**Git& GitHub**

# 1版本控制工具应该具备的功能

协同修改

■多人并行不悖的修改服务器端的同一个文件

数握备份

■不仅保存目录和文件的当前状态，还能够保存每一个提交过的历史状态

版本管理

■在保存每一个版本的文件信息的时候要做到不保存重复数据，以节约存储空间，提高运行效率，这方面SVN采用的是增量式管理的方式，而Git采取了文件系统快照的方式。

权限控制

■对团队中参与开发的人员进行权限控制；

■对团队外开发者贡献的代码进行审核—Git独有。

历史记录

■查看修改人、修改时间、修改内容、日志信息；

■将本地文件恢复到某一个历史状态。

分支管理

■允许开发团队在工作过程中多条生产线同时推进任务，进一步提高效率。

# 2版本控制简介

## 2.1版本控制

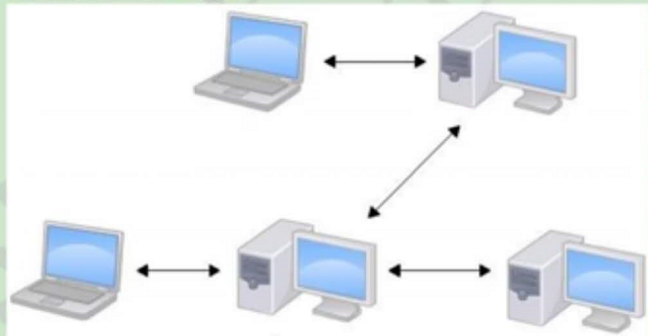
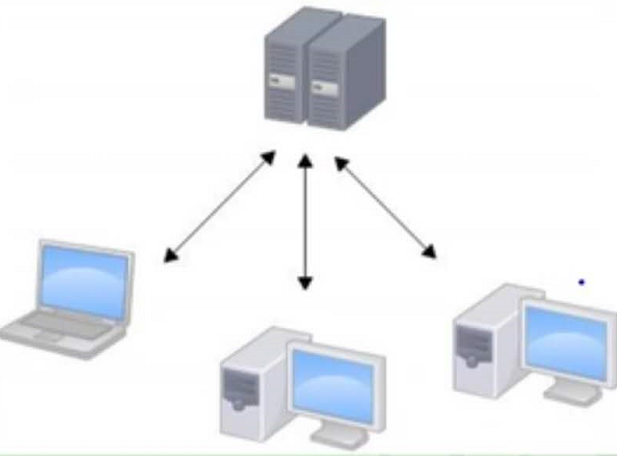
工程设计领域中使用版本控制管理工程蓝图的设计过程。在IT开发过程中也可以使用版本控制思想管理代码的版本选代

## 2.2版本控制工具

思想：版本控制

实现：版本控制工具

集中式版本控制工具:CS、 SVN. VSS 分布式版本控制工具Git、 Mercurial、 Bazaar…



注意：集中式版本控制工具有单点故障问题，服务器宕机后就没有数据了；因为文件和历史版本都在服务器，每个程序员本地只有当前的代码数据；分布式版本控制工具因为文件和历史版本每个程序员本地都有，所以没有这个问题。

# 3git简介

## 3.1Git官网和Logo



## 3.2Git的优势

大部分操作在本地完成不需要联网

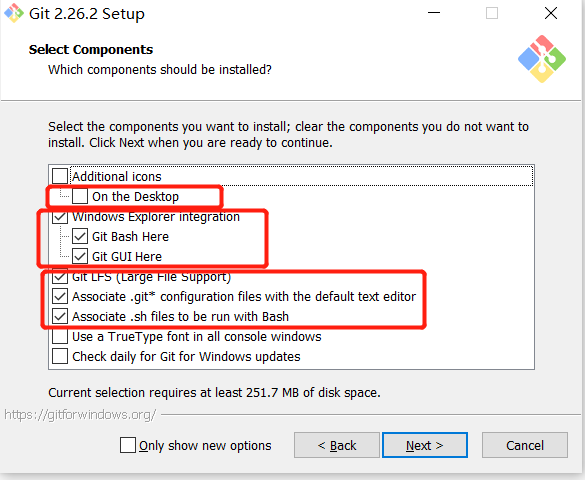
完整性保证

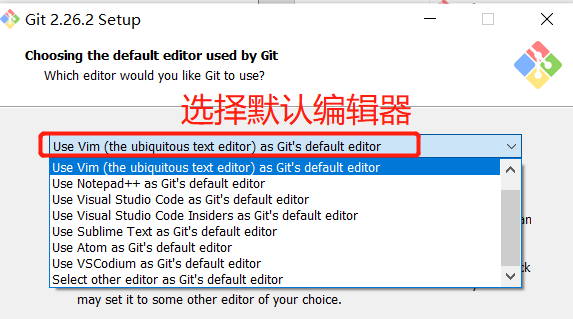
尽可能添加数据而不是删除或修改数据

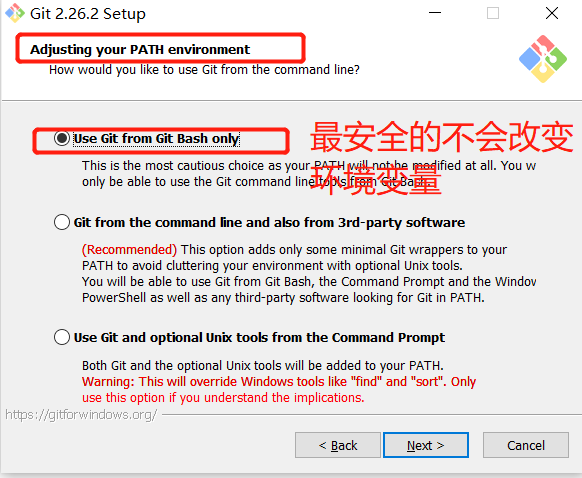
分支操作非常快捷流畅

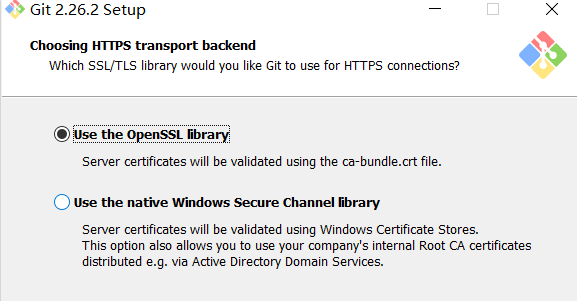
与Lnux命令全面兼容

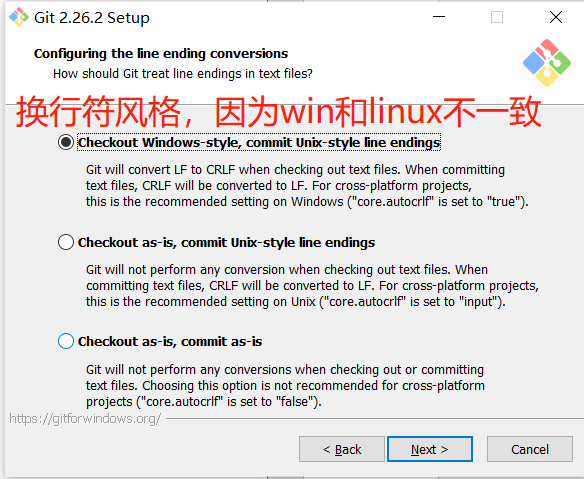
## 3.3 git安装

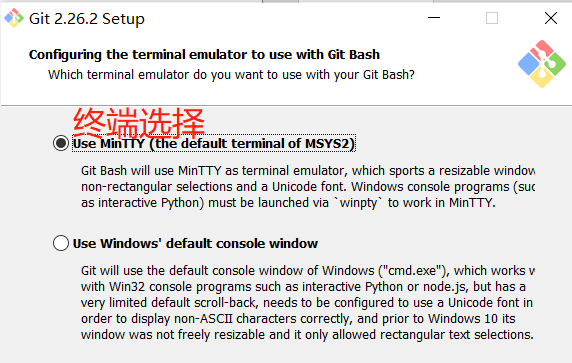


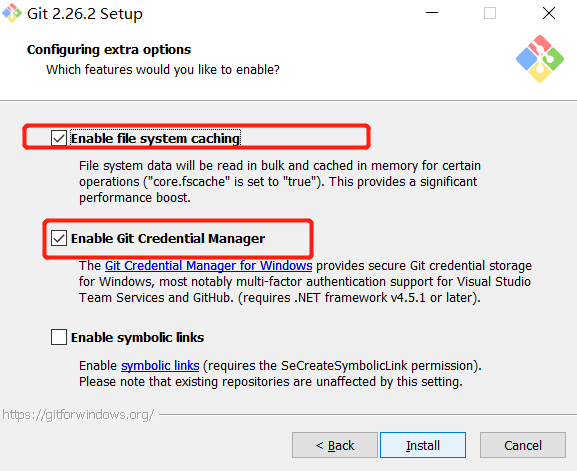




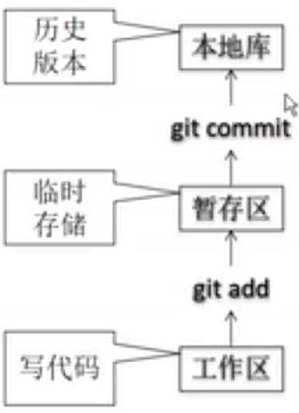








## 3.4 git结构



本地结构

## 3.5 git和代码托管中心

代码托管中心的任务：维护远程库

局域网环境下

■ GitLab服务器

外网环境下

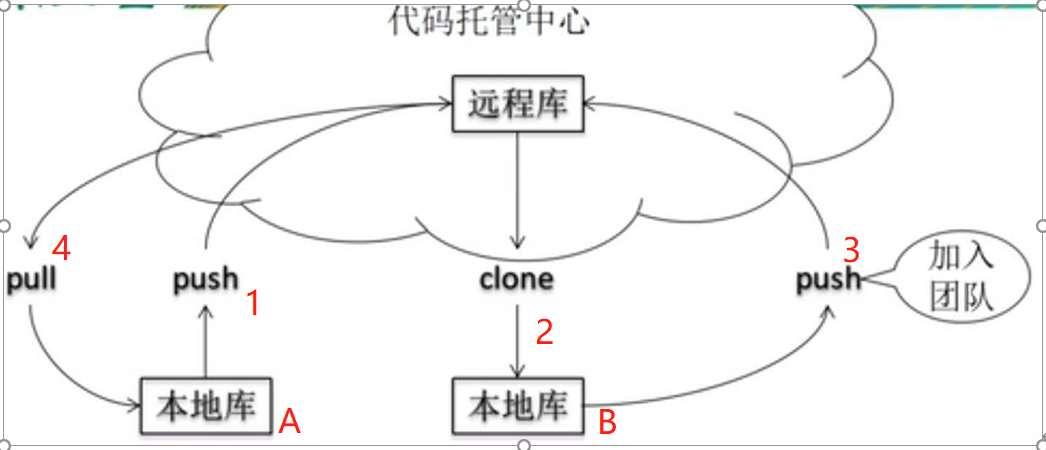
■GitHub

■码云

## 3.6本地库和远程库

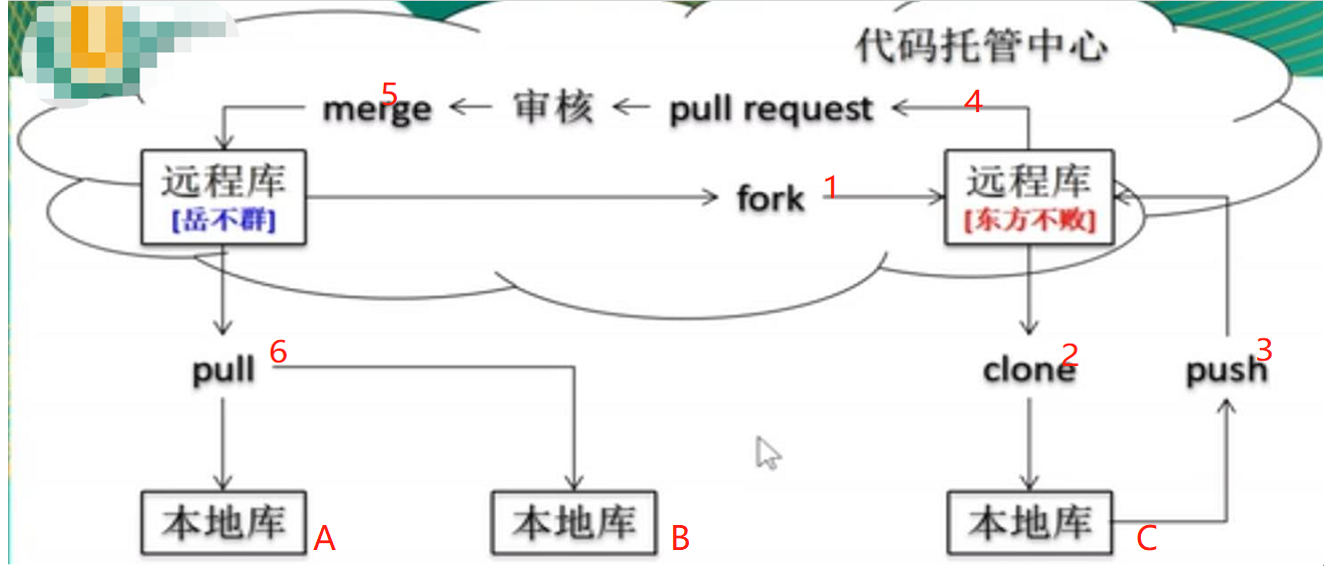
团队内部协作

跨团队协作



团队内部协作

步骤：A先创建本地库，为了推送到远程代码托管中心，就在代码托管中心创建远程库（开始是空的）；再将本地库push到远程库；B如果clone远程库（不止下载远程库数据而且会将本地库初始化）；B修改完代码但是不能直接push到远程库，需要A邀请加入团队才可以推送；B完成推送后A就可以pull拉取远程库到自己的本地库。就完成团队协作。



跨团队协作

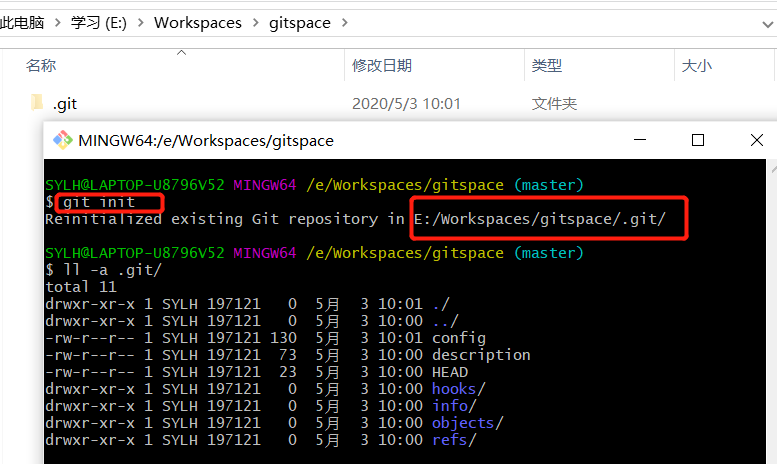
步骤：AB为同一团队；C为团队外人员；A给B安排任务B无法完成但是B的朋友C可以完成；然后C就从A的远程库fork一份属于自己的远程库，然后clone到C本地修改后push到C的远程库，然后C就向A的远程库发起拉取请求；A审核后同意就会进行在线merge合并到A的远程库；然后A和B就可拉取A的远程库内容。完成跨团队协作。

# 4 git命名行操作

## 4.1本地库初始化

命令：git init

效果:



注意：.git目录中存放的是本地库相关的子目录和文件，不要删除，也不要胡乱修改。

## 4.2设置签名

形式:

用户名：tom

Email地址：[goodMorning@atguigu.com](mailto:goodMorning@atguigu.com)

作用：区分不同开发人员的身份,就算Email不存在也可以；

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

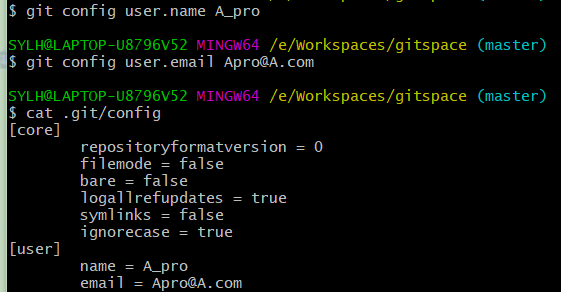
命令

项目级别/仓库级别：仅在当前本地库范围内有效

**git config** user name tom\_pro

**git config** user email [goodmorningpro@atguigu.com](mailto:goodmorningpro@atguigu.com)

信息保存位置：./.git/config文件

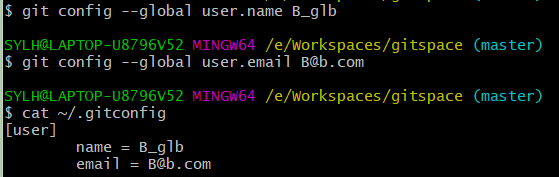


■系统用户级别：登录当前操作系统的用户范围

**git config --global** user name tom\_glb

**git config --global** [goodmorningpro@atguigu.com](mailto:goodmorningpro@atguigu.com)

信息保存位置：~/.gitconfig文件



■级别优先级

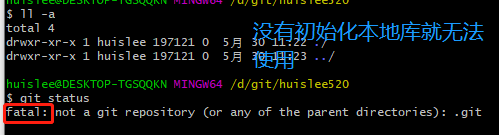
就近原则：项目级别优先于系统用户级别，二者都有时采用项目级别的签名

◆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

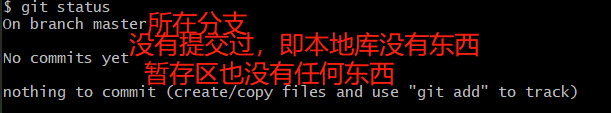
◆二者都没有不允许

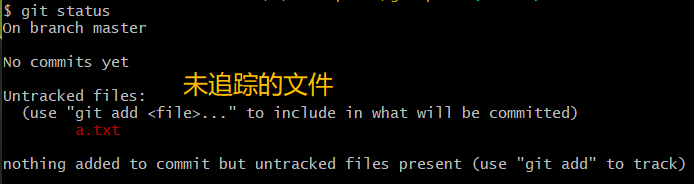
## 4.3基本操作

### 4.3.1状态查看操作



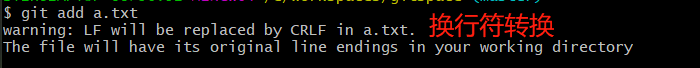
git status查看工作区、暂存区状态

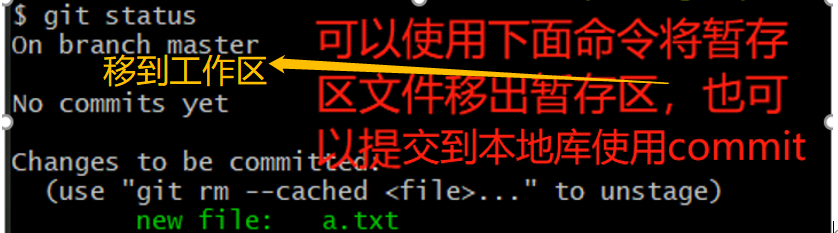


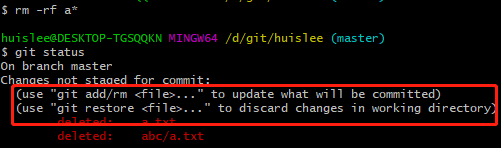


### 4.3.2添加操作

git add [file name] 将工作区的“新建/修改”添加到暂存区





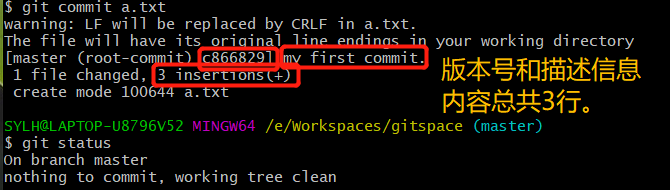


可以删除本地库内容

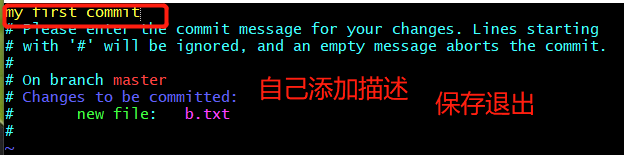
### 4.3.3提交操作

git commit-m "commit message" [file name]

将暂存区的内容提交到本地库

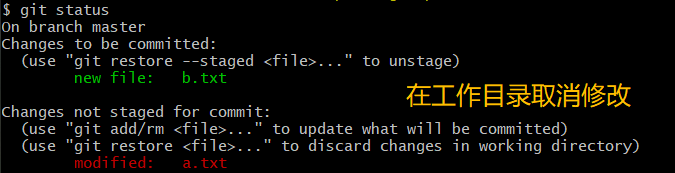


注意：使用git commit 文件；不加描述就会进入下面：



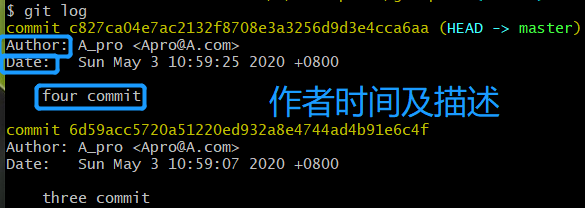
不想进去编辑就使用 git commit 文件 -m “commit一下”

修改文件后查看状态如下：



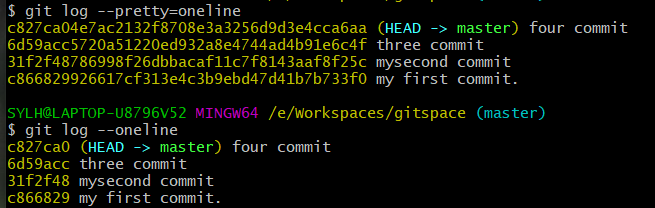
修改后可以直接提交也可以先加到暂存区再提交；

### 4.3.4查看历史记录操作

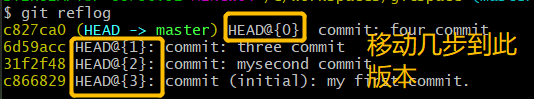


Git log是最完整形式

也可以使用下面方式查看



最常用的是：上面有移动几步可以回到此版本



### 4.3.5 版本前进后退

基于索引值操作：

git reset –hard[局部索引值]

git reset-hard a6ace91;其中a6ace91为查看的版本号对应的哈希

使用^符号: 只能后退

git reset --hard HEAD^

注：一个^表示后退一步，n个表示后退n步

使用~符号: 只能后退

git reset --hard HEAD~n

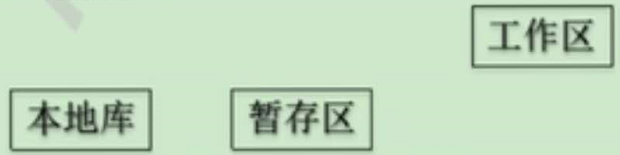
注：表示后退n步

### 4.3.6 reset命令的三个参数对比

**soft参数:仅仅在本地库移动HEAD指针**



**mixed参数:本地库移动HEAD指针并重置暂存区**



**hard参数：在本地库移动HEAD指针并重置暂存区和重置工作区**

### 4.3.7删除文件并找回

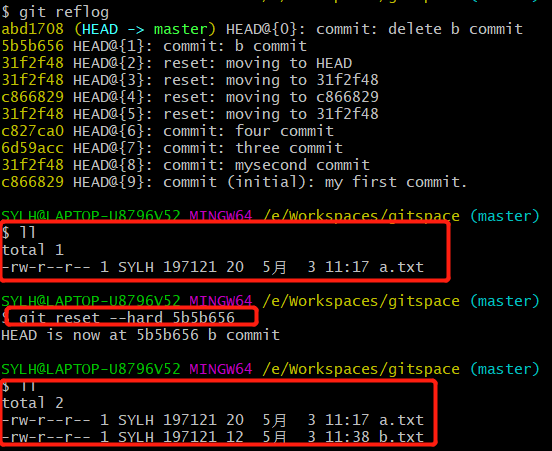
前提：删除前，文件存在时的状态提交到了本地库.

操作：git reset—hard[指针位置]

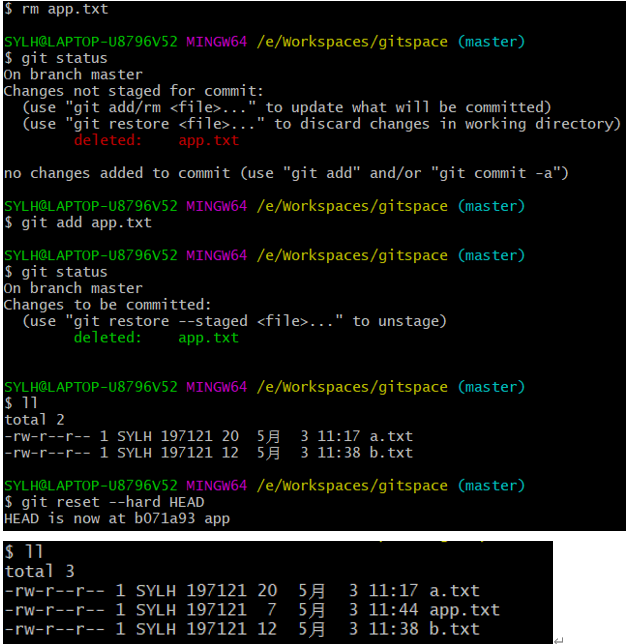
■删除操作已经提交到本地库：指针位置指向历史记录

■删除操作尚未提交到本地库：指针位置使用HEAD

注意：一般是先将文件提交到本地库然后删除后再提交就可以完成删除文件；但是可以使用head指针通过历史记录找回；



本地库文件删除后找回



暂存区删除文件找回

### 4.3.8比较文件差异

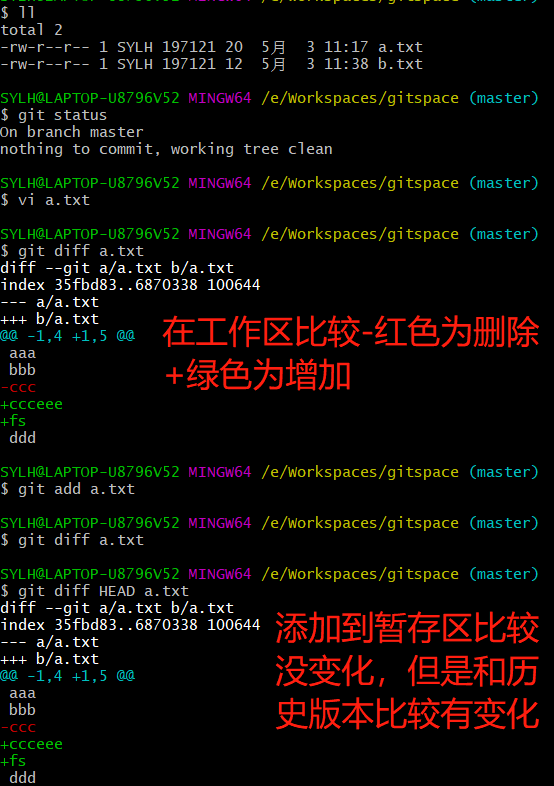
git diff [文件名]

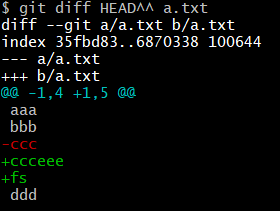
■将工作区中的文件和暂存区进行比较

git diff [本地库中历史版本][文件名]

■将工作区中的文件和本地库历史记录比较

不带文件名比较多个文件



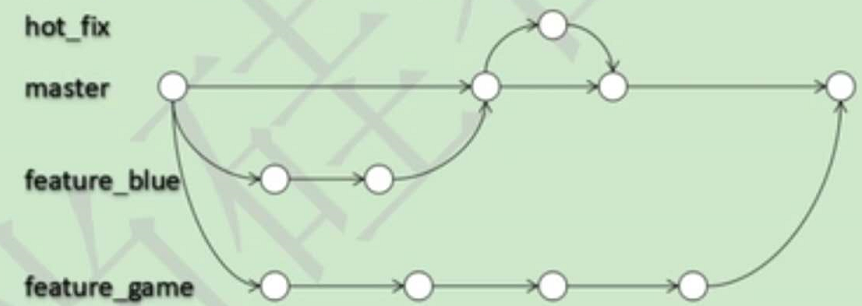
和历史版本比较

### 4.3.9命令帮助

## 4.4分支管理

### 4.4.1为什么要分支？

在版本控制过程中，使用多条线同时推进多个任务。

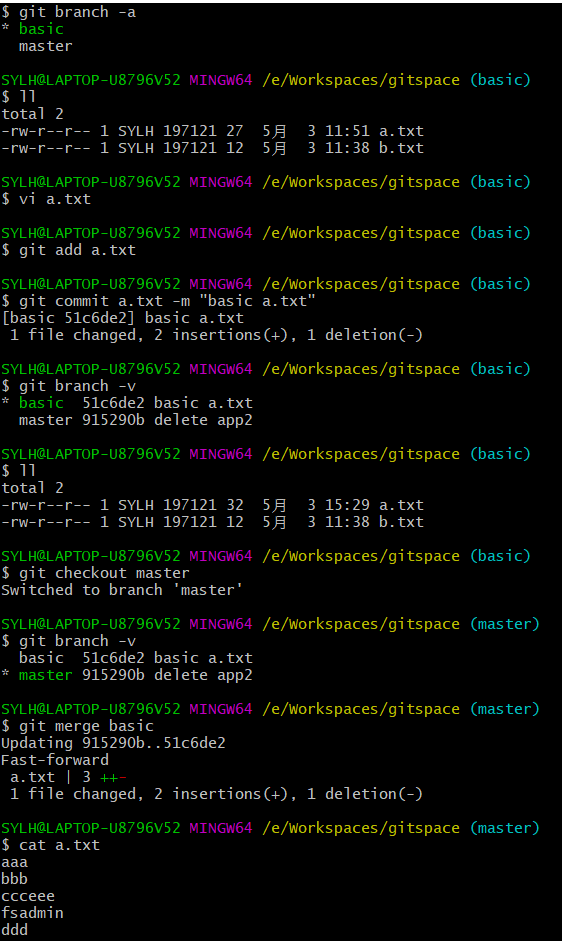


执行过程：先建立新分支并切换到新分支；然后修改bug后先提交到暂存区再提交到本地库；然后切换到master分支执行合并命令就可以将新分支修改后的合并到master分支。

### 4.4.2分支好处

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。



### 4.4.3分支操作

**创建分支**

git branch [分支名]

**查看分支**

git branch-v git branch-a

**切换分支**

git checkout [分支名]

**合并分支**

第一步：切换到接受修改的分支（被合并，增加新内容）上

git checkout【被合并分支名】

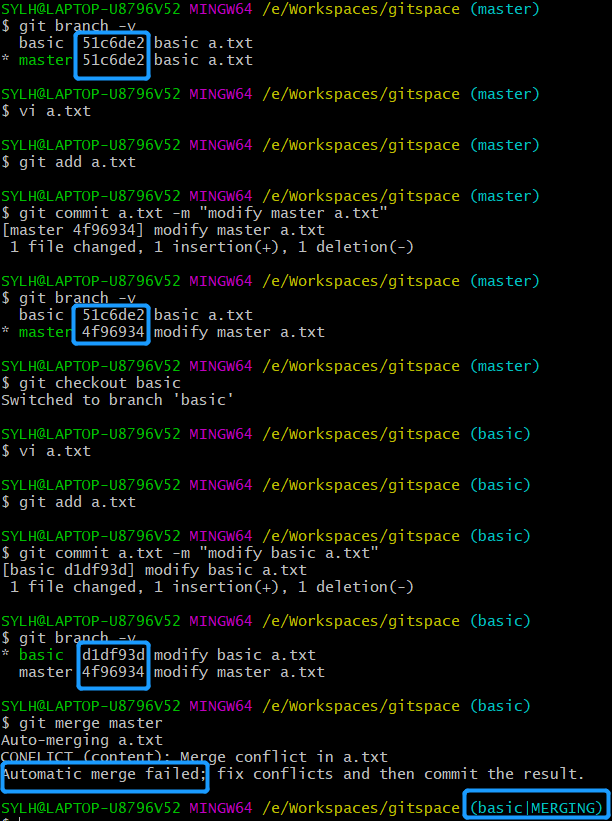
第二步：执行merge命令

git merge【有新内容分支名】

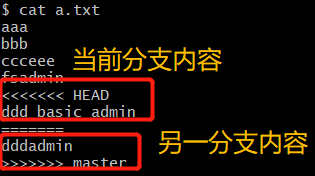
**解决冲突**

在不同分支中同时修改同一文件的同一位置而且修改后的不相同，在合并分支时就会冲突。

冲突的表现：



合并前状态一致指向同一个提交；在master修改后指向不同提交，合并时自动合并失败，后面的“basic | MERGING”显示说明正在合并中。此时查看修改的文件内容如下：



冲突的解决

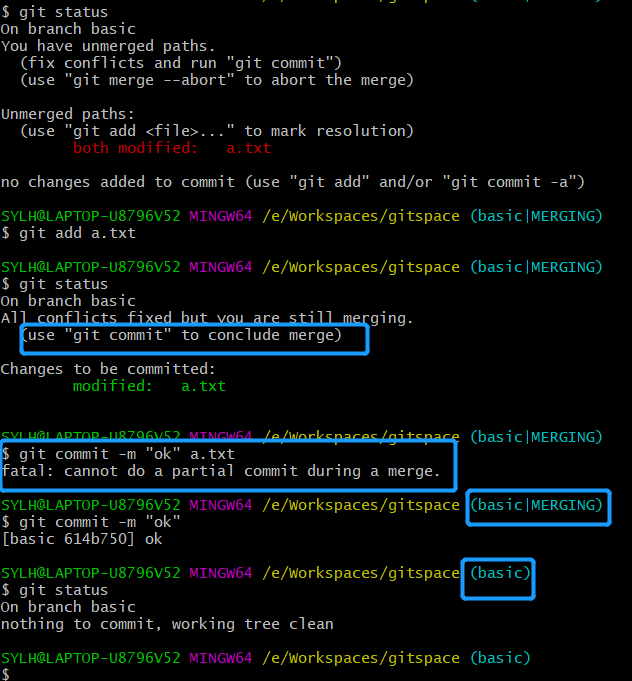
◆第一步：编辑文件，删除特殊符号；

◆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



# 5 Git基本原理

## 5.1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；

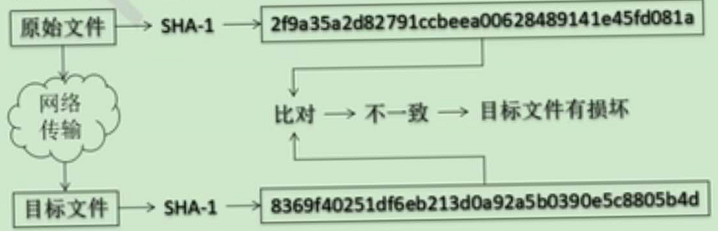
②哈希算法确定，输入数据确定，输出数据能够保证不变；

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：

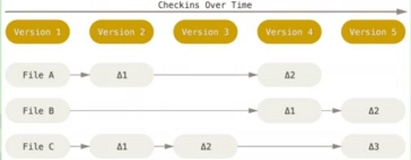


Git就是靠这种机制来从根本上保证数据完整性的.

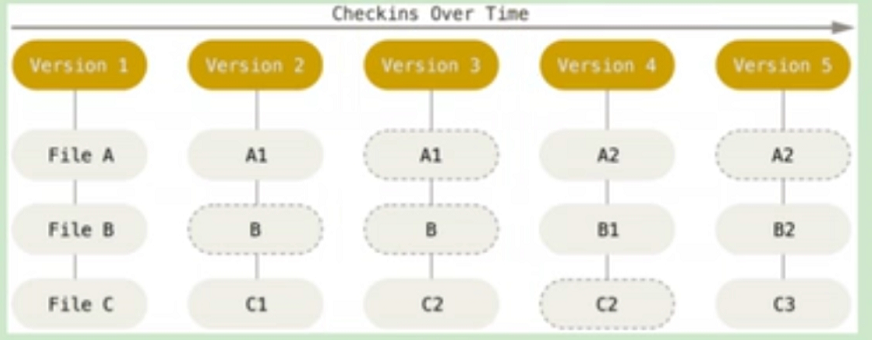
5.2Git保存版本的机制

5.2.1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。

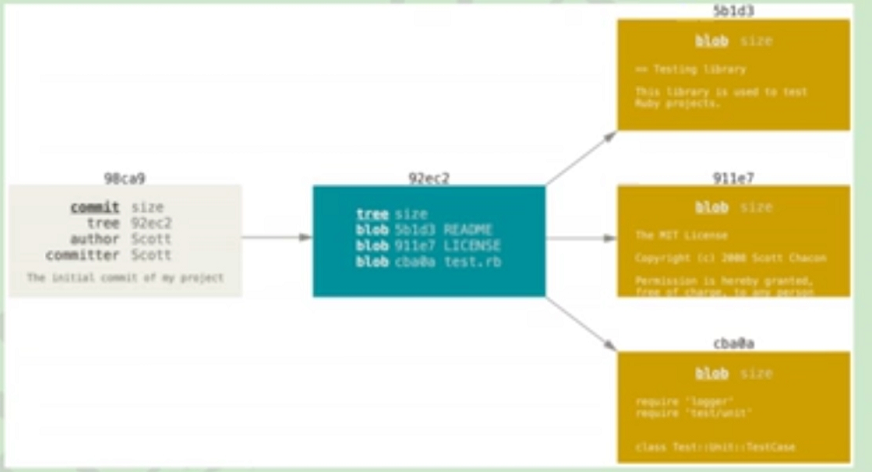


### 5.2.2Git的文件管理机制

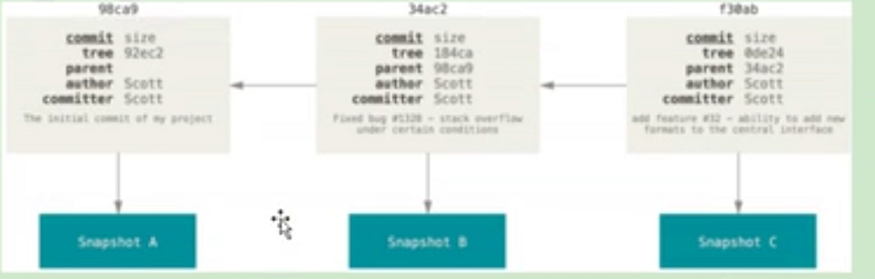
Git把数据看作是小型文件系统的一组快照。每次提交更新时Git都会对当前的全部文件制作一个快照并保存这个快照的索引，为了高效，如果文件没有修改;Git不再重新存储该文件，而是只保留一个链接指向之前存储的文件，所以Git的工作方式可以称之为快照流。

### 5.2.3Git文件管理机制细节

Git的“提交对象”



提交对象及其父对象形成的链条



## 5.3 Git分支管理机制

### 5.3.1分支的创建

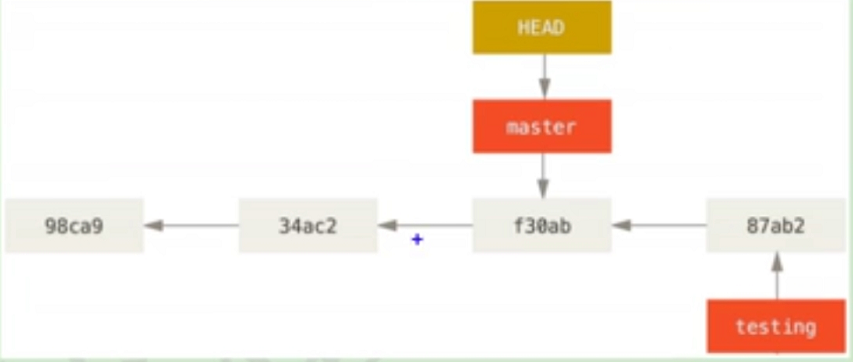


如上：98ca9是根提交，后面提交的会指向前面提交的；当创建分支时会新建指针指向某一版本（当前所在版本）。

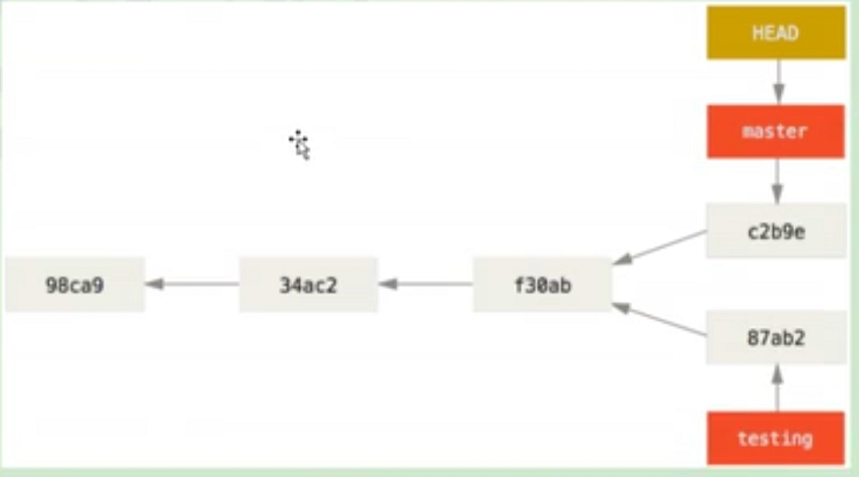
### 5.3.2 分支的切换



切换分支只是head指针的指向变化而已。



如上表示为新建分支发布了新版本，当切换回master分支时还会指向原来的位置。



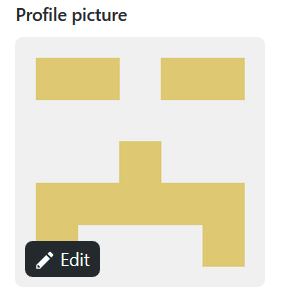
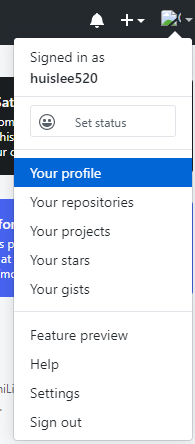
当master分支也提交一次版本就会和其他新建分支指向不同的新版本。

# 6 GitHub

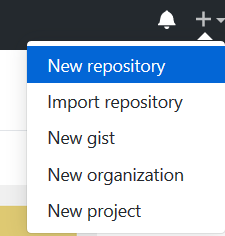
6.1 账号注册 最好用阿里云邮箱

<https://github.com>

## 6.2 修改GitHub头像



## 6.3 创建仓库



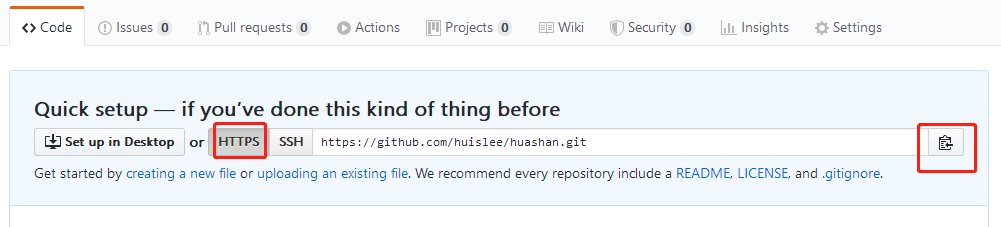


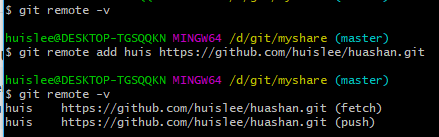
## 6.4 在本地创建远程库别名

git remote查看当前所有远程地址别名

git remote add别名远程地址

先复制仓库地址，如：https://github.com/huislee/huashan.git



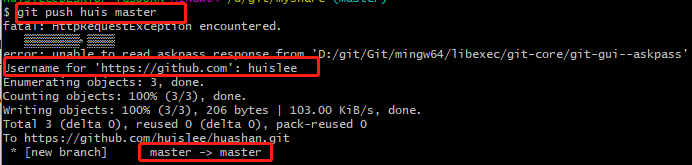


6.5 推送操作

git push别名 [分支名]

注：将本地分支推送到远程库（别名所在分支）





推送后就可以在远程库找到本地库推送的内容。

## 6.6 克隆

命令git origin [远程地址]

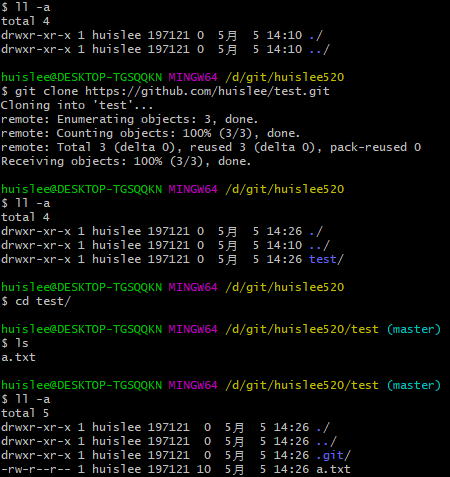
效果:

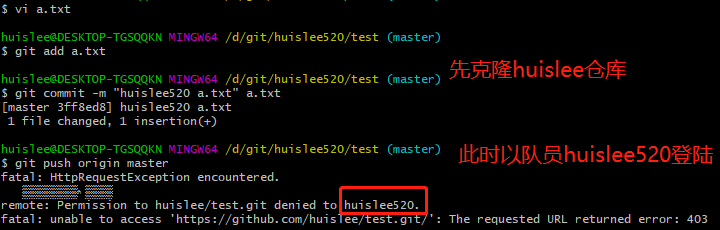
完整的把远程库下载到本地

创建 origin远程地址别名

初始化本地库

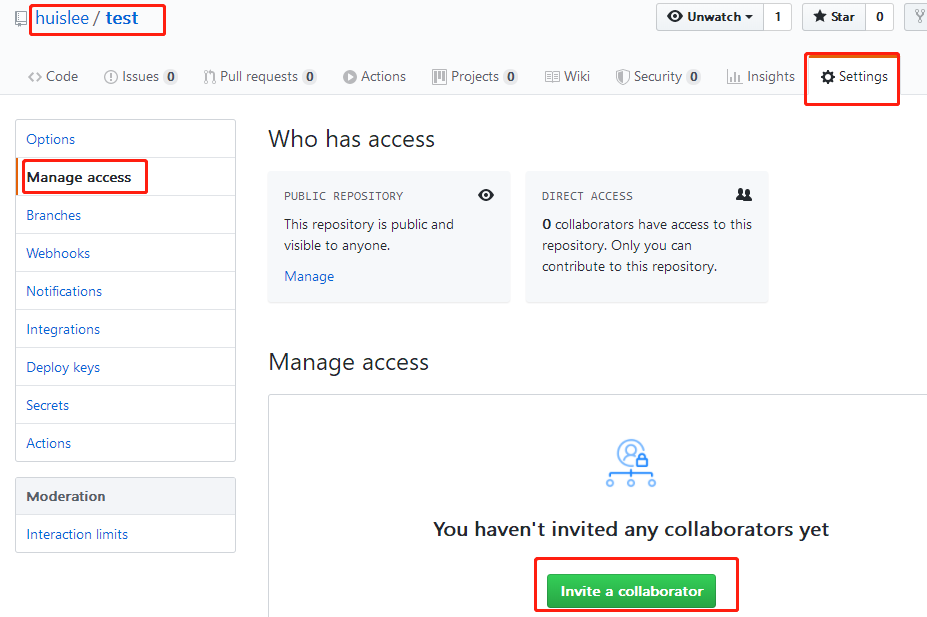
注意：一般是团队中别的人克隆项目管理人员的远程库。如上面的B克隆A远程库。

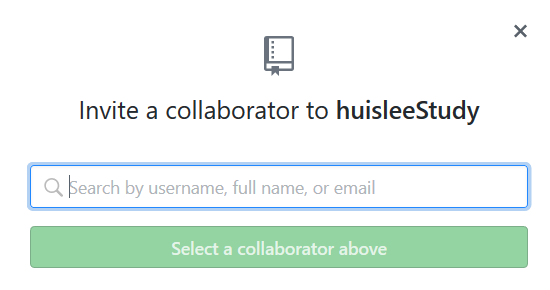


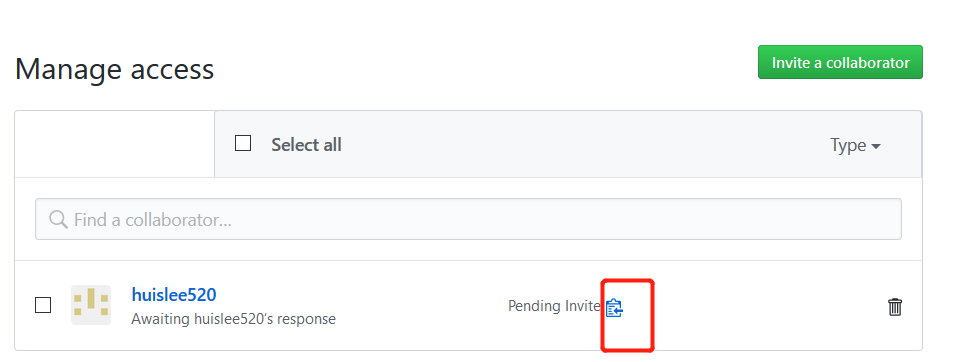


没邀请进团队前只能在自己本地库操作，不能推倒团队的远程库。

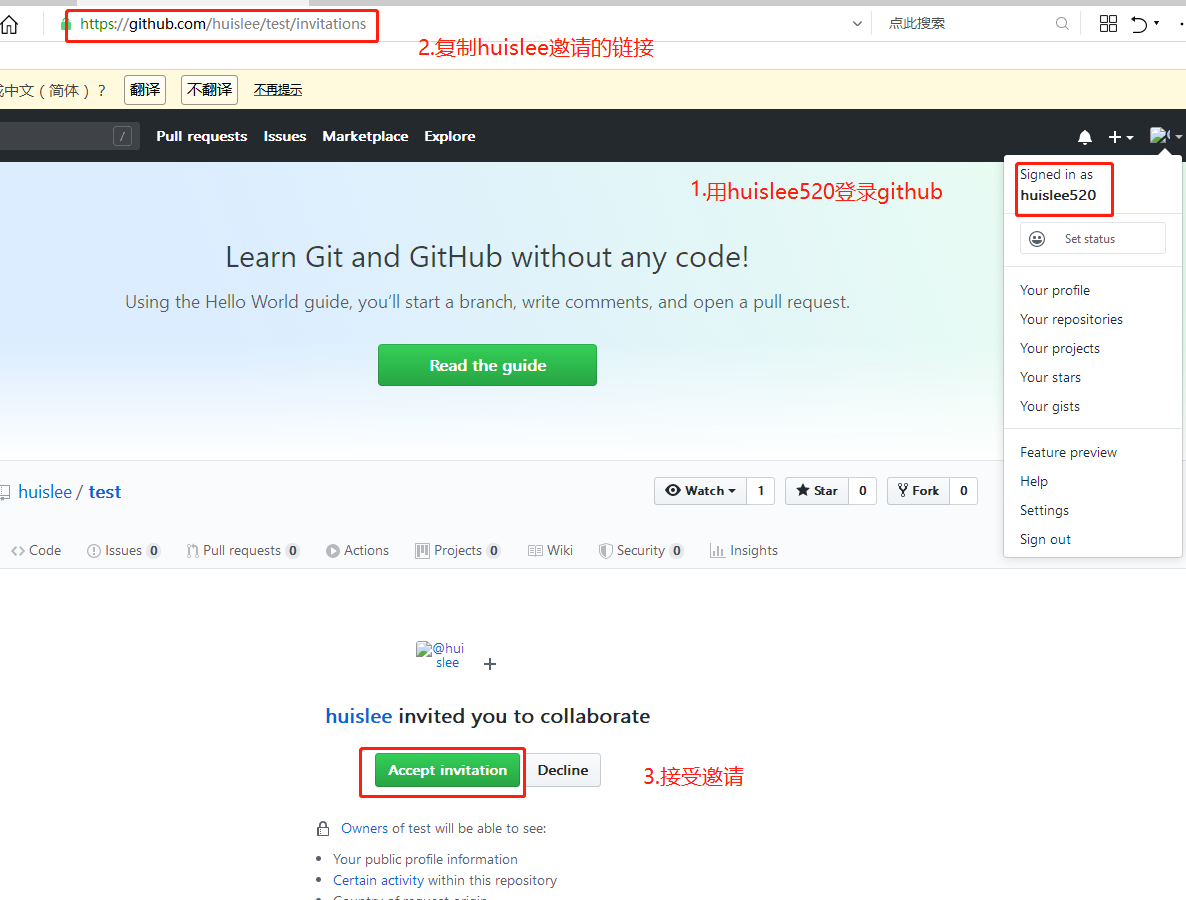
## 6.7 邀请团队成员进团队



填写邀请的GitHub账号

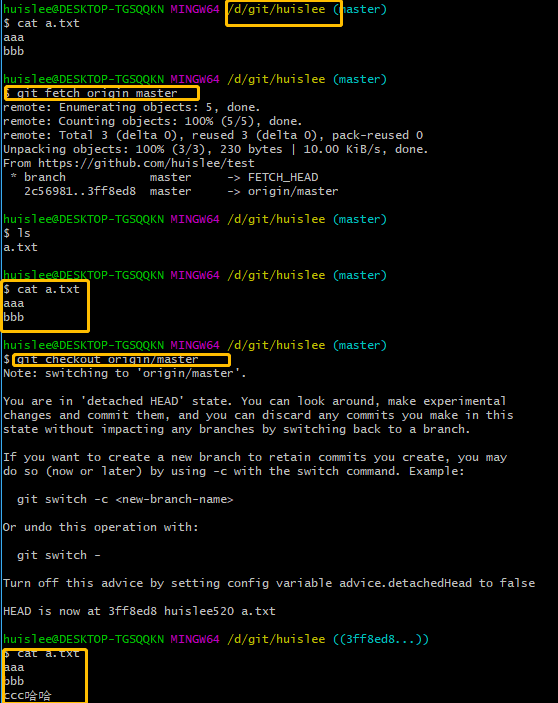


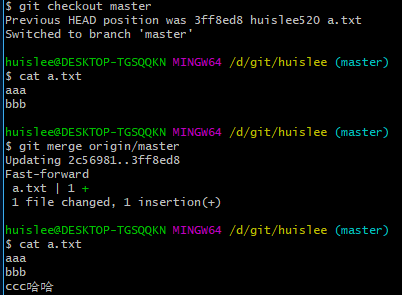
选择后在队友（huislee520）登陆后的页面复制打开进去接受邀请。

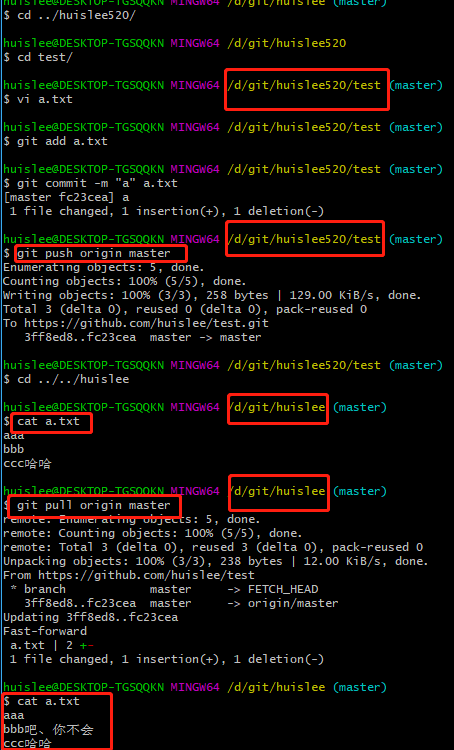


## 6.8 pull远程库到本地库

git fetch远程地址别名 [远程分支名] git merge远程地址别名/远程分支名pull=fetch+merge 拉取=抓取+合并 git pull 远程地址别名 [远程分支名]

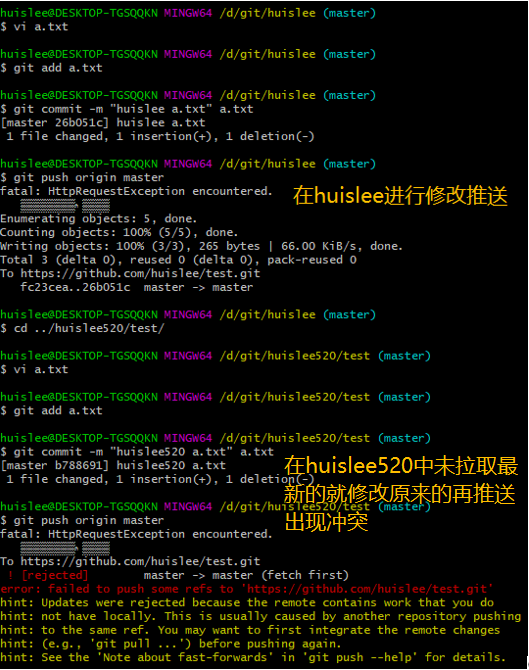




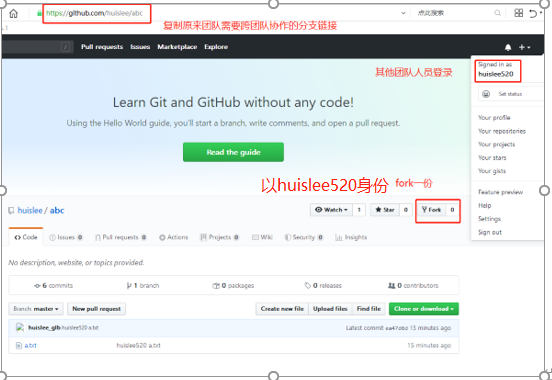


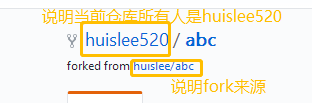
## 6.9 团队协作开发时冲突的解决

出现原因：如果不是基于github远程库最新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。



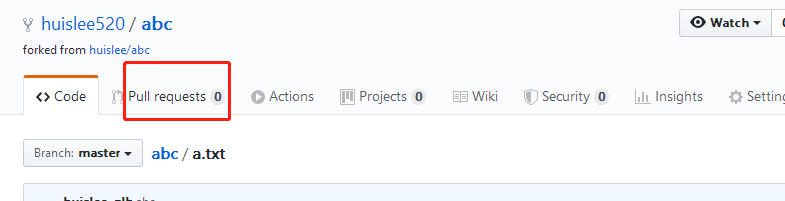
## 6.10 跨团队协作

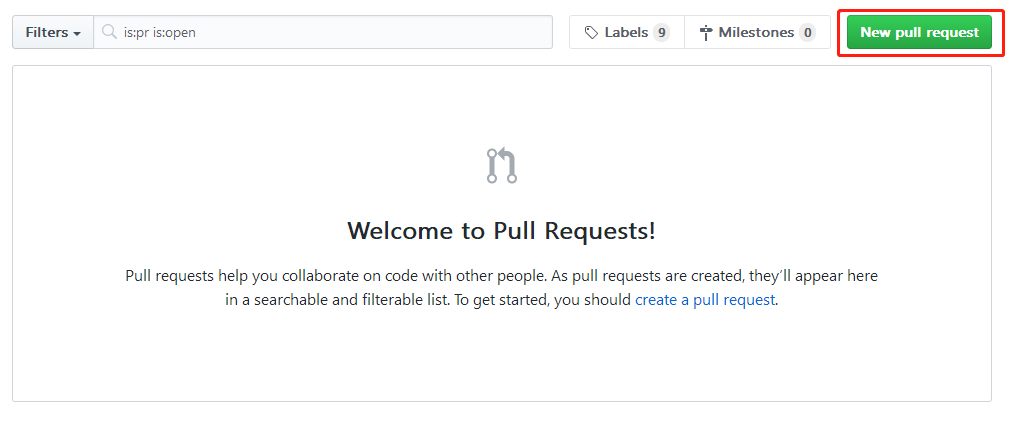




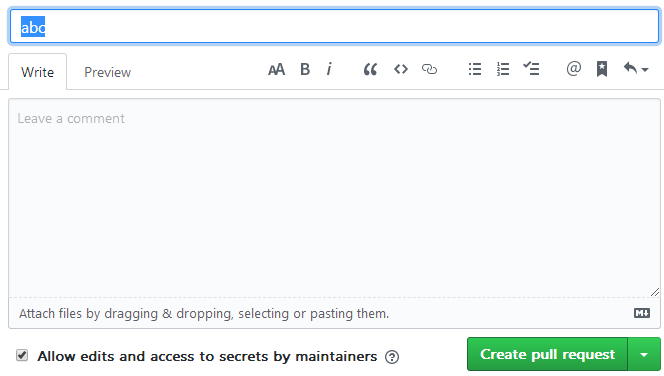
先在本地拉取自己的远程库然后修改后再推送到远程。

Pull request

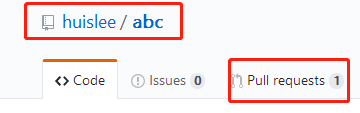




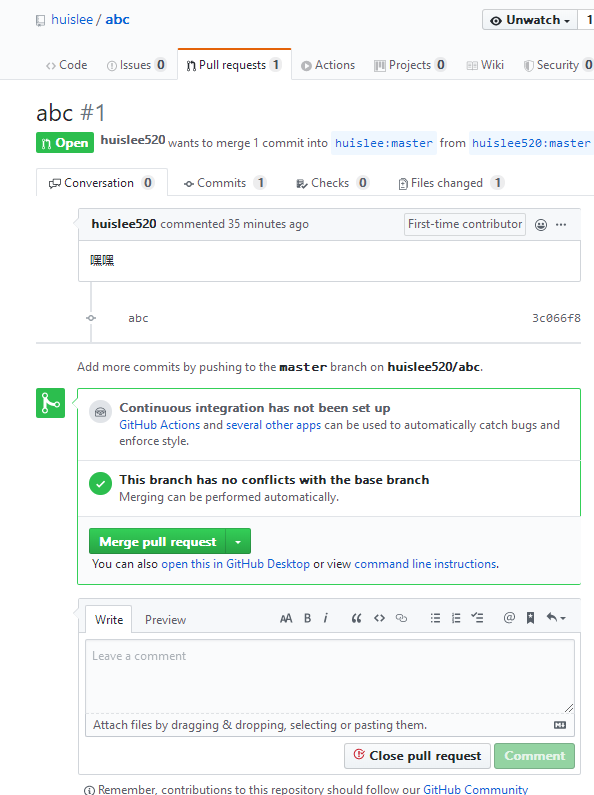




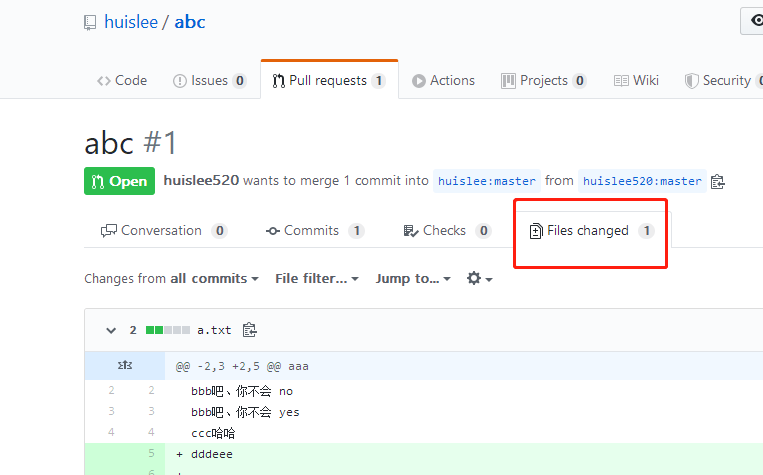
然后在团队的远程库登录



可以进行对话



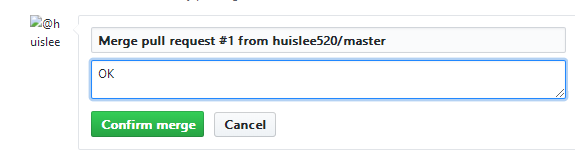
点这里看具体的修改内容

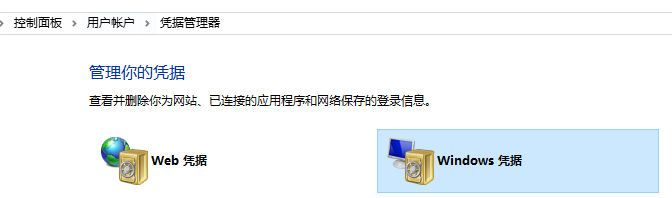


点这里合并代码



添加合并信息





## 6.11 SSH登录

进入当前用户的家目录

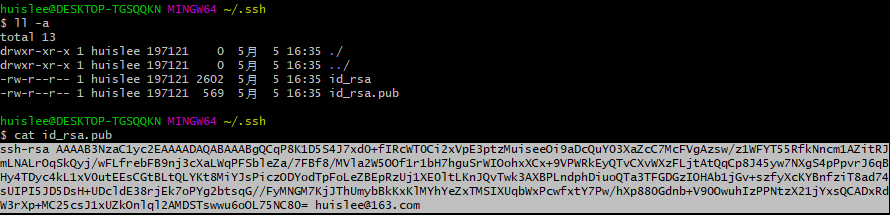
cd~

删除.ssh目录rm-rf. .ssh

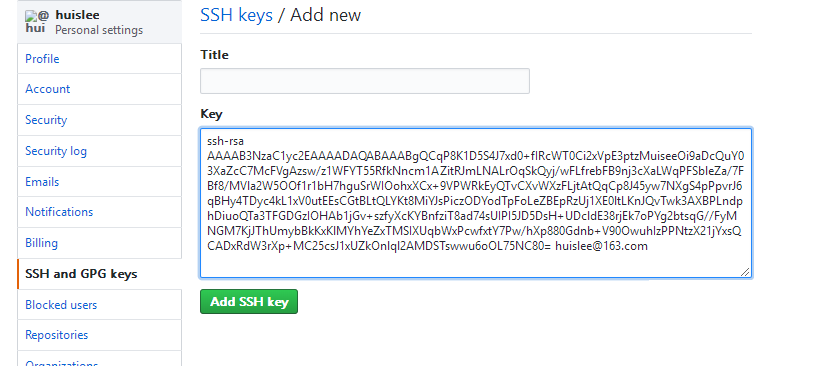
运行命令生成.ssh密钥目录

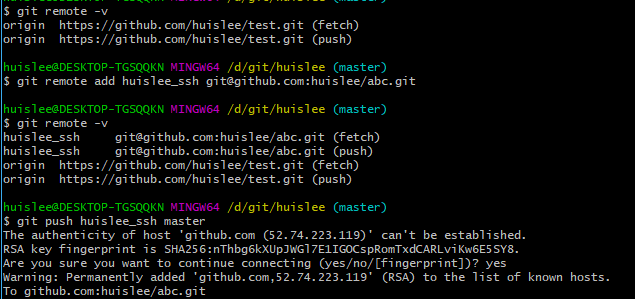
ssh-keygen -t rsa -C huislee@163.com

注意：这里-C这个参数是大写的C



复制 id rsa. pub文件内容，登录 GitHub，点击用户头像→ Settings- SSH and GPG New SSH Key输入复制的密钥信息





进入目录添加别名操作即可。

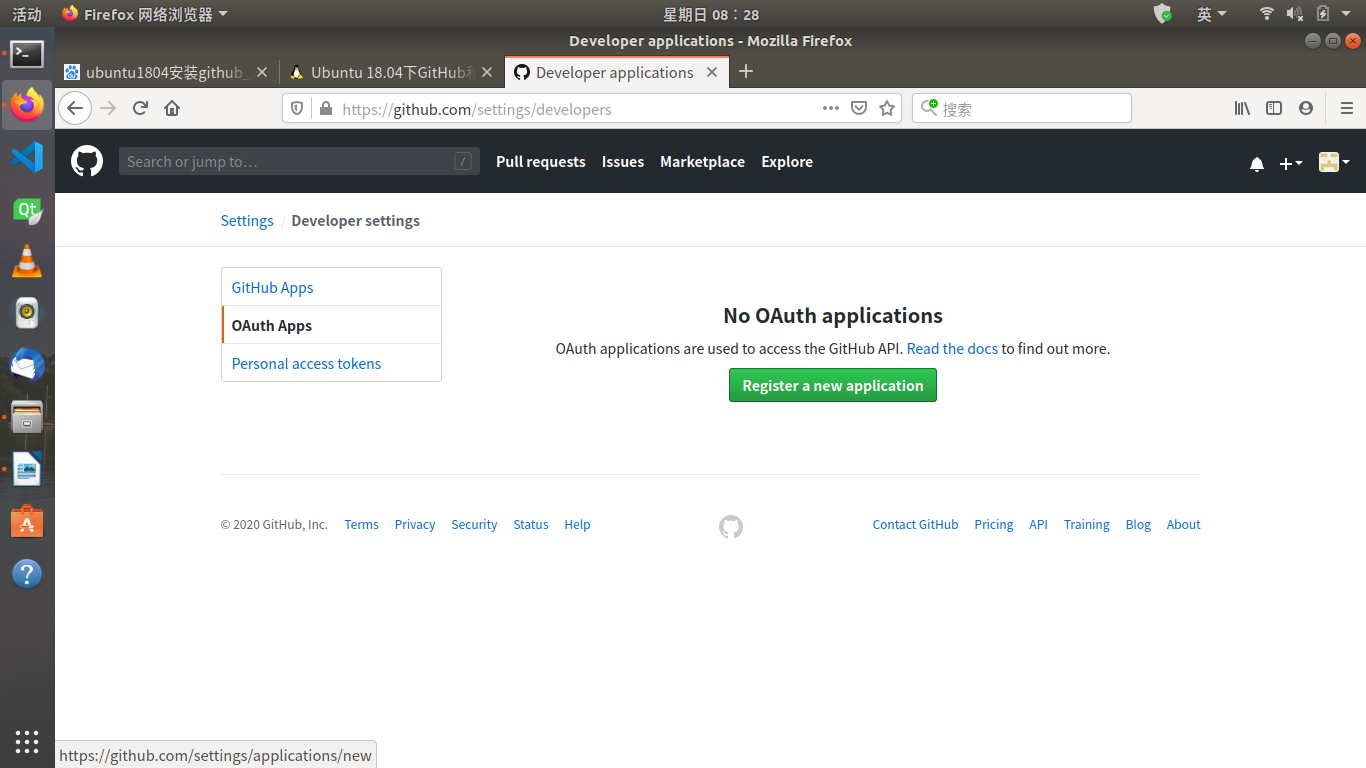
注：通过http时每次push操作都需要输入密码；但是使用ssh就可以设置一次以后直接登录，但缺陷是只能为一个账号设置登录密码。

补充如下：

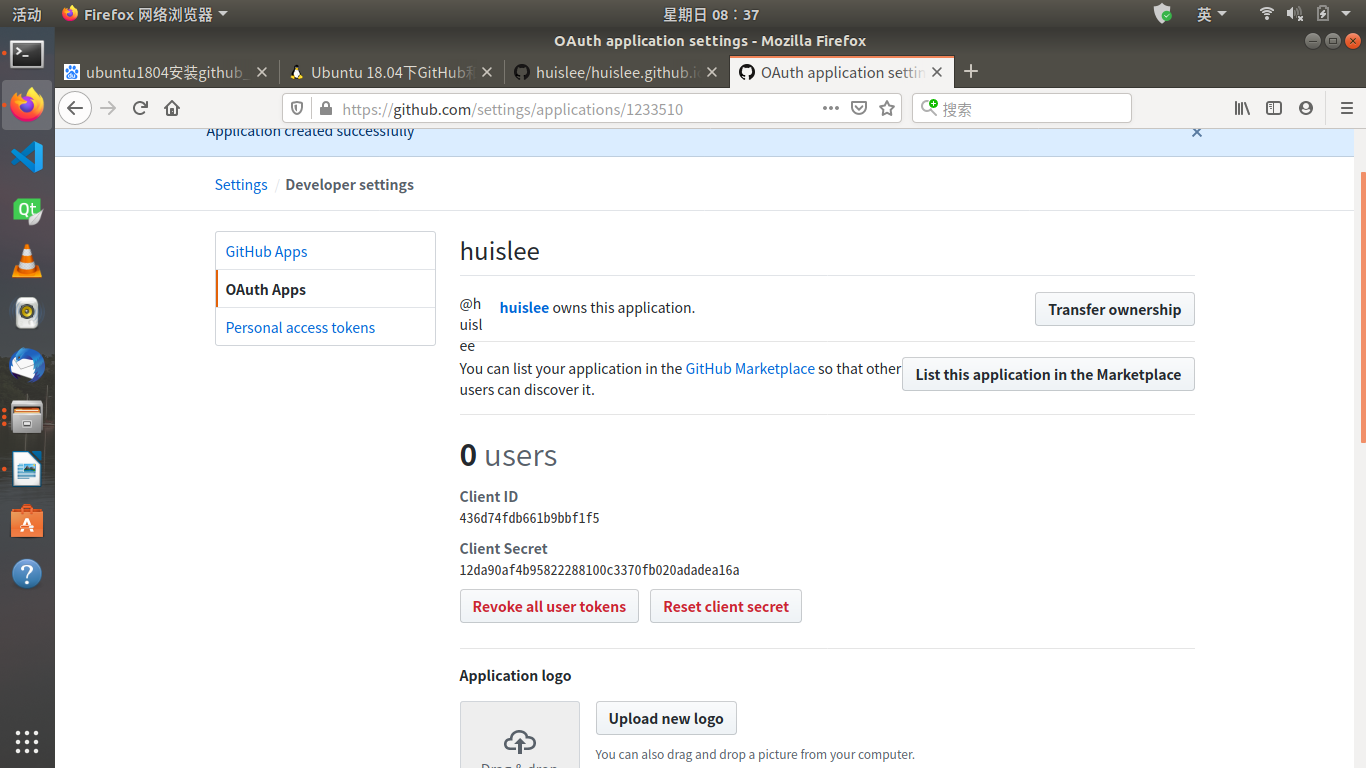
创建一个GitHub开发者应用

1.点击右上角你的头像，在下拉菜单中选择Settings。

2. 在左边的Personal settings(个人设置)中选择OAuth Apps。



3点击上图中的绿色按钮Register applciation注册应用，注册成功后，记住下一页中的Client ID和Client Secret值。



OK，现在你就可以使用此Client ID和Client Secret做GitHub三方登录了。

第二部分：Git相关基础知识

Ubuntu Linux下安装Git很简单

使用命令apt-get来安装 sudo apt install git

配置用户名密码 git config --global user.name "huislee"

git config --global user.email huislee@163.com

此时，Home目录下会新建一个.gitconfig文件

为GitHub账号添加SSH Keys

1.生成Keys

ssh-keygen -t rsa -C "huislee@163.com"

输出如下示例：

huislee@huislee-Lenovo-G410:~$ ssh-keygen -t rsa -C "793612510@qq.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/home/huislee/.ssh/id\_rsa):

/home/huislee/.ssh/id\_rsa already exists.

Overwrite (y/n)? y

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/huislee/.ssh/id\_rsa.

Your public key has been saved in /home/huislee/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:JXm5zrgurBqrOaxB8W0itbNebIVpijVfLjyy+qZ1fjo 793612510@qq.com

The key's randomart image is:

+---[RSA 2048]----+

| |

| . . |

| . . o + |

| + o o + . |

| o B \* oS . |

|. + % + + |

|o..= @ .. o |

|.+o+BE=. . |

|==B=.++oo |

+----[SHA256]-----+

系统会提示key的保存位置（一般是~/.ssh目录）和指定口令，保持默认，连续三次回车即可

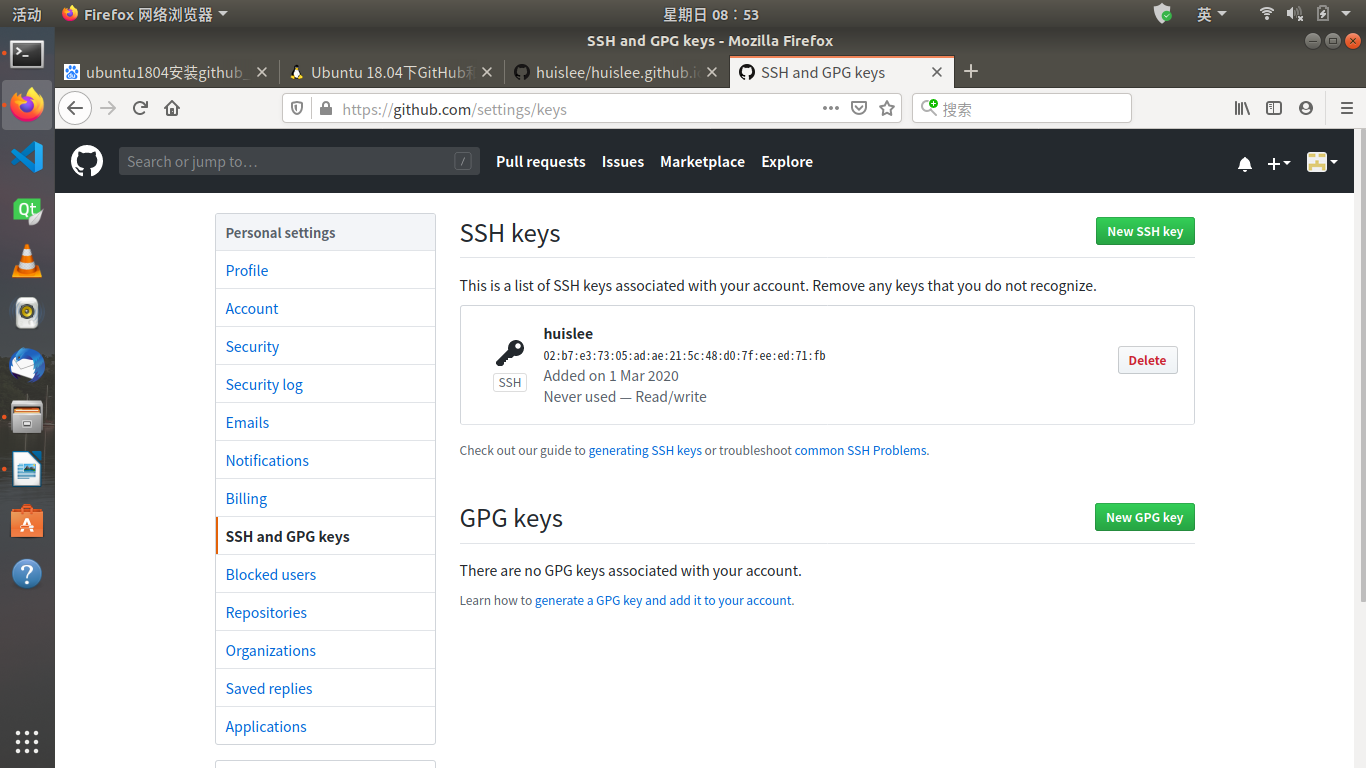
2.复制SSH Key到GITHUB

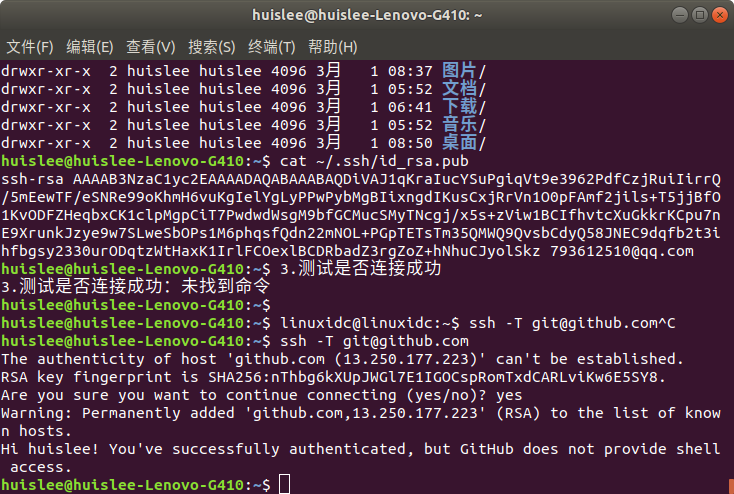
打开该文件，id\_rsa.pub文件内的内容，粘帖到github帐号管理的添加SSH key界面中

cat ~/.ssh/id\_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDiVAJ1qKraIucYSuPgiqVt9e3962PdfCzjRuiIirrQ/5mEewTF/eSNRe99oKhmH6vuKgIelYgLyPPwPybMgBIixngdIKusCxjRrVn1O0pFAmf2jils+T5jjBfO1KvODFZHeqbxCK1clpMgpCiT7PwdwdWsgM9bfGCMucSMyTNcgj/x5s+zViw1BCIfhvtcXuGkkrKCpu7nE9XrunkJzye9w7SLweSbOPs1M6phqsfQdn22mNOL+PGpTETsTm35QMWQ9QvsbCdyQ58JNEC9dqfb2t3ihfbgsy2330urODqtzWtHaxK1IrlFCOexlBCDRbadZ3rgZoZ+hNhuCJyolSkz [793612510@qq.com](mailto:793612510@qq.com)

登录github-> Settings-> SSH and GPG Keys-> New SSH key添加

3.测试是否连接成功 ssh -T git@github.com

****

### 创建一个本地huislee仓库

1. mkdir githuislee，创建一个新文件夹。

2. 输入cd命令进入新文件夹，注意在输入命令时，你可以用Tab键来自动补全。

3. 输入git init初始化Git仓库。此时用ls -a查看当前目录，可以看到多了一个.git/的文件夹，此文件夹保存了版本控制的所有相关信息。

git init

已初始化空的 Git 仓库于 /home/huislee/gitshare/.git/

注意，在此处使用的所有命令，如果你是在Linux环境下开发，用法都是完全一样的。所以对于完全没有Linux使用经验的学员，这也是一个开始接触Linux工作方式的好方法。

接下来，让我们创建一份简单的说明文件，并提交到版本库中。

4. 将需要管理的文件放至目录下。

5. 输入git status查看当前版本库状态，在Untracked files(未跟踪文件)下，会出现红色的文件，代表此文件还未被Git所管理。

6.使用git add 文件或目录，将该文件加入缓冲区，如果你确定所有的修改都需要提交，可以使用git add .来加入所有修改。现在用git status查看，将看到文件名变为绿色。添加上传文件的描述

git commit -m "test" （”test“为分支名）

（创建分支）git branch test

（切换分支）git checkout test

与远程分支相关联：

git remote add origin https://github.com/yangxiaoyan20/BowlingScore.git

（”BowlingScore“ 为工程名）

（将分支上传）git push origin test 注意:提示 “请输入github用户名和密码“

7.使用git commit -m "This is my first commit via Git!"来提交修改，-m后面所带的参数是本次提交信息，一般用来记录本次提交的主要意图。

8. 提交成功后，可以用git log查看历史提交记录。每个记录都会有提交id，作者和提交日期。

9. 你可以用git branch查看当前有哪些分支，当然，因为我们没有创建任何分支，目前只会有一个master分支。

10. 使用git checkout -b feature创建一个名为feature的分支，再用git branch查看一下。

### 项目的下载，查看和修改

第一步. 从GitHub上下载我们的项目代码。

1. 以huislee项目为例，点击绿色按钮Clone or download，然后在弹出窗口中点击剪切板图标，复制仓库的URL。

2. 在git bash中下载项目源码。 git clone -b分支名 URL

第二步. 查看版本历史

1. cd到项目文件夹下，使用git log能看到我们的历史提交记录。

2. 要回到某一历史版本，可以使用git checkout commitId，看完后要回到最新代码，使用git checkout master。

第三步. 本地修改代码

你可以在我们的代码基线上任意修改，但为了下载新代码时不出现冲突，请遵循以下步骤：

1. 下载新代码：git pull。

2. 从master出捡出一个新的分支：git checkout -b feature。feature是分支名称，你可以随意取名，但请用英文。

3. 在feature分支上随意修改，改完后你可以提交你的修改：git commit -m "some message" 。

4. 此时要同步代码，请先切回主分支：git checkout master，然后更新git pull。

5. 如果想删除自己建立的分支，使用git branch -D feature，注意执行此命令后分支被强制删除，无法恢复。

# 8Git工作流

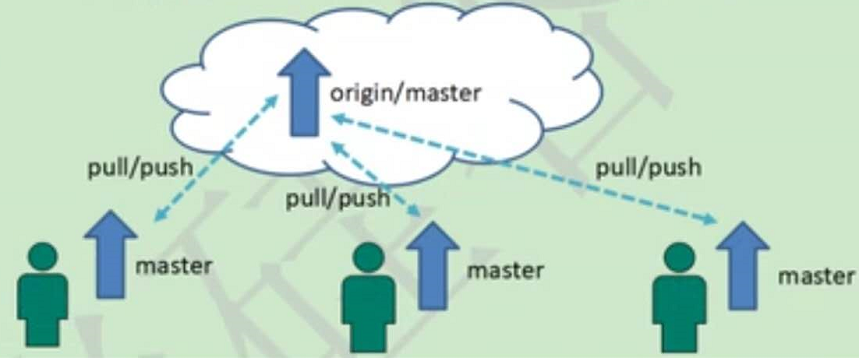
## 8.1概念

在项目开发过程中使用Git的方式

## 8.2分类

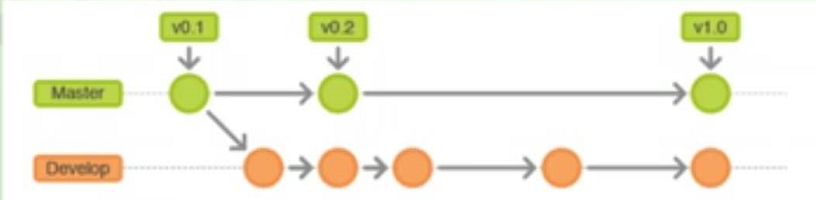
### 8.2.1集中式工作流

像SWN一样，集中式工作流以中央仓库作为项目所有修改的单点实体。所有修改都提交到 Master这个分支上这种方式与SVN的主要区别就是开发人员有本地库，Git很多特性并没有用到。



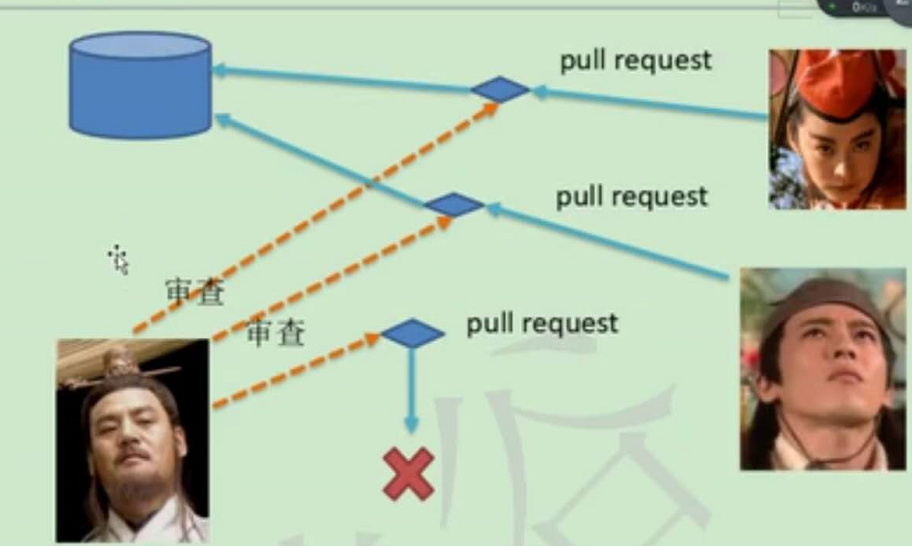
### 8.2.2GitFlow工作流

Gitflow工作流通过为功能开发、发布准备和维护设立了独立的分支，让发布迭代过程更流畅。严格的分支模型也为大型项目提供了一些非常必要的结构。



### 8.2.3 Forking工作流

Forking工作流是在 Git Flow基础上，充分利用了Git的Fork和 pull request的功能以达到代码审核的目的。更适合安全可靠地管理大团队的开发者，而且能接受不信任贡献者的提交.



## 8.3 GitFlow工作流详解

### 8.3.1分支种类

主干分支 master

主要负责管理正在运行的生产环境代码。水远保持与正在运行的生产环境完全一致.

开发分支 develop

主要负责管理正在开发过程中的代码，一般情况下应该是最新的代码.

bug修理分支 hotfix

主要负责管理生产环境下出现的紧急修复的代码。从主干分支分出，修理完毕并测试上线后，并回主干分支，并回后，视情况可以删除该分支.

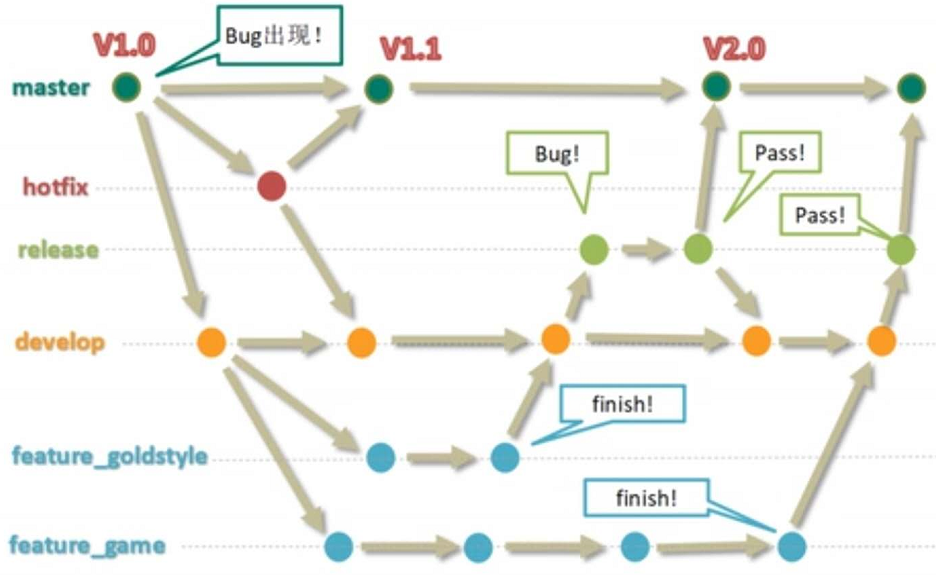
准生产分支（预发布分支） release

较大的版本上线前，会从开发分支中分出准生产分支，进行最后阶段的集成测试，该版本上线后，会合并到主干分支，生产环境运行一段阶段较稳定后可以视情况删除.

功能分支feature

为了不影响较短周期的开发工作，一般把中长期开发模块，会从开发分支中独立出来。开发完成后会合井到开发分支.

### 8.3.2GFow工作流举例



### 8.3.3 分支实战

