

# Git/Github

April 28, 2023

Yujie Niu

## 1 Git优势

### 1.1 优点

大部分操作在本地完成，不需要联网

完整性保证(提交每条数据进行hash运算)

尽可能添加数据而不是删除或修改数据

分支操作非常快捷流畅(1因为用快照,2每个分支只是创建一个指针)

与 Linux 命令全面兼容

### 1.2 结构

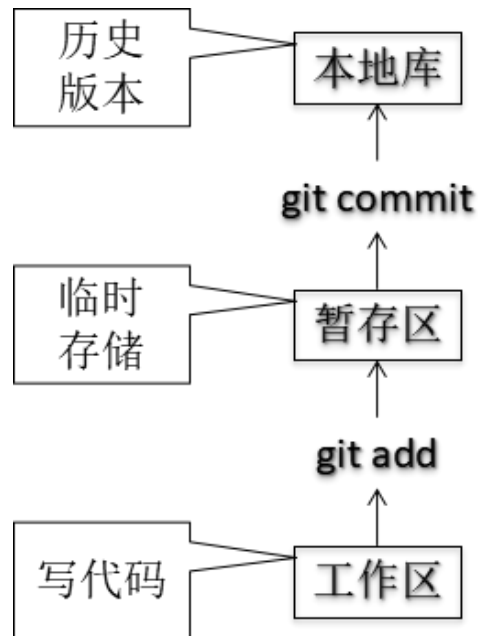


图 1: Git结构

## 2 Git和代码托管中心

\*Github只是其中一个！

为什么需要代码托管中心？

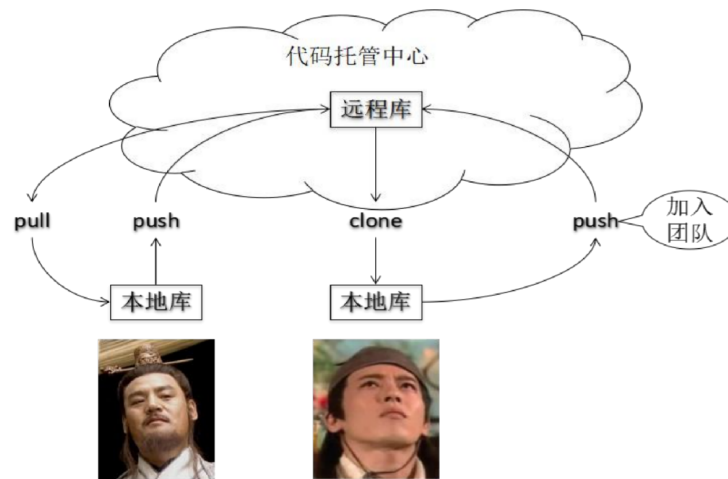


图 2: 团队内部合作

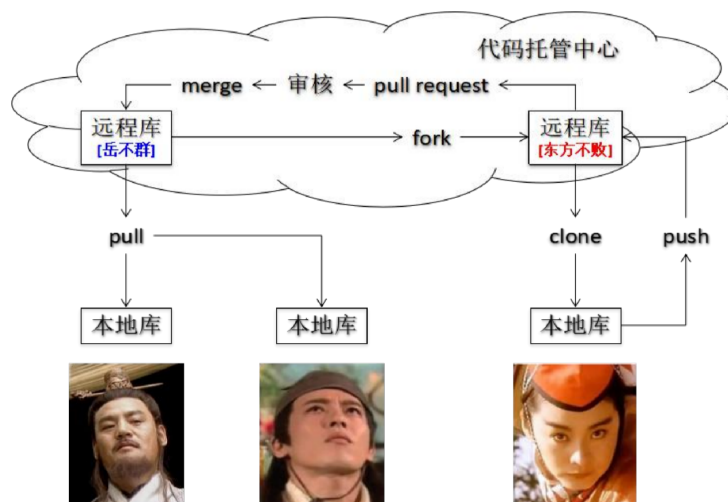


图 3: 跨团队合作

## 3.Git命令行操作

### 3.1 本地库初始化

1.git中的命令与Linux兼容

2.创建目录，在本地“Git bash here”

3.本地库初始化 git init (创建.git文件，.git目录中存放的是本地库相关的子目录及文件，不要删除，也不要乱修改)

### 3.2.1 设置签名

形式：用户名；Email地址

作用：区分不同开发人员的身份

辨析：这里设置的签名和登录远程库(代码托管中心的账号、密码没有任何关系)

### 3.2.2 命令

3.2.2.1 项目级别/仓库级别：(不带参数-global)仅在当前本地库范围内有效

git config user.name tom\_pro

git config user.email goodMorning\_pro@taku.com

信息保存位置: `./git/config` 文件

**3.2.2.2 系统用户级别:** 登录当前操作系统的用户范围

`git config --global user.name tom_glb`

`git config --global user.email goodMorning_glb@taku.com`

信息保存位置: `/.gitconfig` 文件 (家目录下c/user)

**3.2.2.3 级别优先级**

1.就近原则: 项目级别优先于系统用户级别, 二者都有时采用项目级别的签名

2.如果只有系统用户级别的签名, 就以系统用户级别的签名为准

3.二者都没有不允许

### 3.3 基础操作命令

1. `git status` 查看工作区、暂存区状态

2. 添加/撤回>暂存区 (已经add的文件,修改后可以直接commit)

`git add [file name]` 提交到暂存区,并且转换换行符 将工作区的“新建/修改”添加到暂存区

`git rm --cached [file name]` 从暂存区撤回, `[file name]` 使用时写成具体文件名即可

3. 提交(这里和安装是选择vim编辑器有关)

`git commit file` 需要输入提交信息日志 (进入了vim, 为了写这次提交是干啥, 如果提交后修改会出现modified提示)

`git commit -a []`直接提交, 不能够撤销

`git commit -m "commit message" [file name]` 将暂存区的内容提交到本地库, “”为本次提交修改的内容, -m输入后不再需要进入编辑器!

`git restore` 撤销某个文件修改的操作

`git reset HEAD` 可以把暂存区的修改撤销掉(unstage),重新放回工作区

## 4 版本穿梭准备

### 4.1 查看版本记录

`git log`即可 (HEAD指向当前分支)

`git log --pretty=oneline` 每个历史只显示一行(hash值和日志)

`git log --oneline` 每个历史只显示一行且显示hash的部分值

`git reflog` 显示历史只显示一行,并且显示指针(要移动到版本多少步), HEAD@{移动到当前版本需要多少步}

多屏显示控制方式: 空格向下翻页, b向上翻页, q退出

### 4.2 前进后退版本

基于HEAD进行!

1. (推荐) `git reset --hard` 索引值, 即前面提到的hash, 只需要部分即可

2. 使用^符号: 只能后退 `git reset --hard HEAD^` (后退一步, 几个hat回几次)

3. 使用符号, 只能后退 `git reset --hard HEAD n` (后退n步), 没有符号就回到原位置

ps: git help 命令, 本地查询

git reset三个参数对比:

(1) -soft 仅仅在本地库移动HEAD指针(查看状态时,绿色提示,本地库和暂存区不同步,相对位置改变)

(2) -mixed 在本地库移动HEAD指针, 重置暂存区 (工作区凸显出来)

(3) -hard 在本地库移动 HEAD 指针 重置暂存区 重置工作区 (三个一起, 没有突出)

### 4.3 找回删除文件

找回文件前提: 删除前, 文件存在时的状态提交到了本地库。需要注意, 创建, 删除文件的记录永远不可磨灭。

进入的文件目录> rm 文件名(rm aaa.txt)删除本地文件>然后提交到暂存区git add aaa.txt >然后提交到本地仓库git commit -m "delete aaa" aaa.txt (删除完成)

找回操作: git reset --hard [指针位置]

(1)删除操作已经提交到本地库: 指针位置指向历史记录(回到之前未删除版本)

(2)删除操作尚未提交到本地库: 指针位置使用 HEAD (git reset -hard HEAD)

### 4.4 比较文件

git diff [filename] 将工作区中的文件和暂存区进行比较

若使用git add添加到了暂存区, 想看到差异使用git diff HEAD [] (与本地库比较)

甚至可以git diff HEAD^ [] 比较历史记录, 可以用HEAD或者版本索引值

git diff 不加文件名, 比较全部文件

## 5 分支

在版本控制过程中, 使用多条线同时推进多个任务, 在分支里面开发不影响主线进行。

同时并行推进多个功能开发, 提高开发效率, 各个分支在开发过程中, 如果某一个分支开发失败, 不会对其他分支有任何影响。失败的分支删除重新开始即可。



图 4: 分支

### 5.1 分支操作

git branch -v 查看所有分支

git branch [branch name] 创建分支

git checkout [branch name] 切换分支

### 5.2 合并

合并需要注意, 在某一分支进行完修改后, 其版本现在领先master, 合并步骤如下:

- (1) 切换到接受修改的分支上master
- (2) 执行命令 `git merge [modified branch name]`

### 5.3 合并冲突

5.3.1 冲突原因: 2个分支,修改同一文件,同一位置,修改内容不一样时。

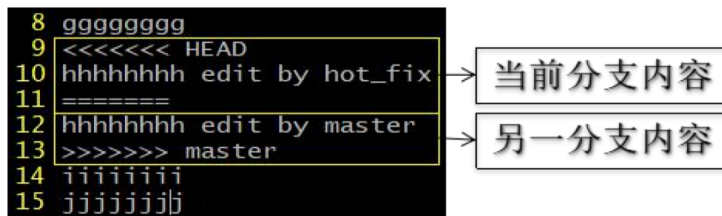


图 5: 冲突

此时转为手动合并!!!!

#### 5.3.2 冲突的解决:

- 第一步: 编辑文件, 删除特殊符号
  - 第二步: 把文件修改到满意的程度, 保存退出
  - 第三步: `git add [文件名]`
  - 第四步: `git commit -m "日志信息"`
- 注意: 此时 commit 一定不能带具体文件名!!!

## 6 Git基本原理

### 6.1 哈希

哈希是一个系列的加密算法, 各个不同的哈希算法虽然加密强度不同, 但是有以下几个共同点:

- ① 不管输入数据的数据量有多大, 输入同一个哈希算法, 得到的加密结果长度固定。
- ② 哈希算法确定, 输入数据确定, 输出数据能够保证不变
- ③ 哈希算法确定, 输入数据有变化, 输出数据一定有变化, 而且通常变化很大
- ④ 哈希算法不可逆

Git 底层采用的是 SHA-1 算法。哈希算法可以被用来验证文件。原理如下图所示: (传输前后hash值对比)

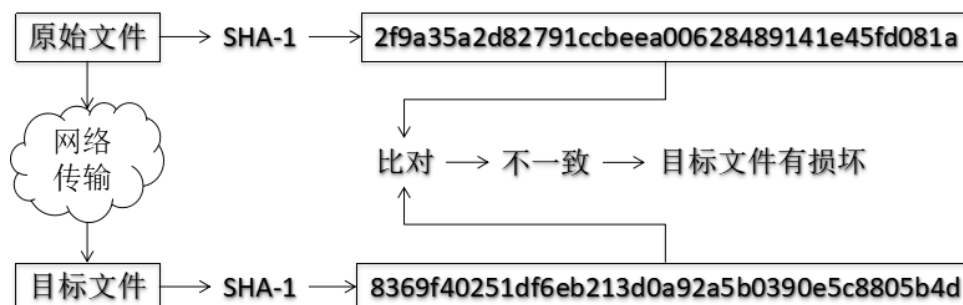


图 6: 哈希算法验证文件原理

### 6.2 Git版本数据管理机制

### 6.2.1 集中式版本控制工具的文件管理机制

以文件变更列表的方式存储信息。这类系统将它们保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。SVN

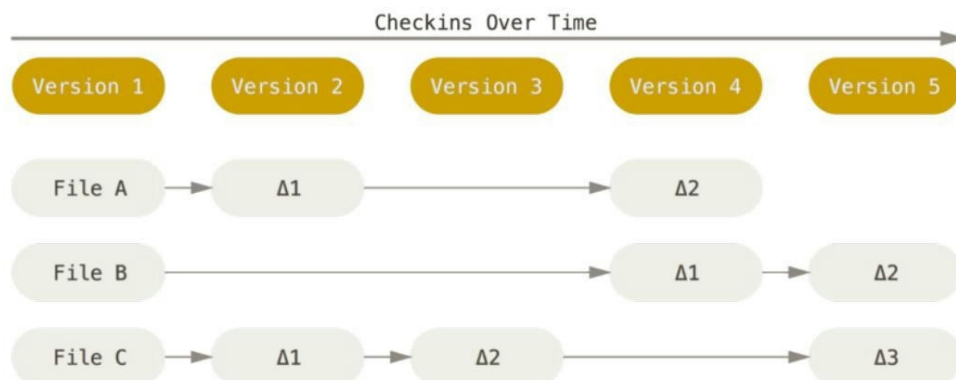


图 7:

### 6.2.2 Git 的文件管理机制

Git 把数据看作是小型文件系统的一组快照。每次提交更新时 Git 都会对当前的全部文件制作一个快照并保存这个快照的索引。为了高效，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。所以 Git 的工作方式可以称之为快照流。

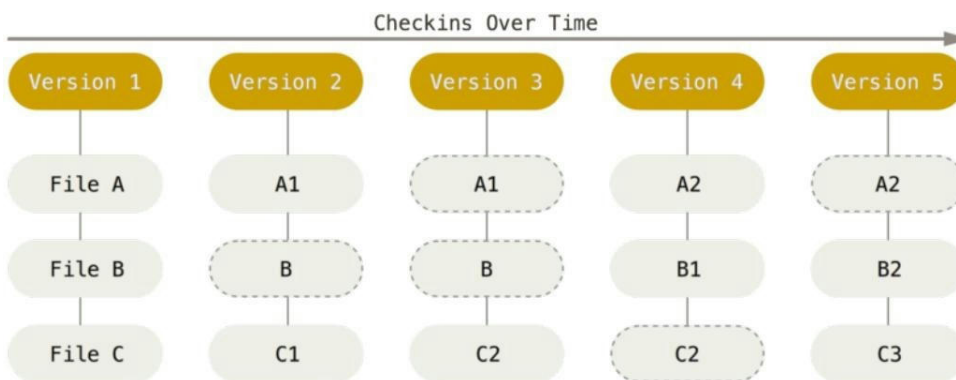


图 8:

### 6.2.3 文件管理机制细节

Git 的“提交对象” (每个文件对应的hash值)



图 9:

提交对象及其父对象形成的链条

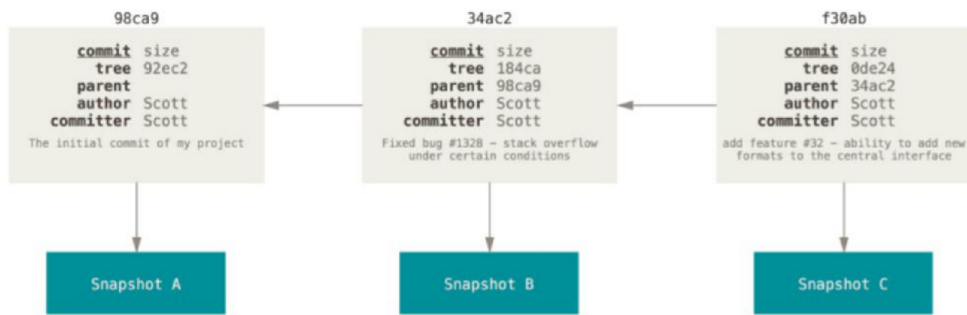


图 10:

## 6.3 Git分支管理机制

### 6.3.1 分支的创建(就是新建一个指针，而不是copy文件)



图 11:

### 6.3.2 分支的切换 (HEAD指向来切换)

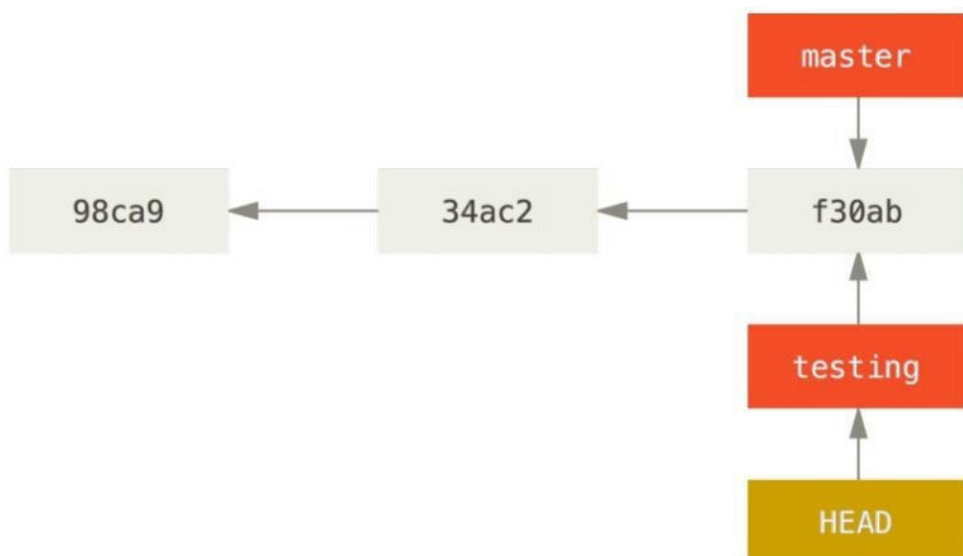


图 12:

### 6.3.3 HEAD指向testing时提交了内容

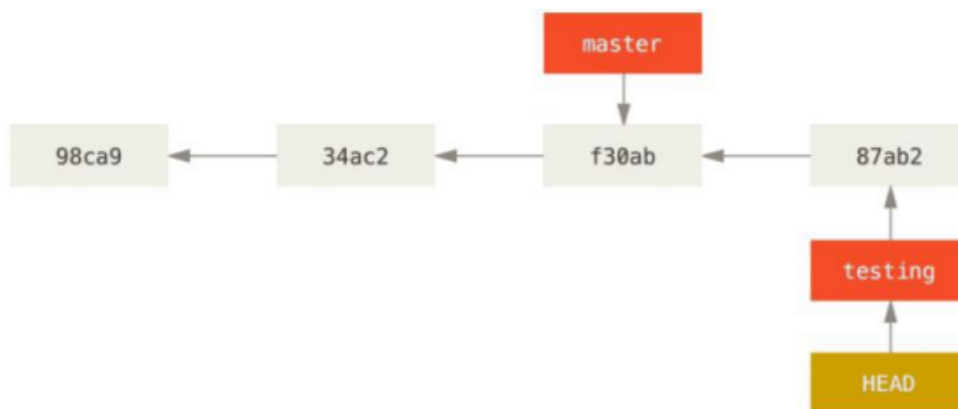


图 13:

### 6.3.4 切换回 master

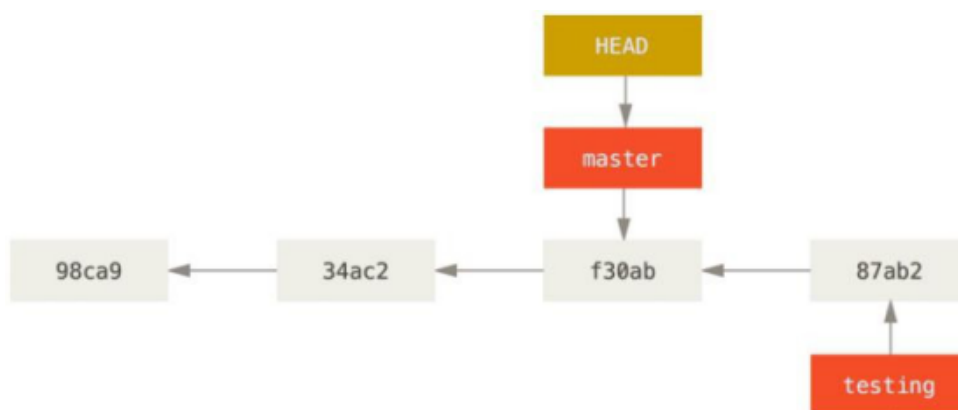


图 14:

### 6.3.5 HEAD指向master时 提交了数据

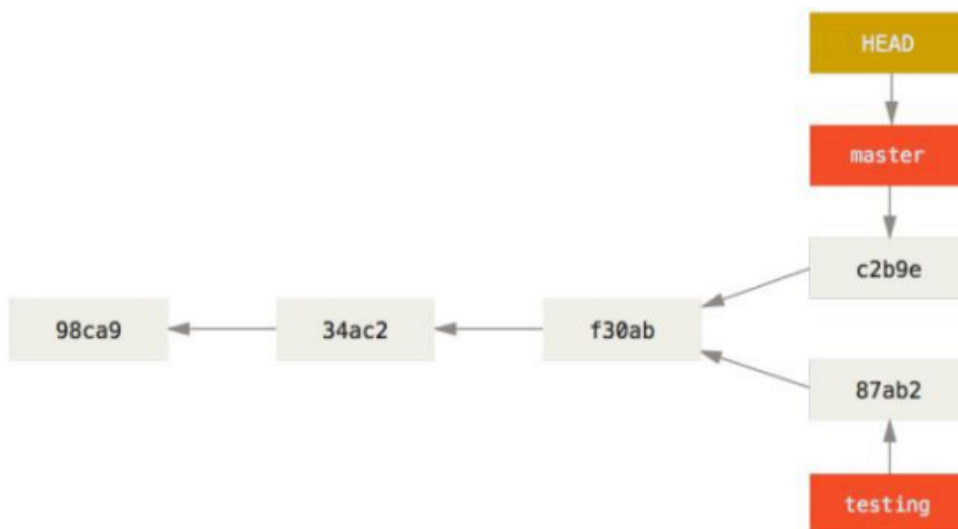


图 15:

## 7 GitHub

### 7.1 注册等略去

### 7.2 远程交互



### 7.2.1 初始化新的本地库

cd 某目录—> git init 初始化—新建文件 —> vim .txt—git add—>git commit -m “”

### 7.2.2 创建远程库（new repository）

这个设置比较简单，略。

### 7.2.3 在本地库创建远程库地址别名

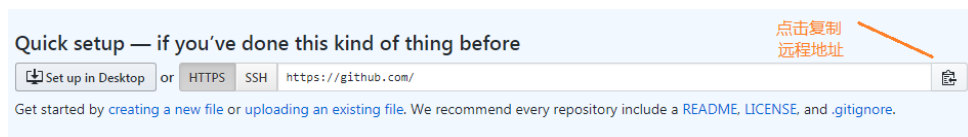


图 16:

git remote -v 查看有无地址别名

git remote add origin http:..... 此处origin为起的别名

### 7.2.4 推送

git push origin（别名） master（分支名）

### 7.2.5 克隆

在新的本地库，不再是之前的owner

git clone http:.....

克隆效果:1.完整的把远程库下载到本地 2.创建 origin 远程地址别名 (git remote -v查看远程库别名) 3.初始化本地库(就是:git init)

### 7.2.6 邀请新用户加入团队，才可以进行推送

点击setting，然后—————

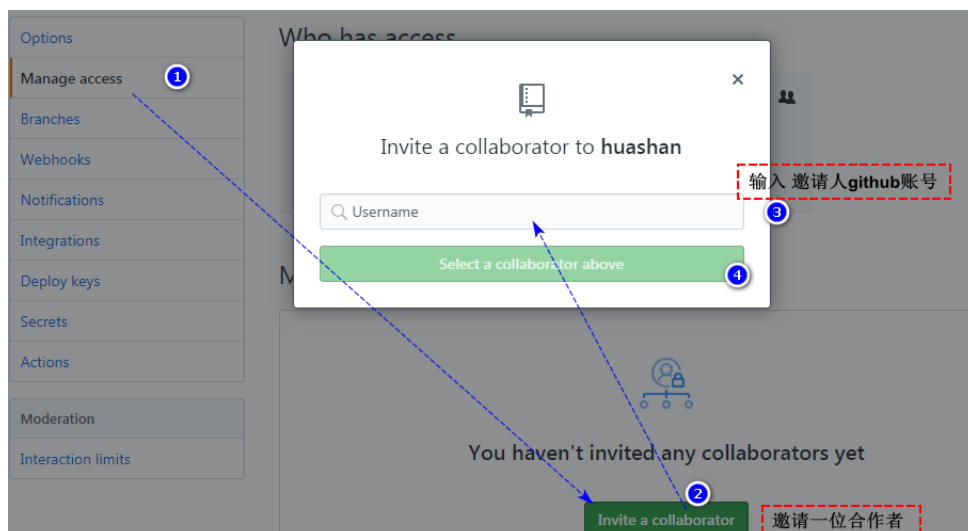


图 17:



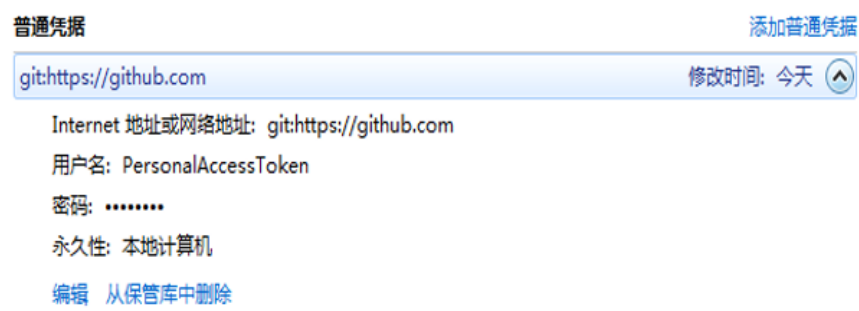
点击接受 > 然后在执行推送



:推送了第一次在此推送不要输入用户名:git 本身不具备记录功能,Windows 中凭据管理器记录用户名和密码

图 18:

控制面板\所有控制面板项\凭据管理器(如果想切换用户:删除记录)



1hc>提交后>>然后推送 `git push origin master`

图 19:

### 7.2.7 远程库修改的拉取

(1) `git fetch origin` (别名——远程库) `master` (远程库分支——master)

只是把远程的内容下载到本地，并没有修改本地工作区文件，如果需要看下载下来的分支，需要：

`git checkout origin/master`

切换到需要的分支，进行查看。

(2) `pull = fetch + merge` (如果操作修改简单，不会产生冲突，直接用pull，冲突就分开用)

`git fetch [远程库地址别名] [远程分支名]`

`git merge [远程库地址别名/远程分支名]`

### 7.2.8 协作开发的冲突

要点：如果不是基于 GitHub 远程库的最新版所做的修改，不能推送，必须先拉取。拉取下来后如果进入冲突状态，则按照“分支冲突解决”操作解决即可。

类比：

债权人：老王

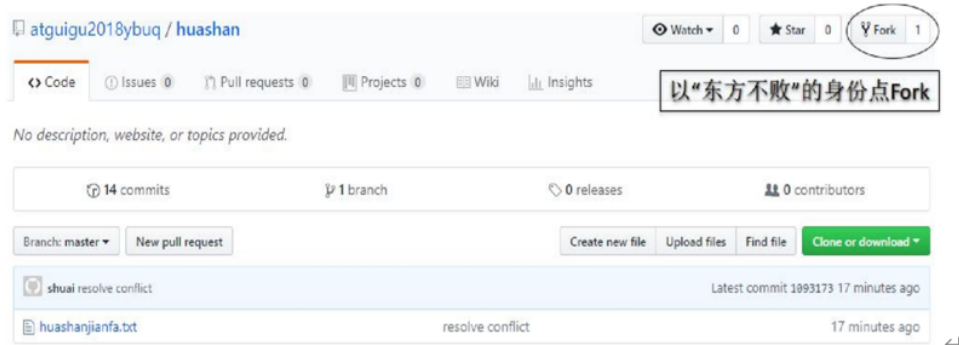
债务人：小刘

老王说：10 天后归还。小刘接受，双方达成一致。

老王媳妇说：5 天后归还。小刘不能接受。老王媳妇需要找老王确认后再执行

### 7.2.9 跨团队合作举例

1(先复制当前库地址,发式给 dfbb,然后有 dfbb 登录访问这个地址)>然后 Fork ↩



### Forking atguigu2018ybuq/huashan

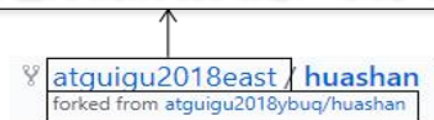
It should only take a few seconds.



正在 fork 的界面↩

fork 过来的仓库说明 回多下面一行(forked from at...)说明 fork 来源↩

说明当前仓库的所有人是：东方不败



说明了fork的来源

2 dfbb("东方不败")本地修改, 然后推送到远程 `git push origin master`

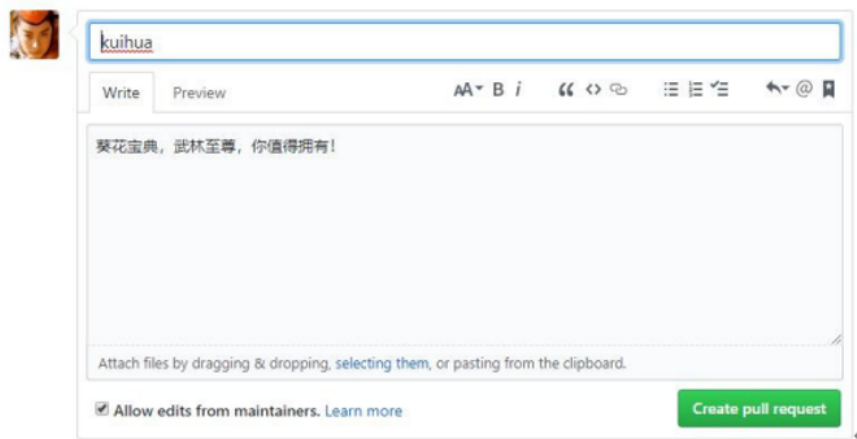
3 dfbb 在远程库中选择 Pull Request



3.2 然后点击里面的 New pull request

3.3 然后点击 Create pull request

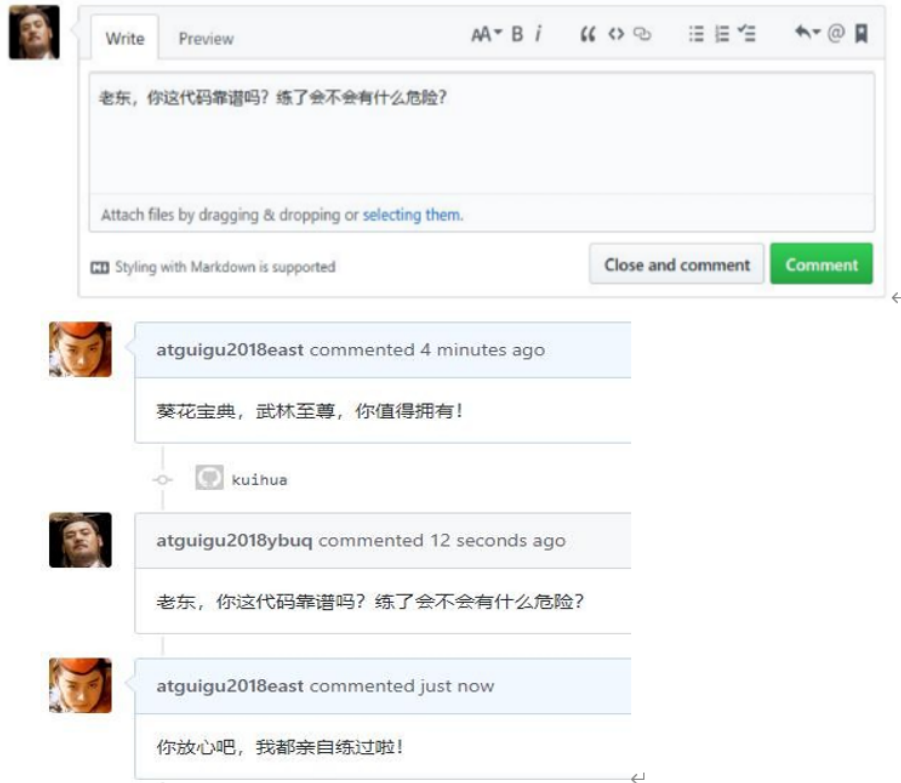
3.4 然后发送消息给, fork 的库 (ybq(岳不群))



4 ybq 操作



## 5.2 对话 (这时还可以相互对话)



## 6 审核代码



合并代码 (回到对话 Conversation>>合并操作如图)



上面操作完了就远程库就有合并内容>然后>将远程库修改拉取到本地

### 7.2.10 SSH免密登录

1 进入当前用户的家目录

```
cd
```

2删除.ssh 目录

```
rm -rvf .ssh
```

3运行命令生成.ssh 密钥目录

```
ssh-keygen -t rsa -C atguigu2018ybuq@aliyun.com
```

[注意：这里-C 这个参数是大写的 C] 后面直接回车(使用默认)

4进入.ssh 目录查看文件列表

```
cd .ssh
```

```
ls -lF
```

5 查看 id\_rsa.pub 文件内容

```
cat id_rsa.pub
```

6 复制 id\_rsa.pub 文件内容,登录 GitHub,点击用户头像→Settings→SSH and GPG keys->New SSH Key

然后>>key中输入复制的密钥信息 Title 自定义输入标题

7 回到工作区 `cd >` 创建远程地址别名 `git remote add origin_ssh git@github.com:atguigu2018ybuq/huashan.git`

8 推送文件进行测试

`git push origin_ssh master`

本笔记