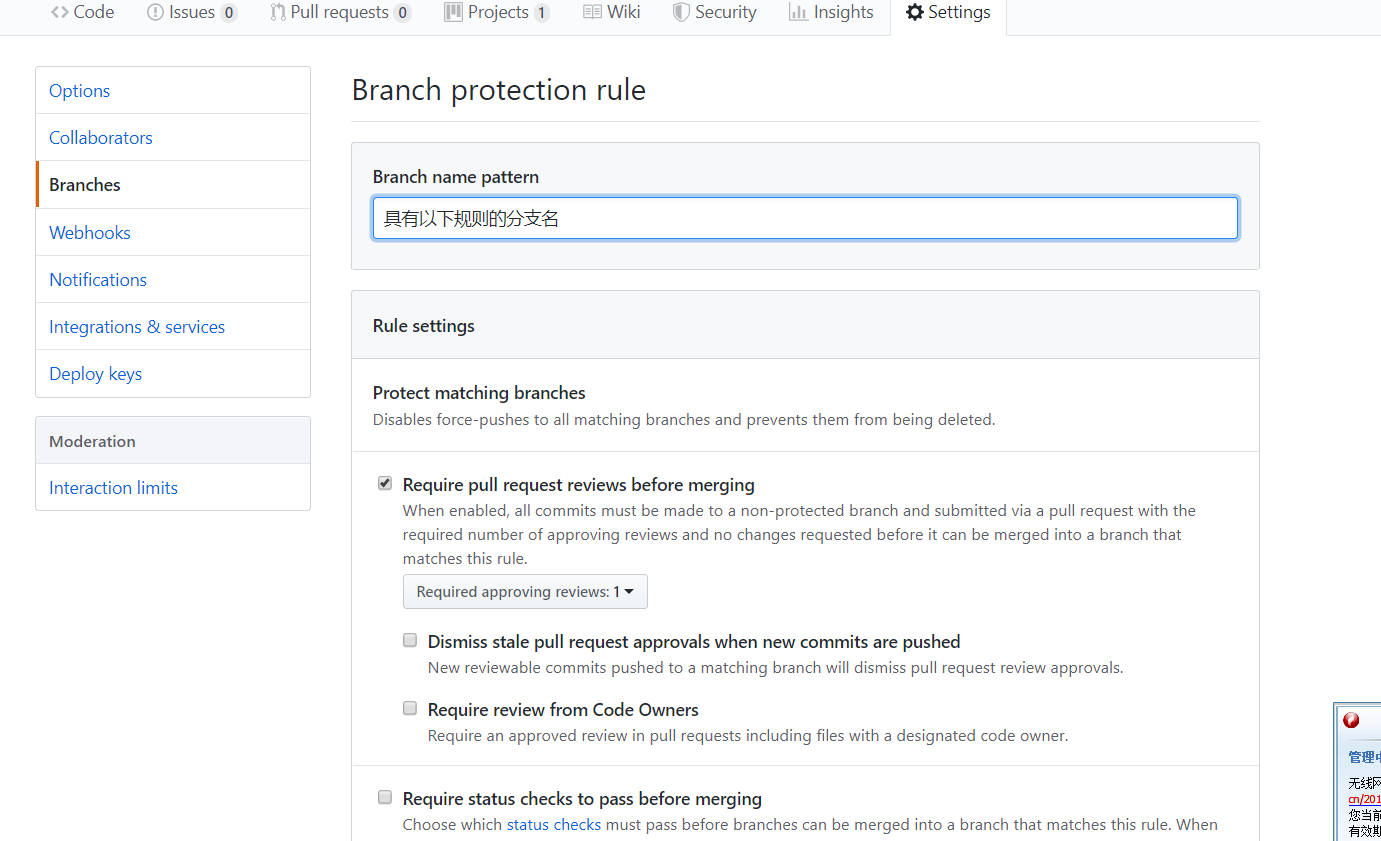


2.git clone项目到各自目录进行开发

3.其他人代码开发完成进行review:



附：为分支添加规则：Add rule



附：改分支的提交需要先完成review, Required approving reviews意思是需要几个人来完成review，默认是一个人

1. github创建仓库，本地git 库推送至远端出错：正确操作顺序，在github上创建仓库，再clone到本地目录，在该目录下打开git，git add . git commit…后再push到git仓库，我的错误操作是github上创建仓库并创建了readme文件，再在本地git目录下创建文件并尝试推送到github仓库，导致两个仓库版本不一致

! [rejected] master -> master (fetch first)问题的解决方案

今天在做git push时出现了如下错误：

分析原因，基本上可以确定是因为github上的远程库与本地库版本不一致（我对github上的文件做了编辑操作，且未更新到本地，当然也可能还有其他原因…），通过一番研究，找到了两种解决方案：

温柔型方案：

通过git pull 先将本地库更新到与远程库一致的版本，但要注意本地库后来做的修改可能被覆盖，最好使用git fetch(不会自动合并)，查看更新情况再有选择合并，或者先将本地库修改过的文件备份，git pull后再重新修改；

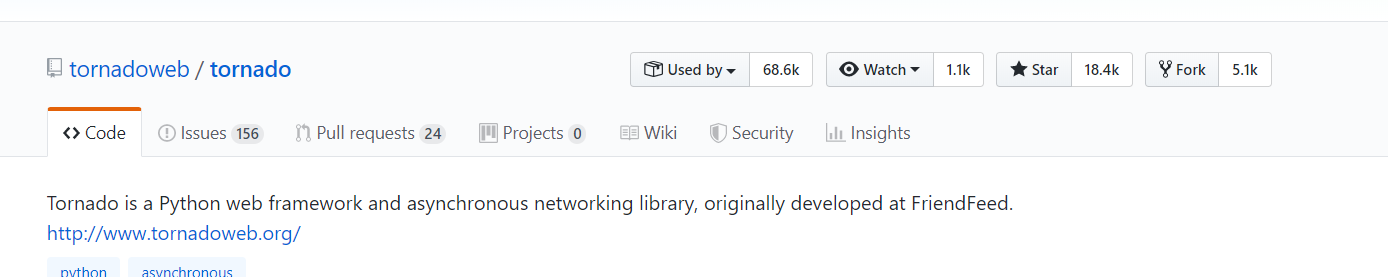
再运行git push即可成功。

暴力型方案：

git提供了一种强制上传的方式：git push -f ，它会忽略版本不一致等问题，强制将本地库上传的远程库，但是一定要谨慎使用，因为-f会用本地库覆盖掉远程库，如果远程库上有重要更新，或者有其他同伴做的修改，也都会被覆盖，所以一定要在确定无严重后果的前提下使用此操作。

25.给开源显目贡献代码：

1.找到一个开源项目：找到开源项目并fork



附：即把开源项目复制到自己的github账号下

1. 在自己的仓库进行修改：创建目录，clone下代码，修改，git add . ，git commit…等操作提交到自己的仓库
2. 给源代码作者提交修复申请：在自己复制的开源项目仓库下选择pull new request,create new request…

26. Git配置文件:

1.项目配置文件/.git/config: 在每个git项目仓库里，都会有 .git/config配置文件：用git config –local即可对配置文件进行修改，即在该项目仓库里生效

如git config –local user.name ‘yourname’ git config –local user.email ‘youremail’

2.全局配置文件:~/.gitconfig: git config –global user.name ‘…’… 对所有的项目都有效

3.系统配置文件：/etc/.gitconfig: git config –system user.name…: 需要有root权限才能修改

4.应用场景：



27.免密登陆：

1.修改方式1：



附：用第三条语句，或直接修改配置，文件的url为第三条即可

2.修改方式2：



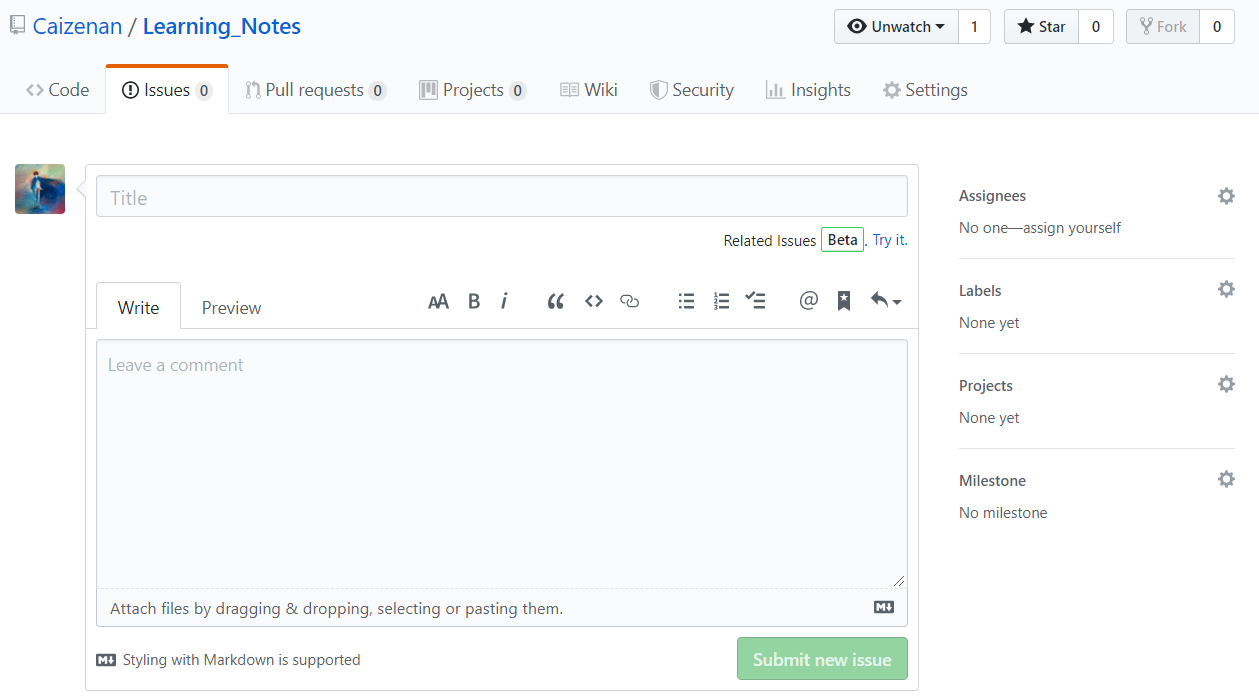
附：有待学习

28.git忽略文件：让git不再管理当前目录的某些文件:

附： Gitignore: 待了解

29.git issues 和 wiki:

1.issues : 可以用于团队间的提问与回复：



附：assignees:指派给某人回复，labels为问题类型…

2.wiki: 用于描述项目，便于交接项目

30. 仓库和project的区别：

一、概述

        Github上边的repositories翻译为代码仓库，可以保存多个代码工程和项目的代码，资源，文本、图片......等;

        而projects可以翻译为项目板，是project-boards的简写。简单说，可以理解为工作计划表之类的书签，制定一下工作计划，Bug，流程什么的。

        这么看，这很反直觉哇，我还以为projects也是项目什么的，没想到和我印象中的差得很远。。。

二、官方说明

        repositories：仓库就像项目的文件夹。 项目的仓库包含项目的所有文件，并存储每个文件的修订记录。 您也可以在仓库中讨论并管理项目的工作。您可以个人拥有仓库，也可以与组织中的其他人共享仓库的所有权。

        projects：GitHub 上的项目板帮助您组织工作和排列工作的优先级。 您可以为特定功能工作、全面的路线图甚至发布检查列表创建项目板。 通过项目板可以灵活地创建适合需求的自定义工作流程。项目板包括议题、拉取请求和注释，在选择的列中分类为卡片。 您可以拖放或使用键盘快捷键对列中的卡片重新排序，在不同列之间移动卡片，以及更改列的顺序。

1. 三大区:



**1.为什么要先执行 git add然后 git commit？**

很多 Git 的初级教程，几乎都有说先执行 git add ，然后 git commit。

那么为什么要先add然后commit呢？

git commit执行时，会提交暂存区的内容；  
git add 命令会将我们做的修改添加到暂存区中。  
这就是为什么 git commit 之前要先执行 git add 的原因,如果不先执行add，那么直接执行commit时不会把当前的修改内容提交到代码库中的。

**2.工作区、暂存区、版本库**

Git 本地数据管理，大概可以分为三个区，工作区,暂存区和版本库。

* 工作区（Working Directory）  
  是直接编辑的地方，肉眼可见，直接操作。
* 暂存区（Stage 或 Index）  
  数据暂时存放的区域。
* 版本库（commit History）  
  存放已经提交的数据，push 的时候，就是把这个区的数据 push 到远程git仓库了。

下图各区之间的数据传递流程示意图。

Git三大分区.png

git对比命令

git diff 工作区 vs 暂存区

git diff head 工作区 vs 版本库

git diff –cached 暂存区 vs 版本库

1.刚开始，什么操作都没有，三个区的数据是一致的，执行 git diff 命令都为空  
2.编辑文件增加代码，现在工作区内容发生变化，暂存区和版本库内容不变。  
3.执行git add 操作后，修改同步到暂存区，现在工作区和暂存区数据一致。  
4.执行git commit 操作后，修改已经同步到版本库，三区数据再次保持一致。

**3.三大分区的好处**

通过 checkout/stash/reset 等命令，通过不同的参数搭配使用，可以在工作区，暂存区和版本库之间，轻松进行数据的来回切换。

修改了多个文件，在不放弃任何修改的情况下，其中一个文件不想提交，如何操作？

（没add操作 : git add 想提交的文件

已经add: git reset –soft 将文件从暂存区回滚到工作区，重新add）

修改到一半的文件，突然间不需要或者放弃修改了，怎么恢复未修改前文件？

(git checkout)

代码写一半，被打断去做其他功能开发，未完成代码保存？

(git stash 提交到缓存区)

代码写一半，发现忘记切换分支了？

(git stash & git checkout)

代码需要回滚了？（git reset）

命令 作用

git reset –soft commit\_xxid

暂存区->工作区 将commit\_xxid提交从暂存区回滚到工作区

git reset –mixed commit\_xxid 版本库->暂存区

将commit\_xxid提交从版本库回滚到暂存区

git reset –hard commit\_xxid

版本库->暂存区->工作区 三个区都清除commit\_xxid提交