**Common02\_git介绍及使用**

**首汽租车版权所有**

**内部资料注意保密**

**修订记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订人** | **修订日期** | **修订描述** |
| 1.0 | 叶衡 | 2018-10-12 | 初始版本 |
|  |  |  |  |
|  |  |  |  |

**目录**

[1 文档说明 - 1 -](#_Toc527489879)

[2 GUI和CLI - 2 -](#_Toc527489880)

[2.1 GUI vs CLI - 2 -](#_Toc527489881)

[2.2 通过GUI和CLI操作GIT的区别 - 2 -](#_Toc527489882)

[2.3 shell - 3 -](#_Toc527489883)

[3 Git的工作原理 - 4 -](#_Toc527489884)

[3.1 名词解释 - 4 -](#_Toc527489885)

[3.2 工作区、暂存区、本地仓库之间的关系 - 5 -](#_Toc527489886)

[4 Git的基本使用 - 6 -](#_Toc527489887)

[4.1 配置用户信息 - 6 -](#_Toc527489888)

[4.2 初始化仓库 - 7 -](#_Toc527489889)

[4.3 基本操作 - 8 -](#_Toc527489890)

[5 常用命令 - 13 -](#_Toc527489891)

[5.1 新建版本库 - 13 -](#_Toc527489892)

[5.2 增加/删除文件 - 13 -](#_Toc527489893)

[5.3 代码提交 - 13 -](#_Toc527489894)

[5.4 撤销操作 - 14 -](#_Toc527489895)

[6 分支操作 - 15 -](#_Toc527489896)

[6.1 Git分支 - 15 -](#_Toc527489897)

[6.2 分支操作 - 15 -](#_Toc527489898)

[6.3 分支常用操作 - 19 -](#_Toc527489899)

[7 远程(GitHub)操作 - 21 -](#_Toc527489900)

[7.1 已有远程仓库 - 21 -](#_Toc527489901)

[7.2 已有本地仓库 - 24 -](#_Toc527489902)

[7.3 远程操作常用命令 - 30 -](#_Toc527489903)

[8 解决冲突 - 32 -](#_Toc527489904)

[9 多人协作(GitHub) - 36 -](#_Toc527489905)

[9.1 Pull Request - 36 -](#_Toc527489906)

[9.2 Collaborators - 40 -](#_Toc527489907)

[10 附录 - 43 -](#_Toc527489908)

# 文档说明

本文档用于指导首汽租车技术人员通过Linux命令使用GIT代码管理和多人协作。

其他GIT知识及版本控制概念可详细阅读首汽租车技术文档《common03\_GIT》

# GUI和CLI

学习及使用git，到最后必然要进行操作。和svn不同的是，使用git将会有两个选择，GUI和CLI 。

## GUI vs CLI

什么是GUI和CLI？

图形用户界面（Graphical User Interface，简称 GUI，又称图形用户接口）是指采用图形方式显示的计算机操作用户界面。通俗的讲就是使用鼠标点点点来进行操作。

命令行界面（英语：command-line interface，缩写：CLI）是在图形用户界面得到普及之前使用最为广泛的用户界面，它通常不支持鼠标，用户通过键盘输入指令，计算机接收到指令后，予以执行。也有人称之为字符用户界面（CUI）。

## 通过GUI和CLI操作GIT的区别

目前流行的git GUI有GitHub Desktop，TortoiseGit，SourceTree，GitKraken等等，它们的风格各不相同，但是都为我们提供了相对健全的功能。因此，选择使用哪一个，视具体情况而定。

另外，目前流行的IDE都集成了git插件和功能，这大大的增加了工作的高效，但是这些IDE所集成的Git功能都比较初级。

使用GUI的好处就是GUI直观，并且更容易排查问题，操作更容易实现连续性；查看版本分支图，提交所有文件，合并和查看差异。缺点是不具备某些CLI所拥有的功能或者在GUI菜单里难以找见。

使用CLI进行版本控制带来的好处就是高效，如果出错，会跟你提供很详细的信息，从而排查错误。功能全面，能几乎解决所有误操作或代码找回。

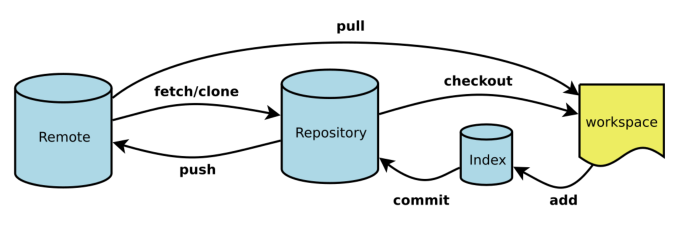
当然，在工作中无论使用哪一种方法都无所谓，但是如果是前期学习使用git，则建议从CLI操作开始，因为git源于命令行。这有助于理解和使用git。

## shell

学习使用CLI来进行git版本控制，则必须使用命令行来完成，目前常见的shell有以下几种：DOS，cmd，Powershell，Terminal，Bash等等。

# Git的工作原理

Git和其他版本控制系统如SVN的一个不同之处就是有暂存区的概念。



**图：Git工作示意图**

## 名词解释

Workspace：工作区，就是在你电脑里能看到的目录，用来直接工作修改的文件。

Index：暂存区，数据暂时存放的区域，可在工作区和本地仓库之间进行数据的友好交流。

Repository：本地仓库，存放已经提交的数据，push 的时候，就是把这个区的数据 push 到远程仓库了。

Remote：远程仓库。

## 工作区、暂存区、本地仓库之间的关系

暂存区将工作区和本地仓库操作之间做了一个缓冲。

即可以对代码进行版本的管理如回溯等等，又避免了多次琐碎的commit提交。

工作区的文件是不能直接提交到本地仓库的，需要先提交到暂存区。只有本地仓库的文件才可以推送到远程仓库。

# Git的基本使用

## 配置用户信息

在使用git之前，需要先进行全局配置，否则，当提交代码到远程仓库的时候，会提示please tell me who you are，直到配置了个人信息后才能继续操作。

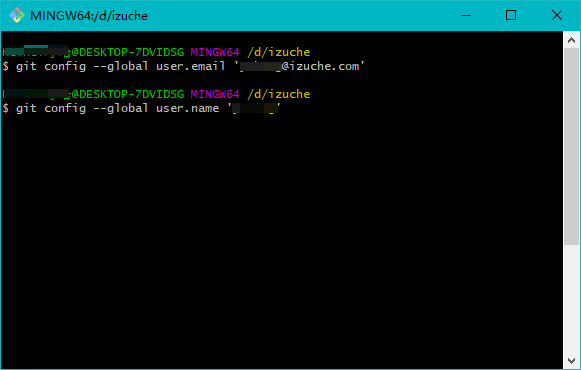
写在前面，git自带的git bash窗口可以在桌面或目录中鼠标右键打开，并默认路径是当前所在目录或窗口路径。

配置很简单，只需要执行以下两行命令

$ git config ‐‐global user.email "你的邮箱"

$ git config ‐‐global user.name "你的名字"

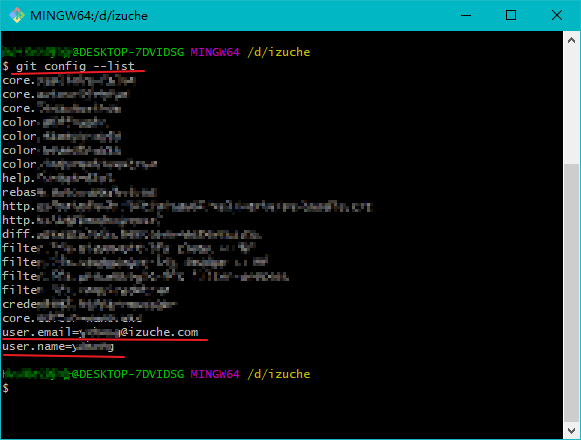
如果你想要修改，则重新执行此命令



**图：配置用户和邮箱**

查看配置信息，执行以下命令

$ git config –-list



**图：配置信息**

另外，配置以下信息，防止gitk图形查看器中文乱码

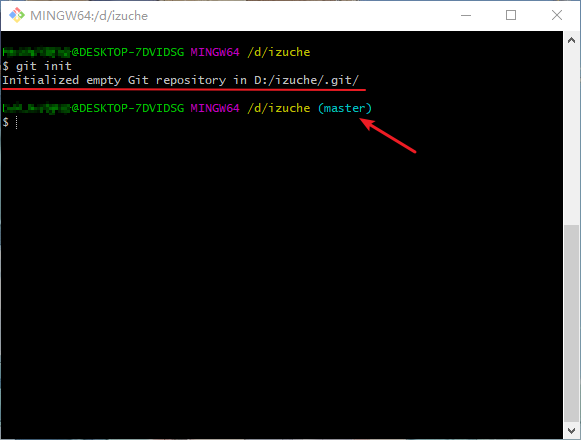
$ git config ‐‐global gui.encoding utf‐8

gitk是git自带的GUI查看器

当配置完基本信息后，便可以开始使用git进行版本管理操作了。

## 初始化仓库

通过使用$ git init命令把这个目录变成Git可以管理的仓库。



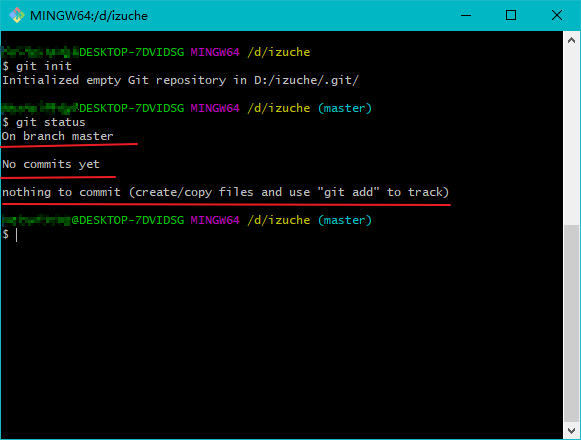
**图：初始化仓库**

以上为初始化了一个空的git仓库在D:/izuche目录下，并且默认生成了master主分支。

除此外，还生成了一个.git文件夹，这个文件夹一般是隐藏的，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

## 基本操作

首先使用 $ git status 查看版本库状态。

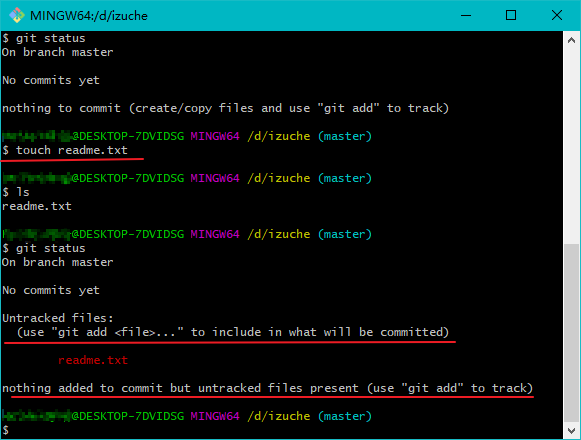


**图：查看版本库状态**

以上提示在分支master上，还没有被提交的文件。

Nothing to commit：没有任何文件被提交。并且提示让创建或复制文件使用$ git add来跟踪管理他们。

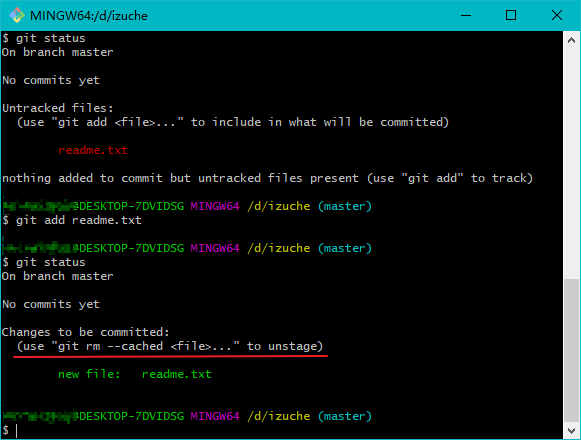
在GIt管理的文件目录下新建示例文件redeme.txt，然后再执行$ git status 进行查看。



**图：查看文件状态**

提示信息表示，在master分支上，还没有文件被提交。Untracked files：未被跟踪的文件。提示使用$ git add提交被包含的文件或目录。

提交文件或目录至暂存区，并且使用$ git status进行查看。



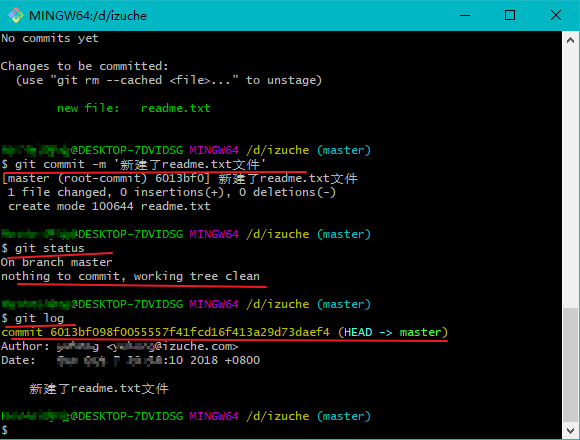
**图：提交文件或目录至暂存区**

Changes to be committed：修改将要被提交。此时文件已经被提交到了暂存区，同时提示使用$ git rm –cached可以将暂存区的文件拉回到工作区。如果提交了错误的文件到了暂存区，可以执行此操作进行修改。

现在已将相应的文件提交到了暂存区，暂存区的文件还不是真正被管理和保存，接着需要将暂存区的文件或目录提交到本地仓库中。

执行$ git commit –m ‘提交日志/注释’ –m的全拼是 - –message，执行完后接着执行$ git status查看暂存区状态，当有文件被提交到了本地仓库，可接着执行$ git log 来查看提交信息。

如果之前没有配置过基本信息，则此处执行完提交命令后会提示提交失败，需要配置基本信息才能继续操作。



**图：提交文件或目录至本地仓区**

以上提示信息表示，执行在工作区的文件或目录已经全部被提交。

在执行了$ git log后，会有一长串文字信息，表示的是此次提交的版本号，将来可以使用此进行版本回滚。将相应的版本拉回到暂存区或工作区。并且此次提交还会显示提交人和提交时间。

# 常用命令

## 新建版本库

在当前目录新建一个GIt仓库

$ git init

新建一个目录，将其初始化为Git仓库

$ git init [project-name]

下载一个项目和它的整个代码历史

$ git clone [url]

## 增加/删除文件

添加指定文件到暂存区

$ git add [file1] [file2] …

添加指定目录到暂存区，包括子目录

$ git add [dir]

添加当前目录的所有文件到暂存区

$ git add .

## 代码提交

提交暂存区到仓库区

$ git commit –m [message]

提交暂存区的指定文件到仓库

$ git commit [file1] [file2] … -m [message]

提交工作区自上次commit之后的变化，直接到仓库区

$ git commit –a –m [message]

## 撤销操作

恢复暂存区的指定文件到工作区

$ git checkout [file]

恢复某个commit的指定文件到暂存区和工作区

$ git checkout [commit] [file]

重置暂存区的指定文件，与上一次commit（最近一次历史记录）保持一致，但工作区不变

$ git reset [file]

# 分支操作

## Git分支

分支( branches ) 是指在开发主线中分离出来，做进一步开发而不影响到原来主线。

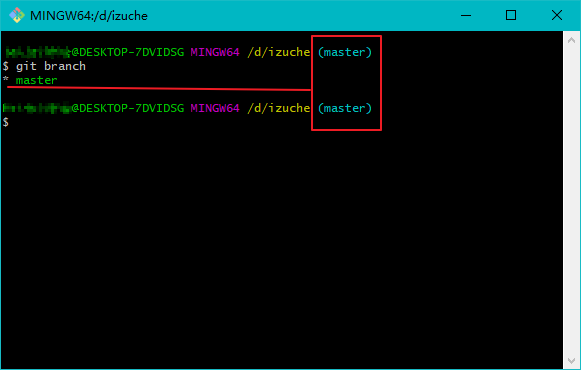
Git 存储的不是一系列的更改集( changeset )，而是一系列快照。当你执行一次 commit 时， Git 存储一个 commit 对象，它包含一个指针指向你当前需要提交的内容的快照。

Git 中的 master 分支的功能，和其他分支一样。master 在 git 项目中常见到，是因为 git init 命令运行时默认创建一个分支，并命名为 master。

## 分支操作

查看所有分支，工作区停留的分支

$ git branch

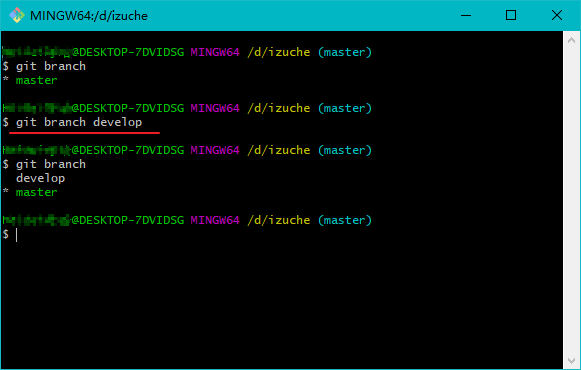


**图：查看分支信息**

上图表示当前有 mater分支，并且工作区停留在master分支。

创建一个分支

$ git branch [branch-name]

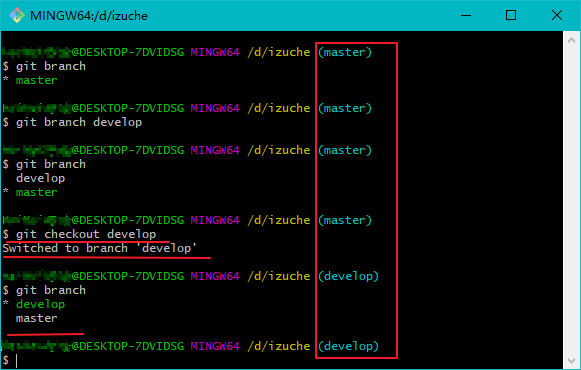


**图：创建新分支**

创建了develop分支，并且当前工作区还在master分支上。

切换分支

$ git checkout [branch-name]



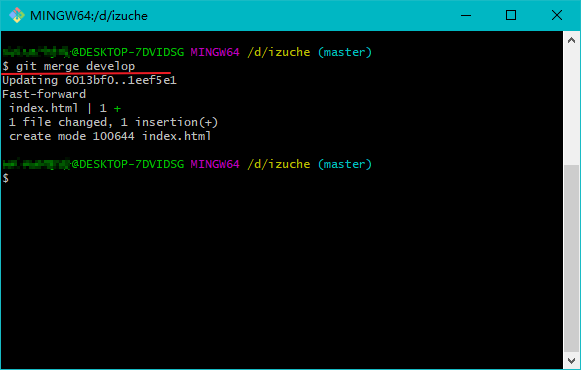
**图：切换分支**

上图表示将分支从master切换到develop分支，并且工作区停留在了develop分支。

合并指定分支到当前分支

$ git merge [branch-name]

合并之前，在develop分支上创建了示例文件index.html。



**图：合并分支**

上图表示将develop分支合并到master分支上，1个文件被更改，增加了1个文件，创建了index.html文件。

## 分支常用操作

列出所有本地分支

$ git branch

列出所有远程分支

$ git branch –r

新建一个分支并停留在当前分支

$ git branch [branch-name]

新建一个分支，并切换到该分支

$ git checkout –b [branch-name]

切换到指定分支

$ git checkout [branch-name]

合并指定分支到当前分支

$ git merge [branch-name]

删除指定分支

$ git branch –d [branch-name]

# 远程(GitHub)操作

使用Git管理的项目都是在本地存储，不便于多人协作；另外，如果本地有毁灭性质的数据丢失，带来的损失是非常巨大的。因此，需要将项目或代码提交到远程仓库。

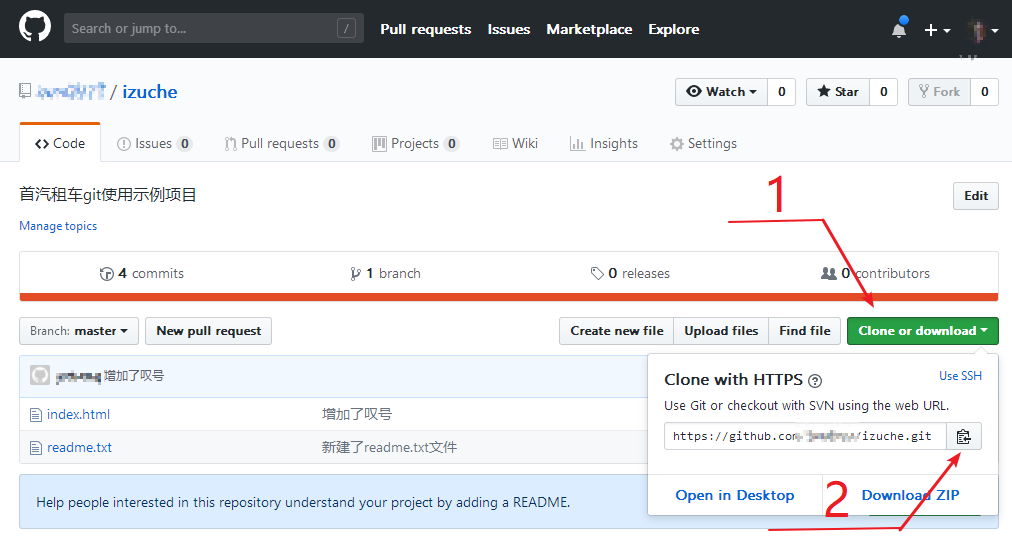
对git提供云存储服务的有很多，GitHub，bitbuccket，码云，coding等等。对于国内来说使用码云更加友好，服务器在国内，界面为中文，便于操作，并且功能相较于GitHub来说基本无区别。但是作为学习使用，建议使用GitHub。

## 已有远程仓库

已有远程仓库，远程的仓库已经有初始的文件，分支等信息，只需要下载到本地进行开发。

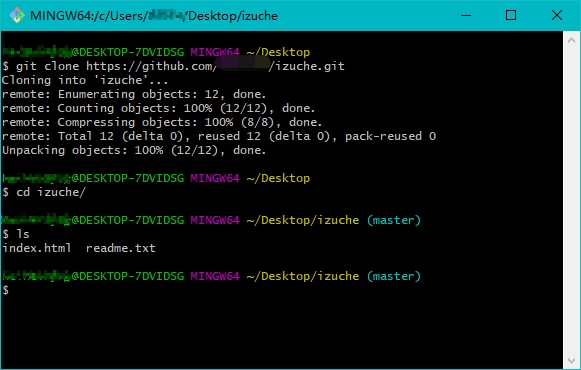
下载远程仓库到本地

$ git clone [url]



**图：远程仓库**

点击远程仓库绿色按钮clone or download，然后点击复制链接按钮，复制完成后，选择存放项目目录，在当前目录下打开命令行。

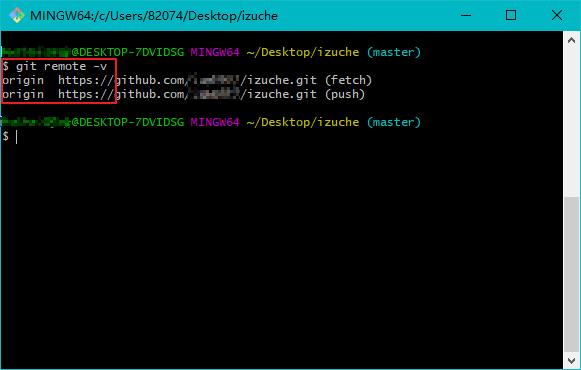


**图：下载远程仓库**

当远程仓库下载完成后，便可以在本地进行开发。

远程推送

当本地项目是从远程仓库下载的，GIT会自动添加一个名称为origin的远程仓库地址。这个地址指向执行clone命令的地址。

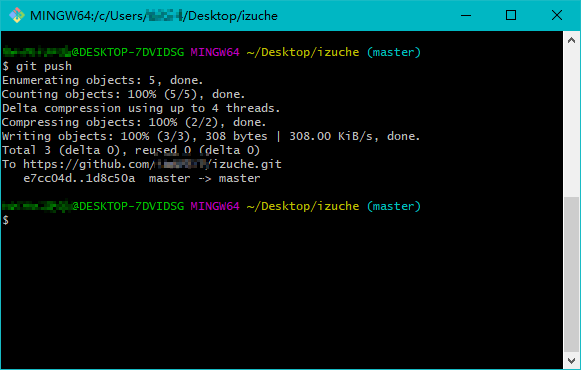


**图：查看远程仓库地址**

当向远程仓库推送项目时，无需设置额外配置，直接执行命令即可。

如果在推送时，弹出输入用户名和密码窗口时，输入后点击确认。

$ git push

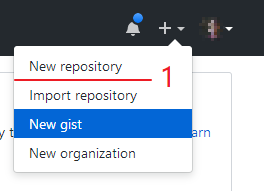


**图：推送到远程仓库**

## 已有本地仓库

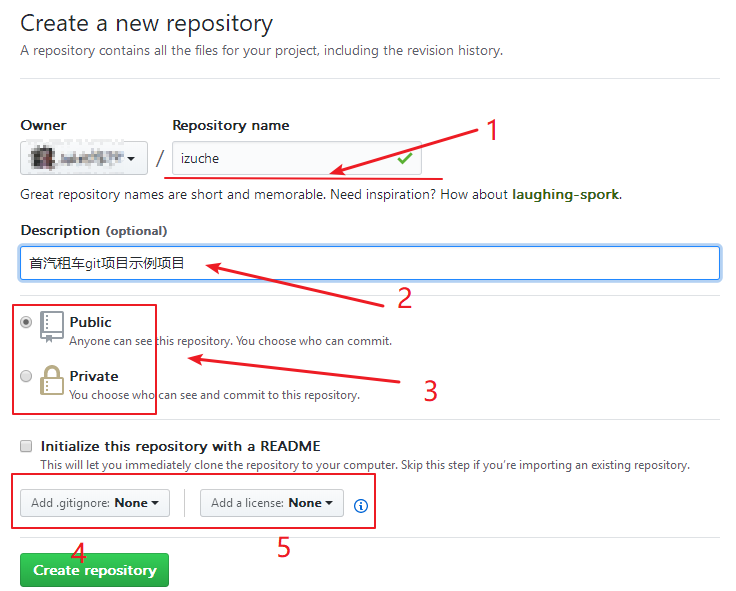
当已有本地仓库时，需要项目负责人或搭建项目的人在远程创建远程仓库。继续以GitHub为例，创建远程仓库。

以下示例将引导创建一个远程仓库。



**图：创建仓库1**

点击GitHub页面导航栏右侧的下拉菜单，点击New repository，跳转到创建页面。



**图：创建仓库2**

1、仓库名称，不能和已有仓库名称重复。

2、项目描述。

3、项目是否公开，public即公开，任何人都可以看和下载；private是创建私有仓库，只有相关人员才能看到和下载，但是收费。

Initialize this repository with a README，是否在仓库根目录下创建README文件。关于创建README文件的相关信息，在后面会提到，这里不勾选。

4、是否在仓库的根目录下添加一个.gitignore文件。

该文件是告诉git每次提交历史记录的时候，让git忽略跟踪某些文件或目录的配置。

比如使用npm来管理项目插件和资源包的项目，都会在根目录下有一个node\_modules文件夹，这个文件夹里往往会有上万个文件。当使用git来管理项目时，像这样的文件或目录是不需要被提交和管理，否则会使提交和下载变得异常缓慢，影响工作效率，但又不能手动去避开这类文件或目录。所以，将这类文件或目录写入. gitignore文件中，便可以让git默认忽略这些文件或目录。

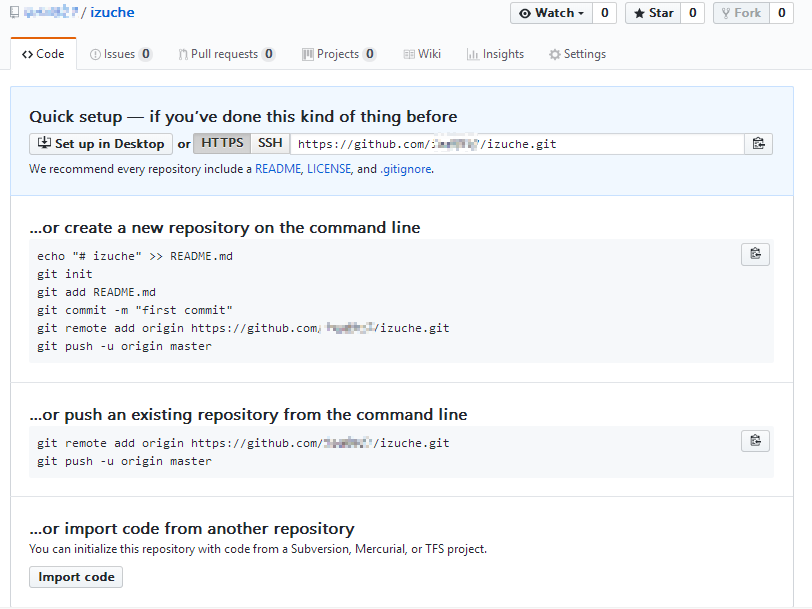
这里选择默认选项，不创建。一般都会在本地仓库中创建好，当向远程仓库推送时一并添加。

5、开源协议，目前的开源协议有上百种，这里选择默认。

公司的项目或文档属于内部保护文件，非开源项目。

如果想了解什么是开源协议或者如何为开源项目选择开源协议，可查看相关资料:如何为你的开源项目选择一个合适的开源协议？（<https://www.oschina.net/news/74999/how-to-choose-a-license>）

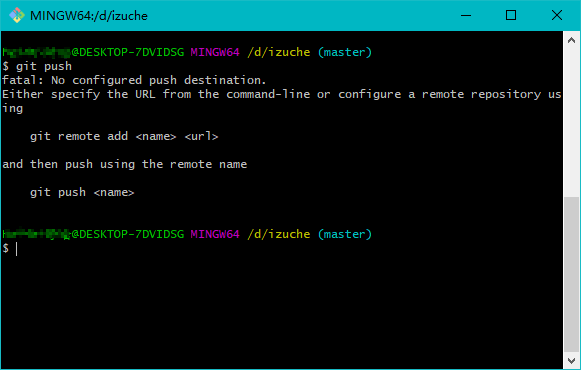
完成以上步骤，便可以点击create repository按钮，来创建一个空的远程仓库。



**图：创建仓库3**

当创建完远程仓库后，便可将搭建或创建的本地项目推送到该仓库。

和已有远程仓库不同的是，本地仓库推送不能直接执行$ git push

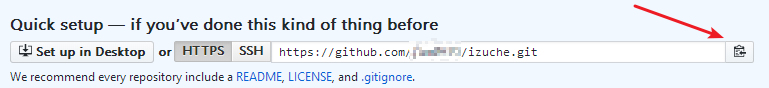


**图：没有远程仓库地址**

根据提示，需要先设置远程仓库地址

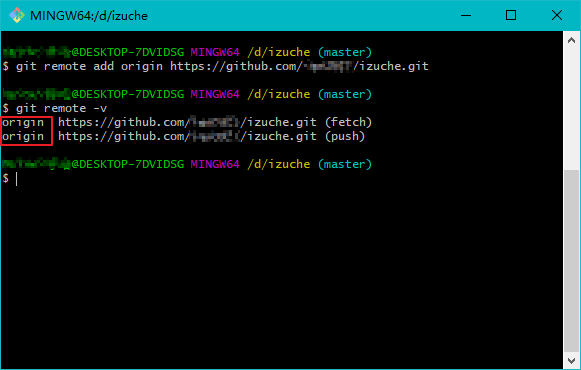
$ git remote add [name] [url]

name为远程仓库别名，名称随意，但最好是有一定的意义，这里取origin为例。url则为远程仓库地址。



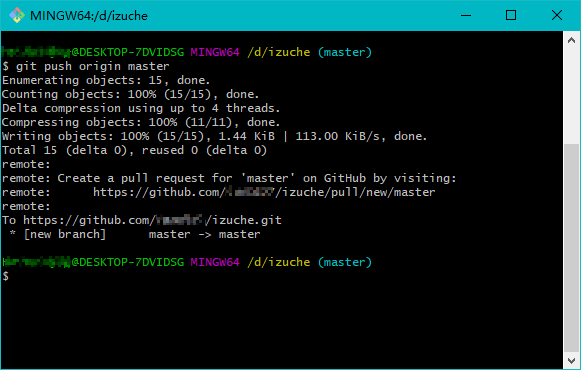
**图：远程仓库地址**

选择HTTPS对应的远程仓库地址，点击复制按钮



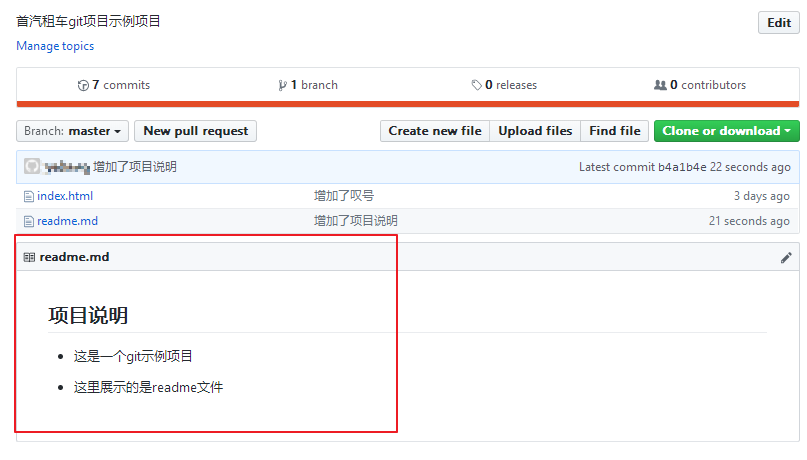
**图：设置仓库地址**

当设置完成后，使用$ git push [name] [branch-name]便可推送本地仓库到远程仓库



**图：远程仓库**

当完成后，刷新远程仓库页面，查看远程仓库。



**图：远程仓库**

这里显示本地仓库已经被成功推送到远程仓库，但是发现红色区域多了一些内容，显示的是readme.md文件内的内容。接前面所说，创建仓库时的readme文件就是在这里显示的，用来展示项目信息。另外，GitHub会将md文件自动渲染成html显示到页面。

## 远程操作常用命令

下载远程仓库的所有变动

$ git fetch [remote]

显示所有远程仓库

$ git remote ‐v

显示某个远程仓库信息

$ git remote show [remote]

增加一个新的远程仓库

$ git remote add [shortname] [url]

取回远程仓库的变化，并与本地分支合并

$ git pull [remote] [branch]

强行推送当前分支到远程仓库，即使有冲突

$ git push [remote] ‐‐force

上传本地指定分支到远程仓库

$ git push [remote] [branch]

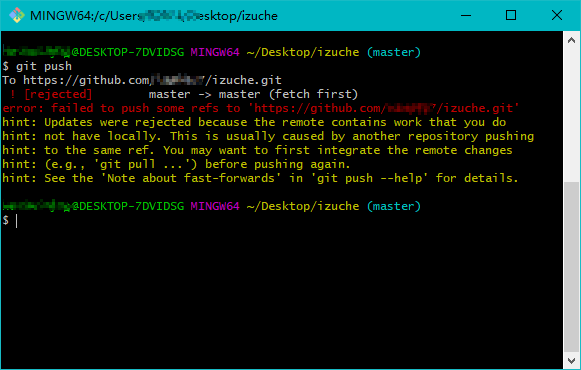
推送所有分支到远程仓库

$ git push [remote] ‐‐all

# 解决冲突

代码冲突是一个永远不能摆脱的问题。当合并分支或者提交到远程仓库的时候都会出现代码冲突。因此，解决代码冲突必然是重中之重。否则，上线后出现莫名其妙的bug，所造成的损失是无法预知的。

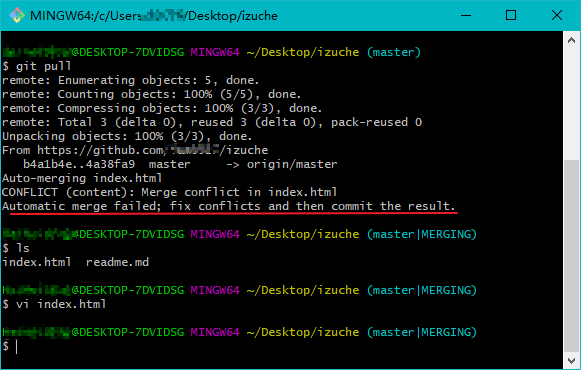
此处，所演示的是远程仓库版本超前本地仓库版本，本地仓库直接向远程仓库做推送操作。



**图：产生冲突**

不出意料，本地仓库无法推送到远程仓库。

以上的意思是说：推送失败，远程仓库包含一个本地仓库没有的文件或目录，在推送前可以先执行 $ git pull

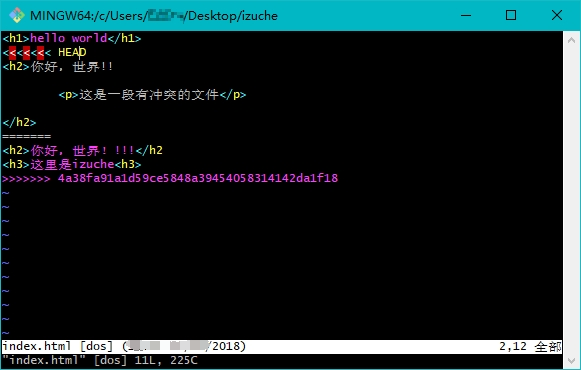


**图：拉去远程仓库**

此时显示的是远程的仓库已经拉取到本地，当git试图去合并的时候，本地和远程的index.html文件出现了冲突，git也不知道该怎么去合并，需要手动合并冲突。

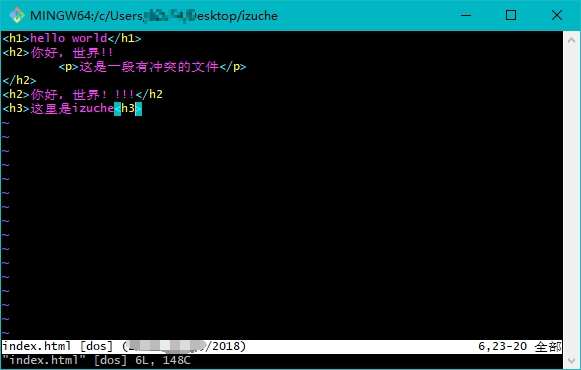
合并冲突有两种，一种是手动合并，一种是使用工具合并。这里介绍的是手动合并。

打开冲突文件，这里使用vi编辑器打开，也可以用其他的IDE打开。



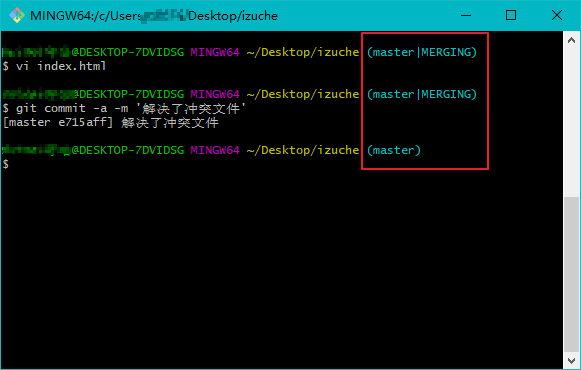
**图：查看冲突**

这里冲突的部分以等号（=）分割，HEAD表示的是本地的，下面带有commit ID的是远程最新的。这里合并取舍，取决于实际业务和情况。这里以所有的代码都有用做示例来合并。做法很简单，删除冲突文件的格式，还有不需要的代码。



**图：解决冲突**

保存后git便会自动合并。完成后需要再次执行$ git status 查看工作区状态。



**图：合并分支**

当然，手动合并有时候也不是一定成功，有时候也需要借助合并工具。这里也不做过多说明，需要自行学习和了解。

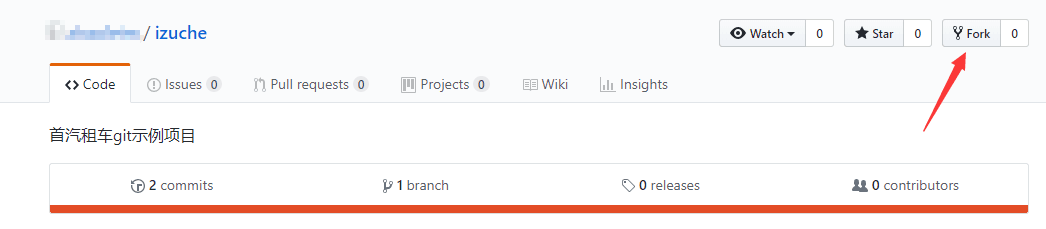
# 多人协作(GitHub)

该文档中，到现在只有一个GitHub账户在操作，但如果有另一个GitHub账户想推送代码呢？

正如实际工作和开发，一个项目不可能只有一个人。因此，便需要多人协作，一同开发和工作。但是，GitHub仓库默认只有仓库的拥有者或者管理员才有权限推送和提交代码，其他人是不能直接push代码的。

## Pull Request

不能推送代码到别人的仓库，但可以推送到自己仓库。可以先将别人的仓库复制一份到自己到账户下，使之变为自己的仓库，便可以将自己的代码推送到远程仓库中。



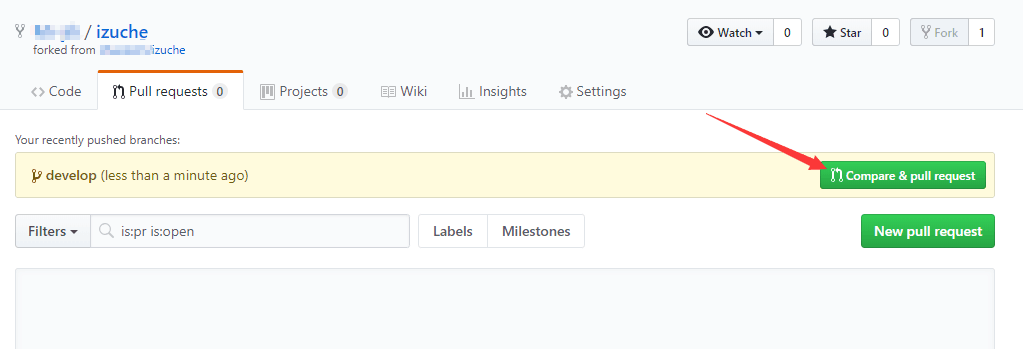
**图：fork仓库**

点击仓库右上角fork按钮，等待页面加载。

页面加载完成后，便可以看到fork到了自己的账户下，这时候后可以下载进行开发和操作，完成后向远程推送。

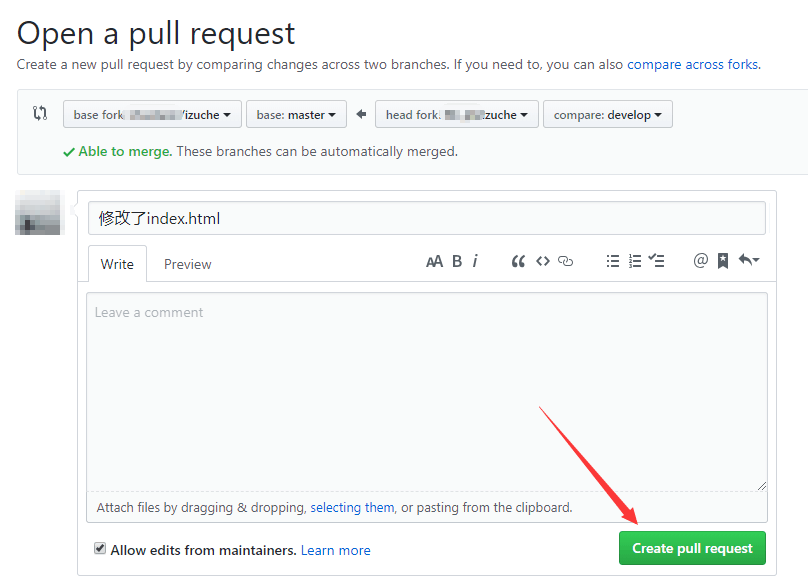
注意，在本地一定是基于新分支进行代码编写，推送也是只推送当前分支。

但是，推送后，也只是推送到了自己的仓库，还没有真正的推送到项目仓库，上图中可以看到导航栏有Pull request按钮。



**图：发起合并请求1**

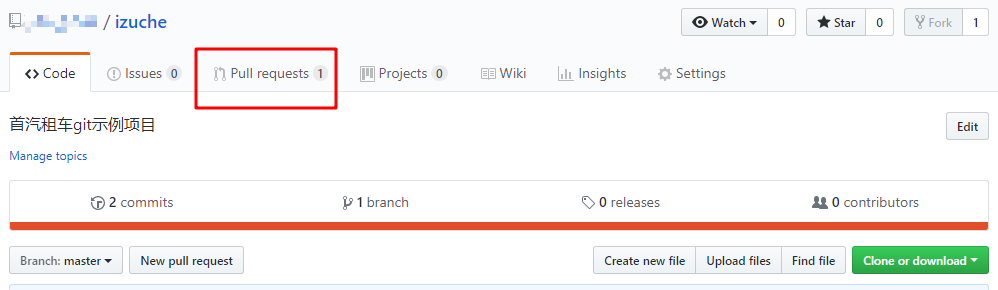
这时候点击compare&pull request发起Pull request请求，请求把自己仓库中的某个分支合并到原始仓库中。



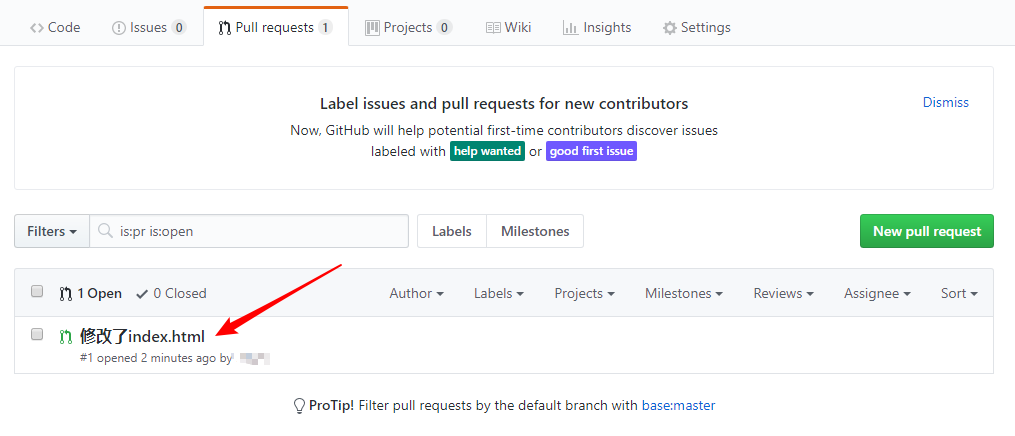
**图：发起合并请求2**

到这里，自己的工作就做完了。接下来就是原始仓库的拥有者或管理者的操作。因为自己对原始仓库没有任何的权限。

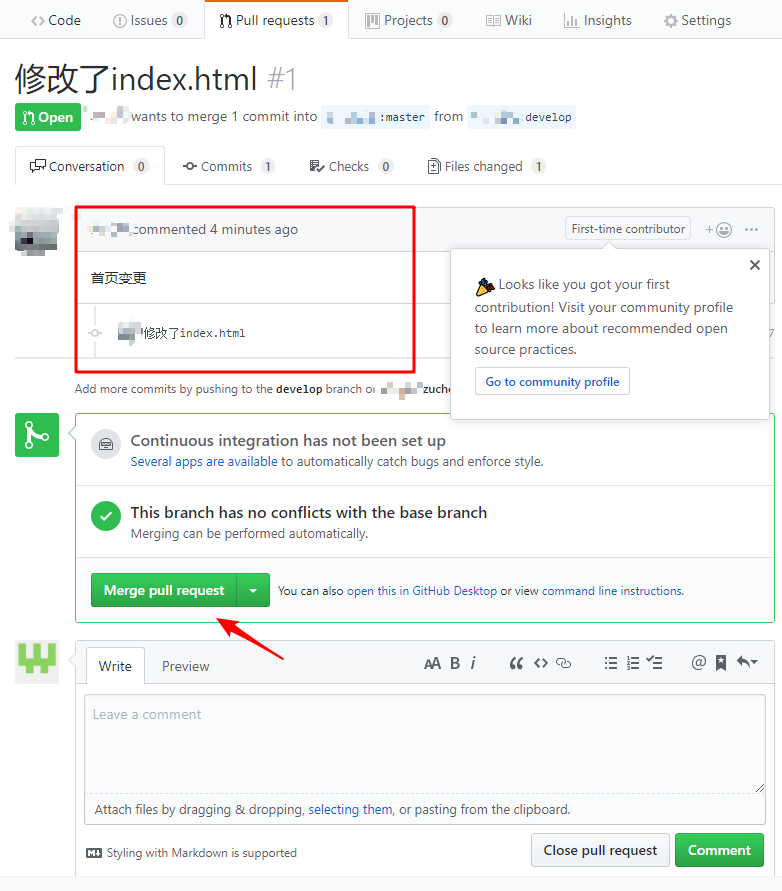
仓库的拥有者或管理者同样也可以看到Pull request按钮，并且多了来自其他GitHub用户的合并请求。



**图：查看合并请求**

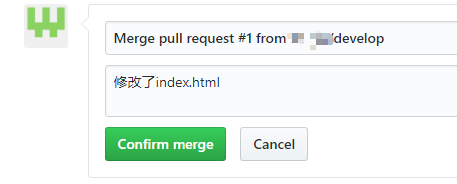
****

**图：查看合并请求详情**



**图：合并请求**

点击打开，可以看到请求合并的详细信息，如果确认没有问题，便可以点击Merge pull request按钮直接合并该分支代码到主仓库的主分支中。如果来自合并请求的代码有bug或其他重大问题，可以使用下面的回复评论功能，点击comment发送详细说明。

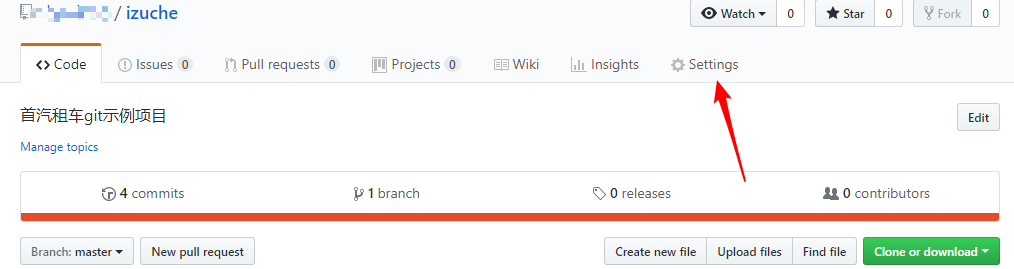


**图：确认合并请求**

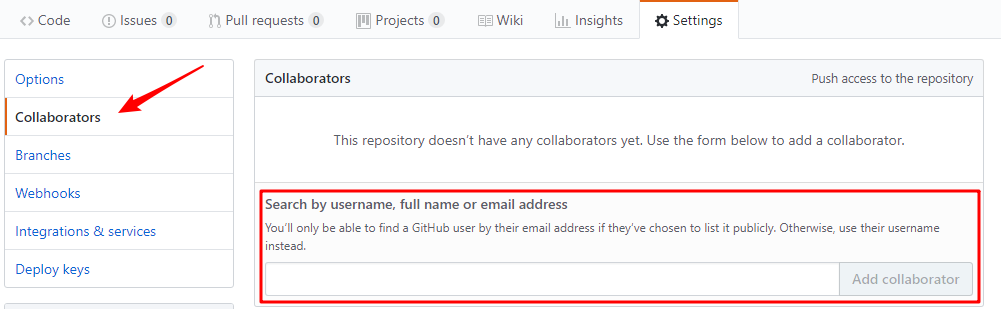
## Collaborators

Fork & Pull request的方式是可以实现多人开发，但是只有仓库的所有者或管理员才有权限，其他人需要繁琐的操作步骤才能进行代码合并。那有没有直接省略这些步骤，让相关人员都有权限直接向仓库提交代码呢？

这里就需要使用另一个功能，Collaborators，意为合作者。为相关人员开通直接push代码的权限。

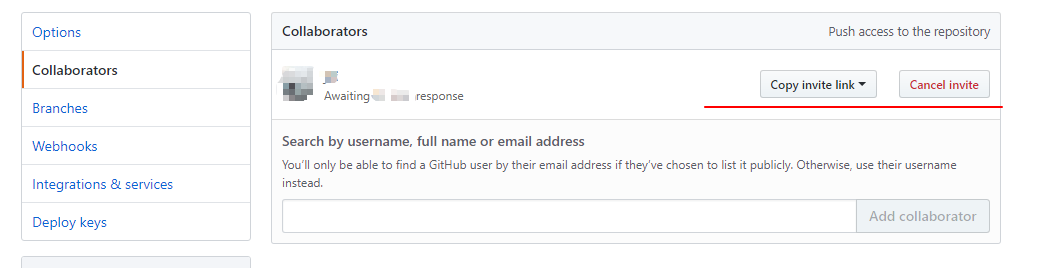


**图：settings**



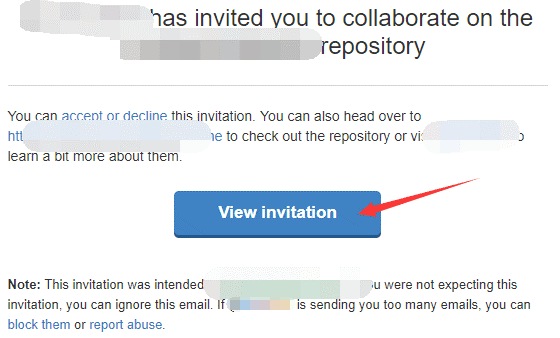
**图：Collaborators**

仓库的所有者或管理者点击仓库的Settings按钮，在页面的左侧菜单选择Collaborators，在右侧可以看到输入GitHub用户名的输入框。



**图：受邀用户**

输入要邀请的GitHub用户名，在下拉菜单中选择相关用户。点击add collaborators按钮。这时候，GitHub会向受邀用户邮箱发送邀请及确认邮件。



**图：确认邀请**

受邀用户确认邀请后，便可以直接clone仓库代码到本地，进行开发工作。并且直接push不会有权限限制。

# 附录

GIt教程-廖雪峰

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

Git分支管理策略及Git flow

<https://www.cnblogs.com/lcngu/p/5770288.html>

Pro Git

<https://gitee.com/progit/index.html>