**Git&GitHub**

笔记也可参考改网站：https://nanzx.top/posts/629e/本笔记根据哔哩哔哩上尚硅谷视频自学总结网站链接：https://www.bilibili.com/video/BV1pW411A7a5?p=1

# 1.版本控制

## 版本控制介绍

### 个人开发改进迭代

### 团队协作

## 版本控制工具介绍

## 版本控制工具应该具备的功能

1.协同修改多人并行不悖的修改服务器的同一个文件2.数据备份不仅保存目录和文件状态，还能够保存每一个提交过的历史状态。3.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的数据，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。4.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。5.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。6.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。

## 版本控制简介

### 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迭代。

### 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：CVS、SVN、VSS......分布式版本控制工具：Git、Mercurial、Bazaar、Darcs......

# 2.Git简介

## 1.Git简义

Linux系统版本控制历史1991年：Linus本人手动合并代码2002年：商业软件：BitKeeper（BitKeeper的东家BitMover公司处于人道主义精神，授权Linux社区免费使用这个版本控制系统。但要求不能进行破解。）2005年：开发Samba的Andrew试图破解BitKeeper的协议，被BitMover公司发现，要收回Linux社区的免费使用权。（Linux社区无法像商业公司那样对参与开发者进行强有力的约束）Linux自己用C语言开发了一个分布式版本控制系统：Git主体程序开发完成只用了两周一个月后Linux系统代码由Git管理2008年：GitHub上线（JQuery、Ruby、PHP、...）

## 2.Git官网和Logo

### 官网地址：https://git-scm.com/

### 



## 3.优势

### 大部分操作在本地完成，不需要联网

### 完整保证性（通过hash操作）

### 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Linux命令全面兼容

## 4.Git安装

## 5.Git结构

### 本地结构

* 工作区（写代码）
* 暂存区（临时存储）
* 本地库（历史版本）

## 6.Git和代码托管中心

### 局域网环境下

* GitLab服务器

### 外网环境下

* GitHub
* 码云

### 代码托管中心的任务：维护远程库

## 7.本地库和远程库

### 团队内部写作

### 跨团队协作

# 3.Git命令行操作

## 本地库操作和远程库操作

### 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

### 2.设置签名

* 形式

• 用户名

• liming

• Email地址

• 1352530896@qq.com

* 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

* 命令

• 项目级别/仓库级别

• 仅在当前本地库范围内有效

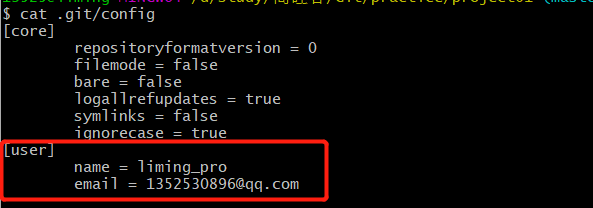
• git config

• git config user.name liming\_pro

• git config user.email 1352530896@qq.com

• 信息保存的位置：./.git/config文件

•



• 系统用户级别

• 登录当前操作系统的用户范围

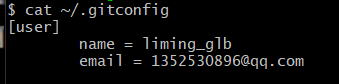
• git config --global

• git config --global user.name liming\_pro

• git config --global user.email 1352530896@qq.com

• 信息保存的位置：~/.gitconfig文件

•



• 级别优先级

• 就近原则

• 项目级别优先与系统用户级别，二者都有时，采用项目级别签名

• 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

### 3.基本操作

* 1.状态查看操作

• git status

• 查看工作区、暂存区状态

* 2.添加操作

• git add [file name]

• 将工作区的“新建/修改”添加到暂存区

* 3.提交操作

• git commit file name -> 然后输入提交信息

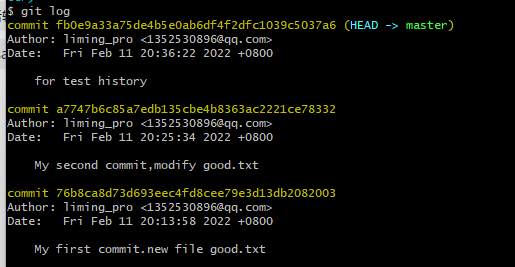
• git commit -m "commit message" file name

• 将暂存区的内容提交到本地库

* 4.查看历史记录操作

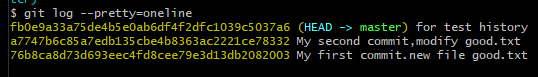
• git log

•



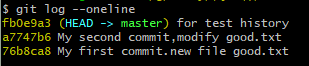
• git log --pretty=oneline

•



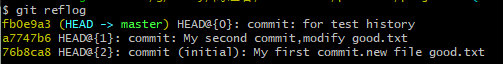
• git log --oneline

•



• git reflog

•



• HEAD@{移动到当前版本需要多少步}

* 5.前进后退

• 基于索引值操作[推荐]

• git reset --hard [局部索引值]

• git reset --hard 76b8ca8

• 基于^符号

• 只能后退

• git reset --hard HEAD^^

• 注：几个^代表后退几步

• 基于~符号

• 只能后退

• git reset --hard HEAD~n

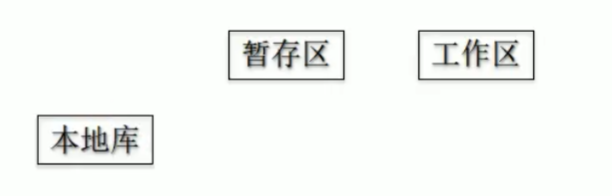
• 注：n表示后退n步

* 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

• 本地库向后移动，相当于暂存区领先一个状态，这时候使用git status命令会显示modified状态



• --mixed参数

• 在本地库移动HEAD指针

• 重置暂存区

•



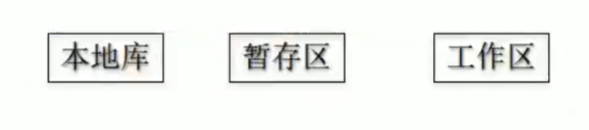
• --hard参数

• 在本地库移动HEAD指针

• 重置暂存区

• 重置工作区

•



• 重置当前本地库、暂存区和工作区命令：git reset --hard HEAD

* 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

* 8.比较文件差异

• git diff [文件名]

• 将工作区中的文件和暂存区的文件进行比较

• 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

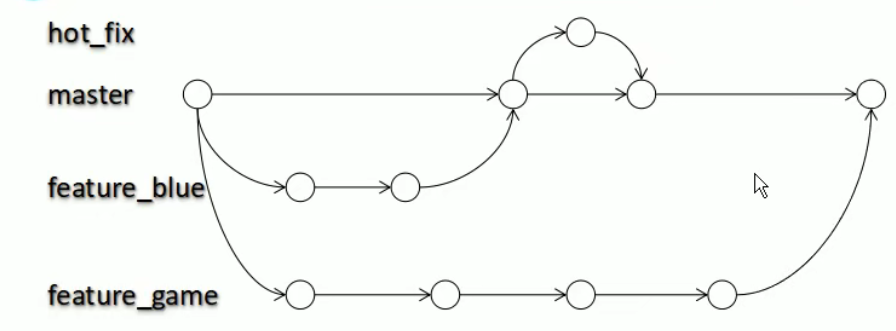
• 不带文件名比较多个文件

### 4.分支管理

* 1.什么是分支

• 在版本控制过程中，使用多条线同时推进多个任务。

•



* 2.分支的好处

• 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。

* 3.分支操作

• 创建分支

• git branch [分支名]

• 查看分支

• git branch -v

• 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上（被合并，增加新内容）上

• git checkout [被合并分支名]

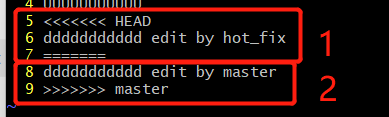
• 第二步：执行merge命令

• git merge [有新内容的分支名]

• 解决冲突

• 冲突的表现

• 1.当前分支内容2.另一分支内容



• 冲突的解决

• 第一步：编辑文件，删除特殊符号

• 第二部：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"

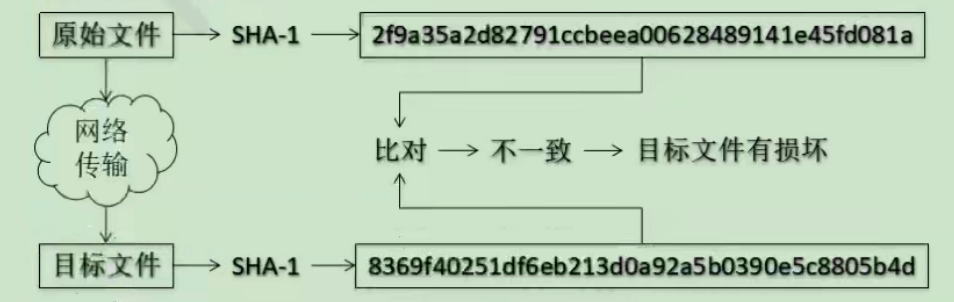
• 注意：此时commit一定不能带具体文件名

### 5.Git基本原理

* 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

• Git就是靠这种机制来从根本上保证数据完整性的

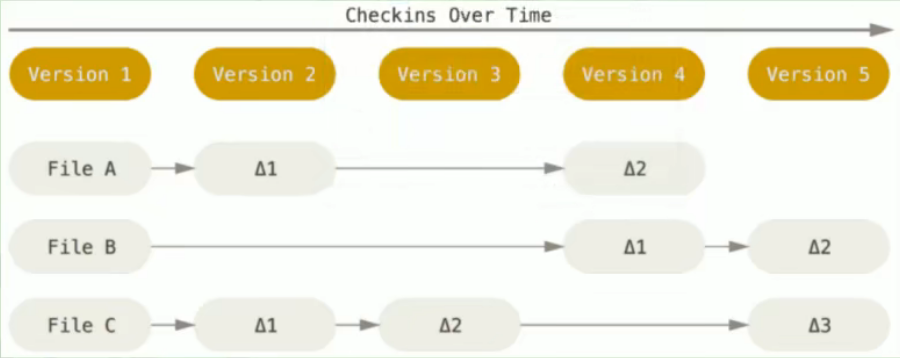


* 2.Git保存版本的机制

• 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。

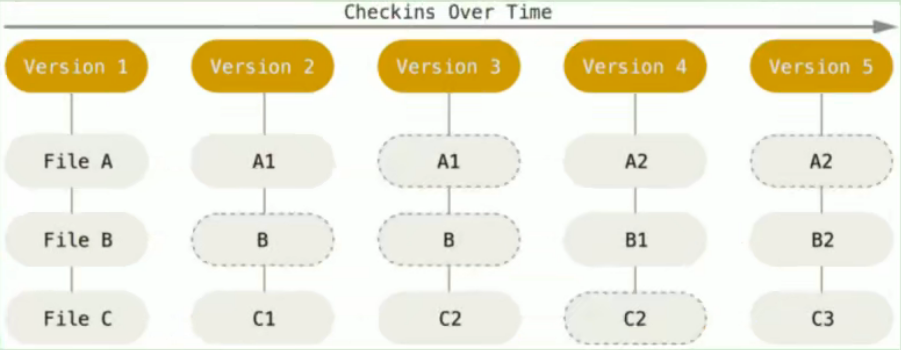
•



• 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不在重新存储该文件，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。

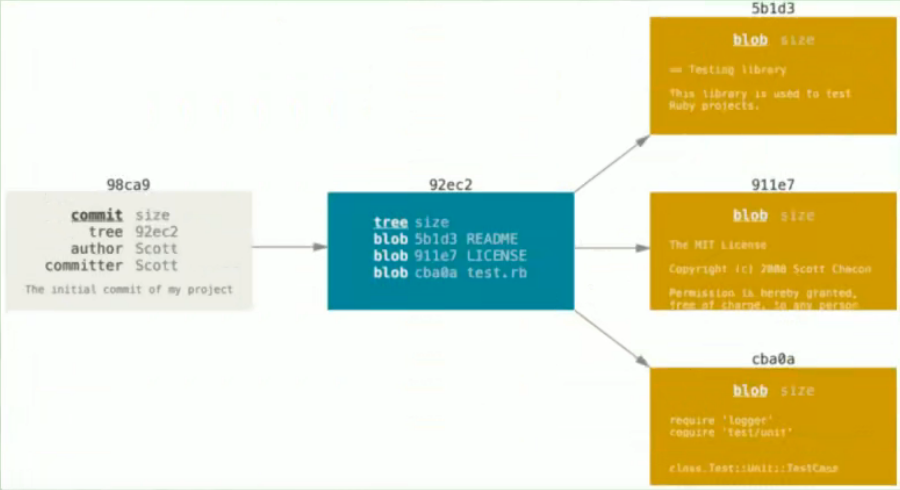
•



• 3.Git文件管理机制细节

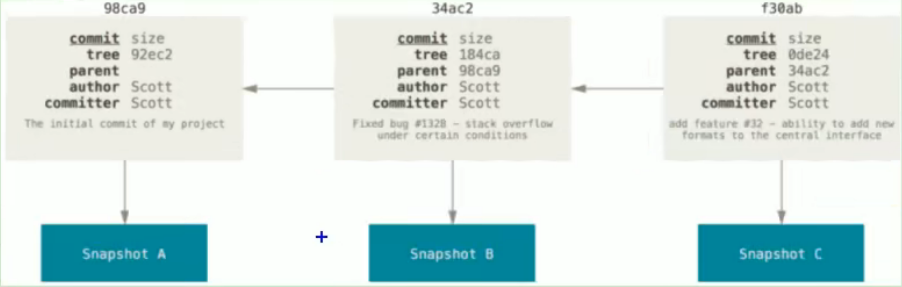
• Git的“提交对象”

•



• 提交对象及其父对象形成的链条

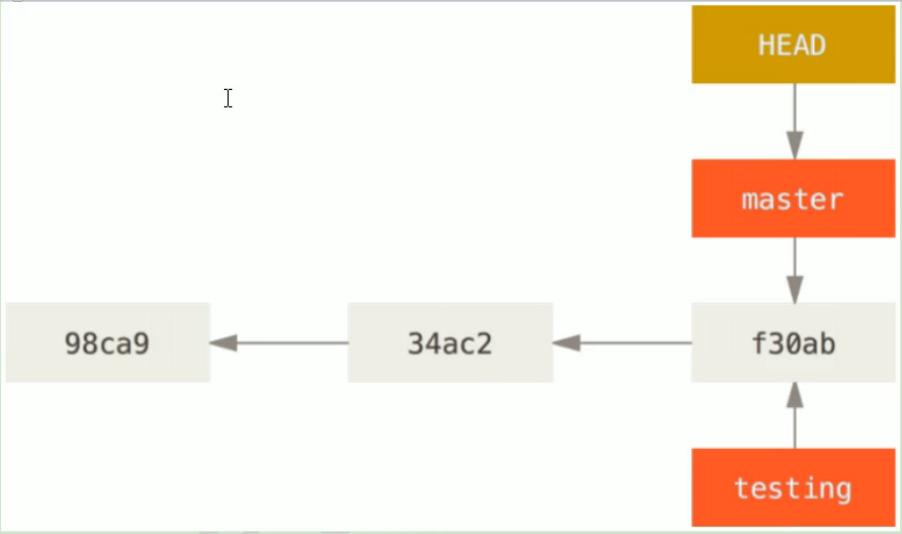
•



* 3.Git分支管理机制

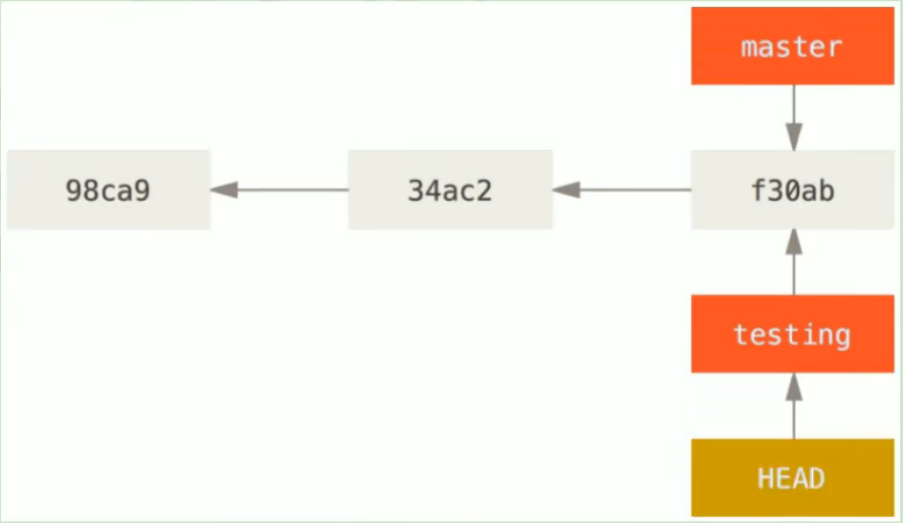
• 1.分支的创建

•

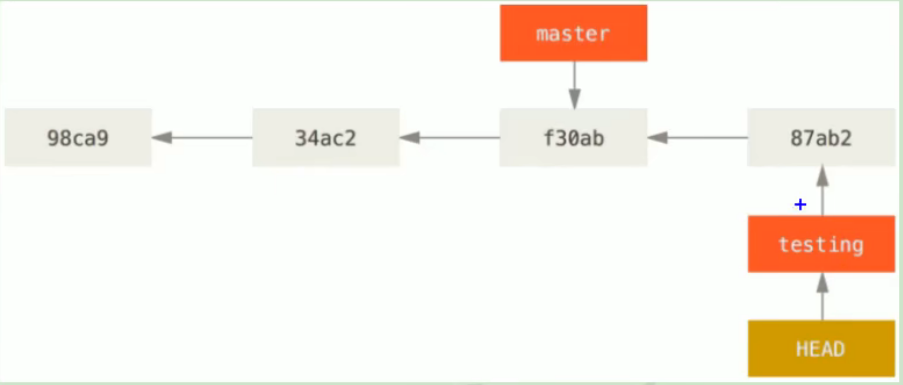


• 2.分支的切换

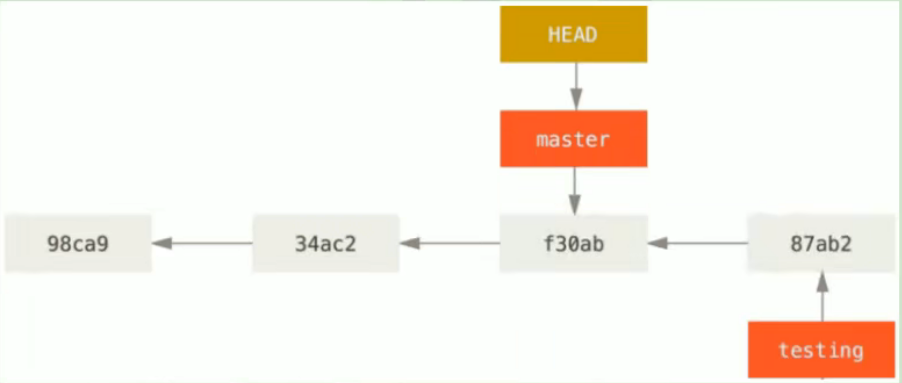
•



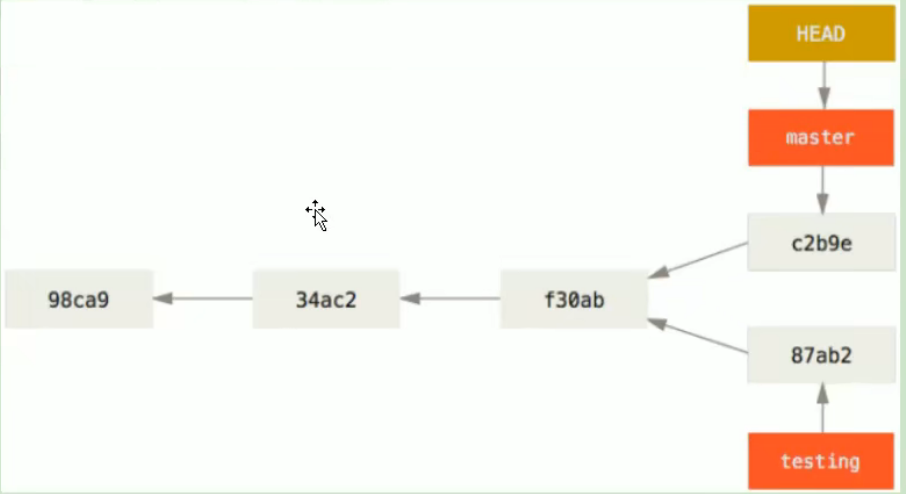
•



•



•



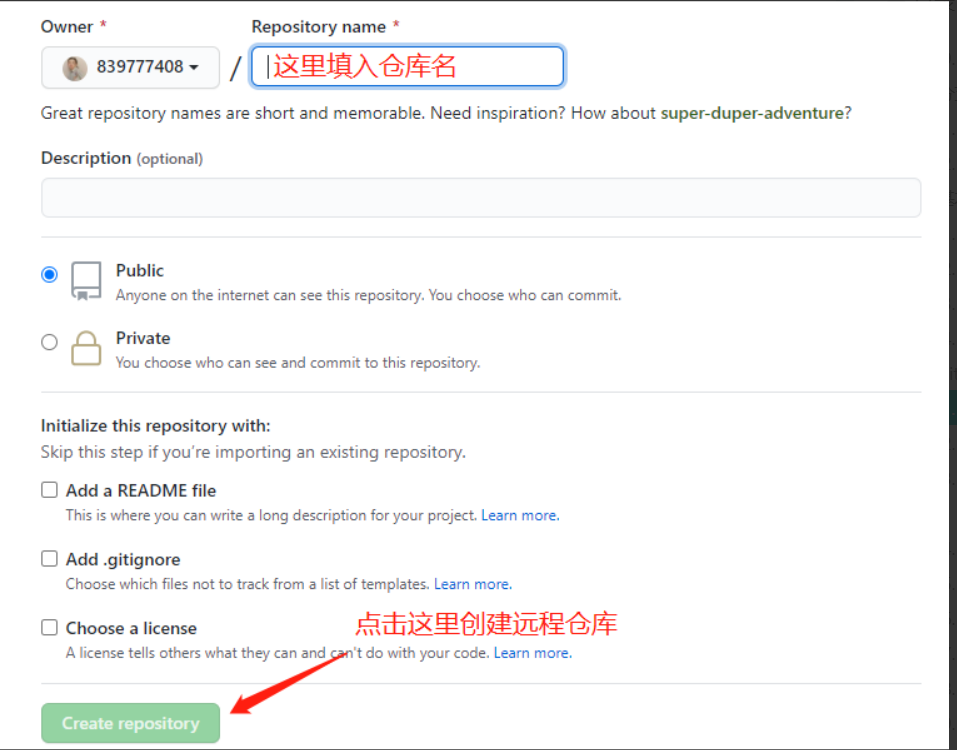
### 6.GitHub

* 1.账号信息

• 1352530896@qq.com

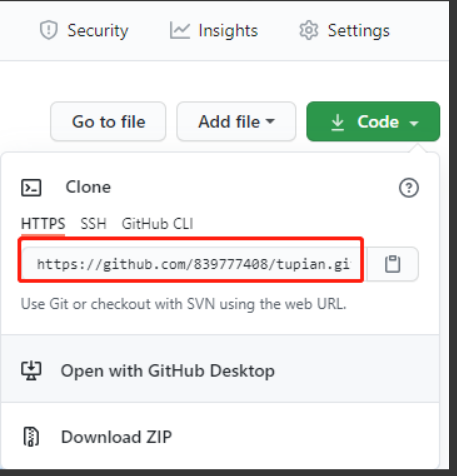
* 2.创建远程仓库

•



* 3.创建远程仓库地址

•

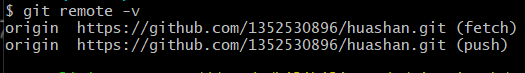


* 4.别名

• 查看当前所有远程地址别名

• git remote -v

•



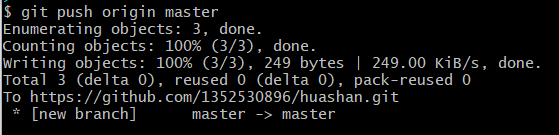
• 添加远程地址别名

• git remote add [别名] [远程地址]

* 5推送

• git push [别名] [分支名]

•



* 6.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

• 3.初始化本地库

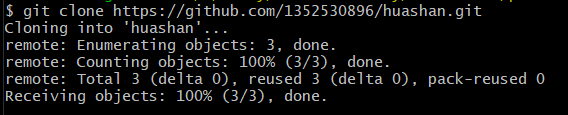
• 命令

• git clone [远程地址]

• git clone <-b 分支名> [远程地址] <本地目录名>

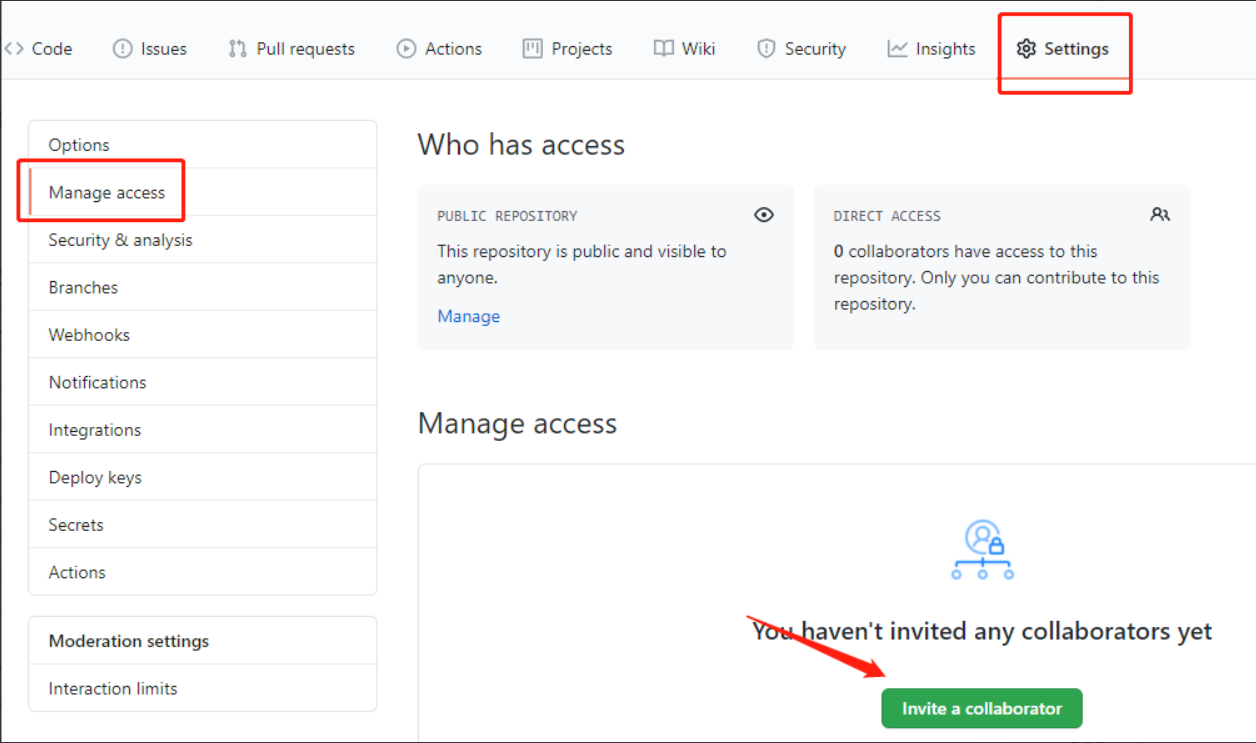
• <>表示可选，默认克隆master分支，本地目录名称与版本库同名。

•



* 7.团队成员邀请

•



* 8.拉取

• pull=fetch+merge

• git fetch [远程库地址别名] [远程分支名]

• git merge [远程地址别名/远程分支名]

• git pull [远程地址别名] [远程分支名]

* 9.解决冲突

• 要点

• 1.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。

• 2.拉去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。

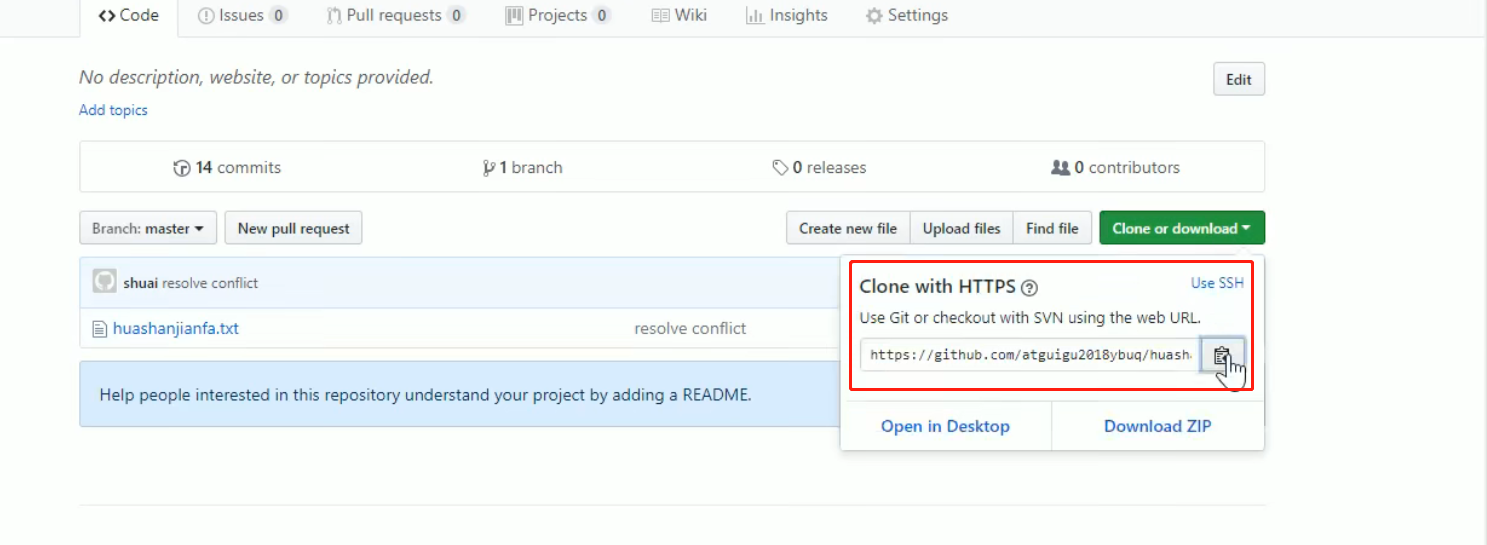
• 老王媳妇说：5天后归还。小刘不能接受。老王媳妇需要找老王确认后再执行。

* 10.跨团队协作

• 操作步骤

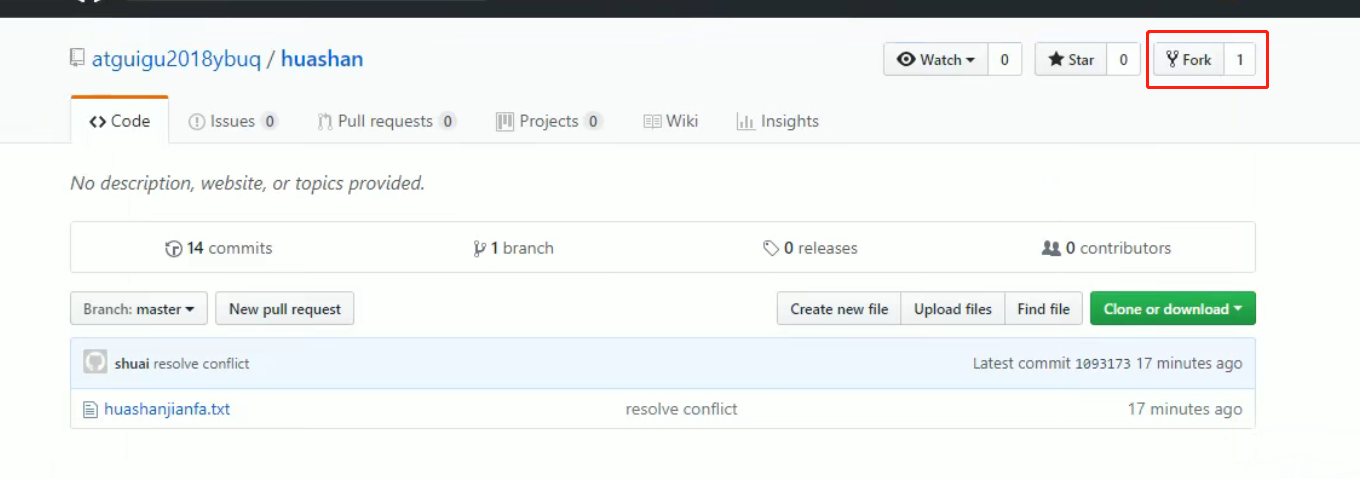
• 复制链接

•



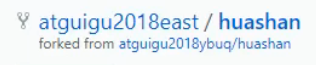
• 登录其他账号后，地址栏输入复制的链接，然后点击fork

•



• fork后的项目显示如图

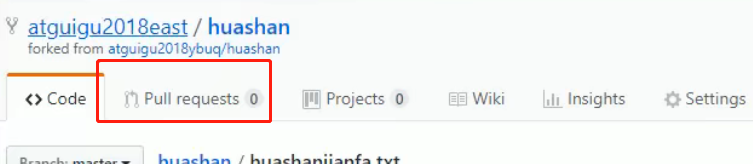
•



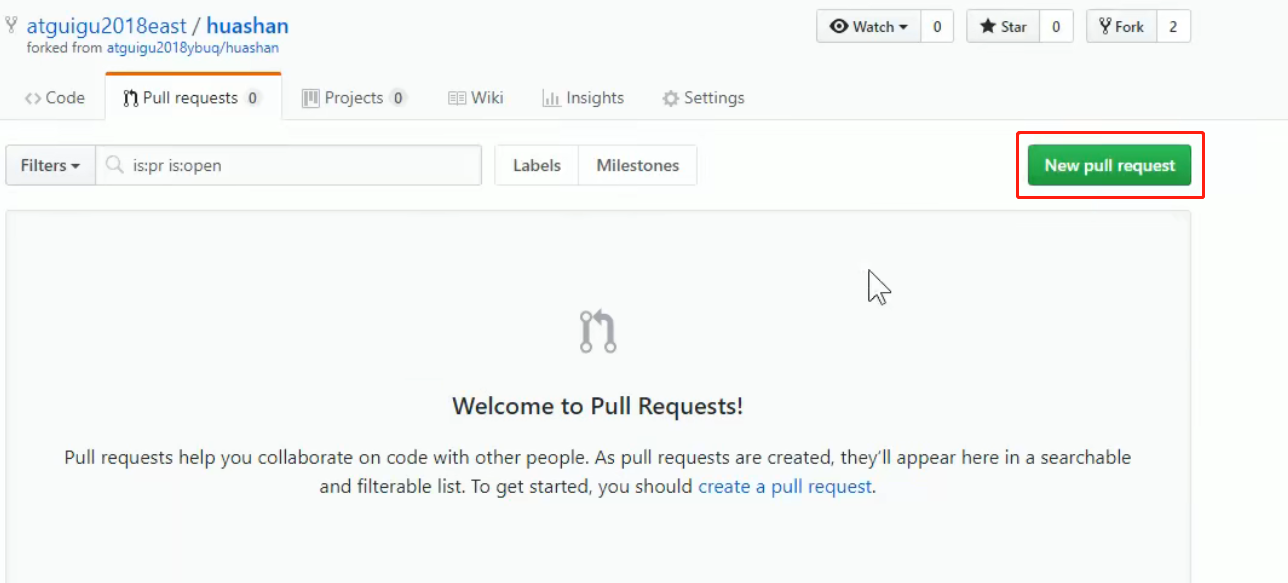
• 本地修改，然后推送到远程

• Pull Request

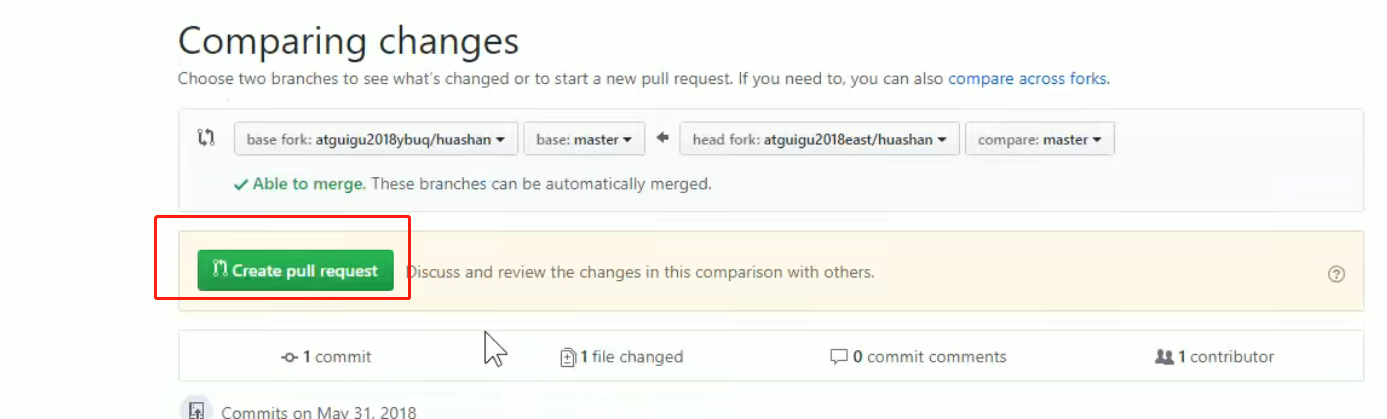
•



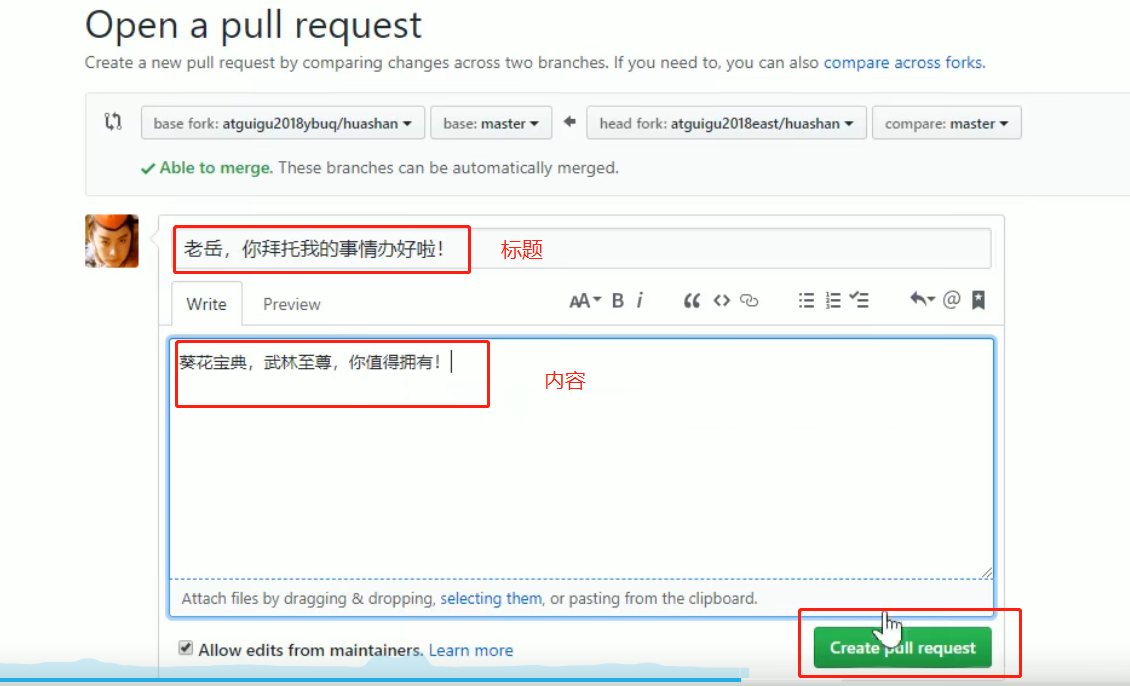
•



•

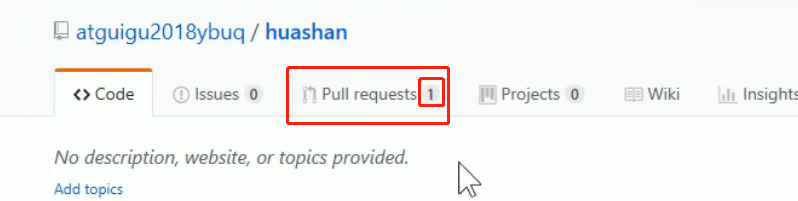


•

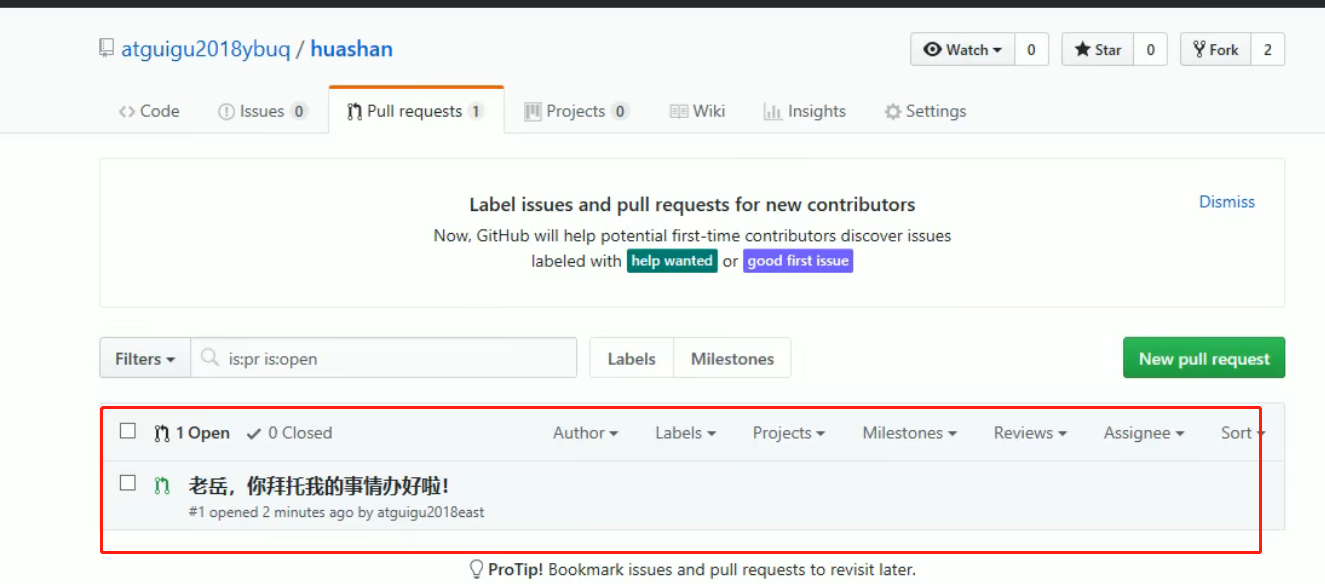


• 然后切换原来账号

•

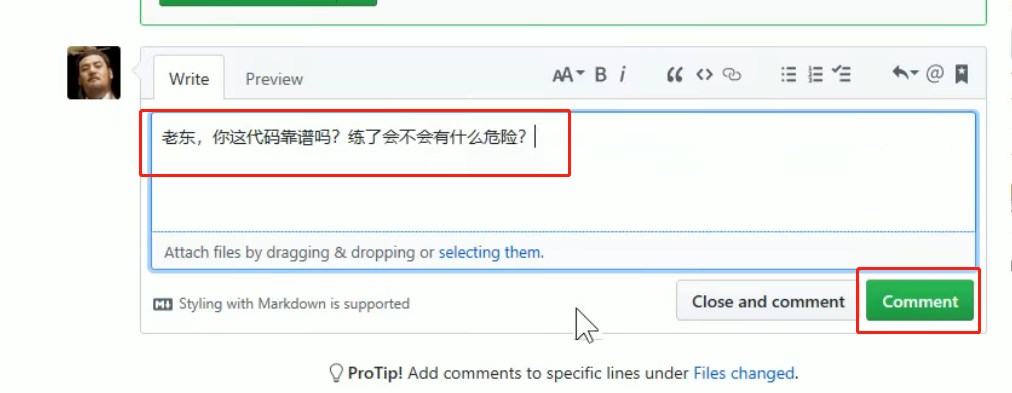


•

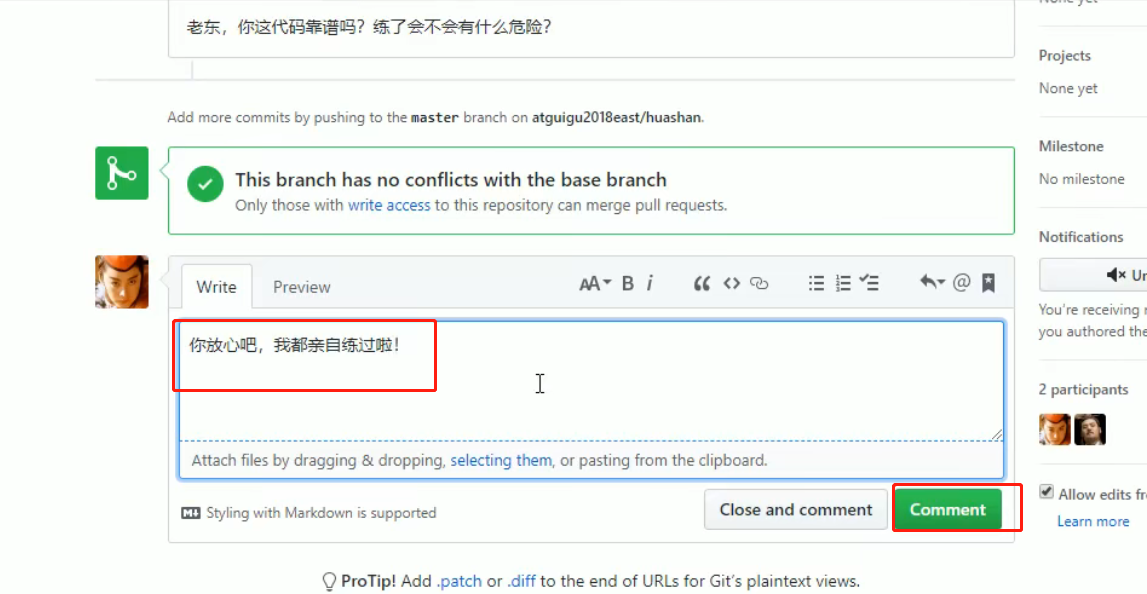


• 对话

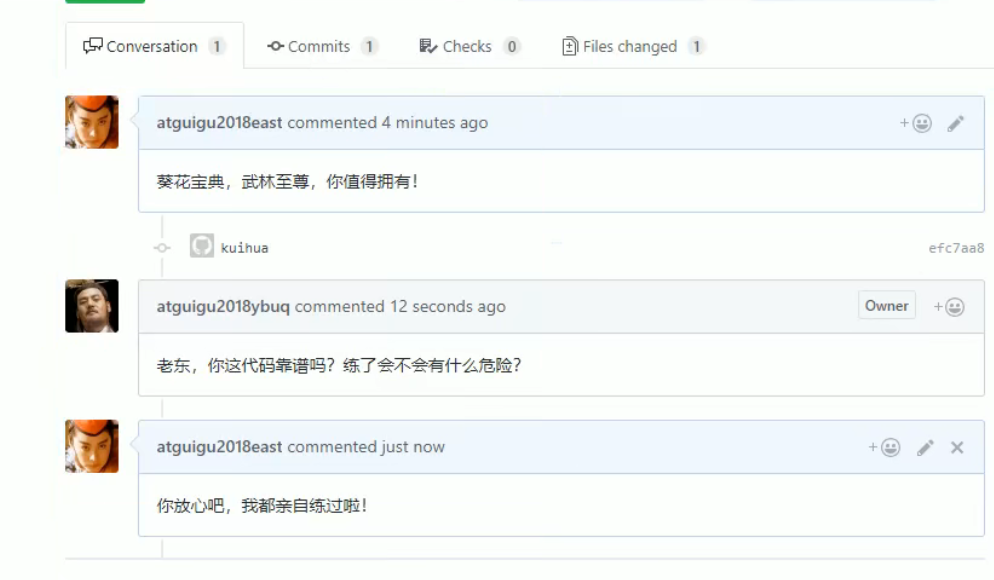
•



•



•



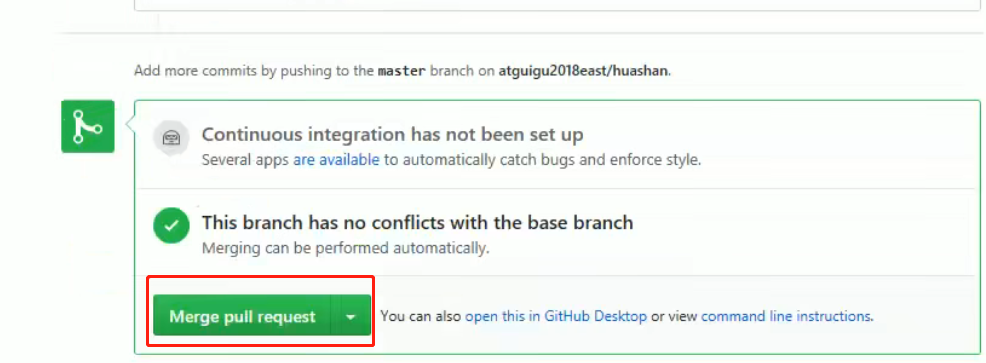
• 审核代码

•



• 合并

•



•



• 将远程库修改拉取到本地

* 11.SSH登录

Windows10和Windows11自动帮我们保存凭据

• 进入当前用户的家目录

• cd ~

•

desc

• 删除.ssh目录

• rm -rf .ssh/

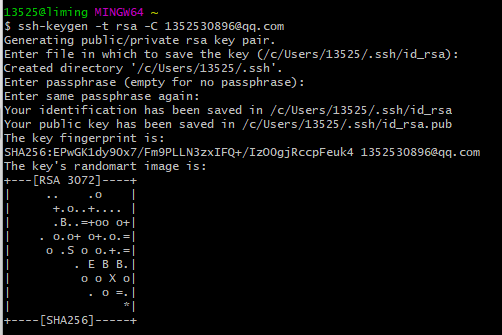
•

desc

• 运行命令生成.ssh秘钥登录

• ssh-keygen -t rsa -C [邮箱]

•

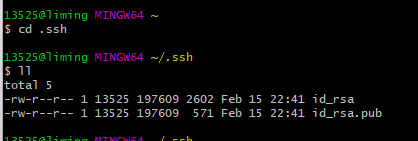


• 输入命令后一直回车，走默认值

• 进入.ssh目录查看文件列表

• cd .ssh

•

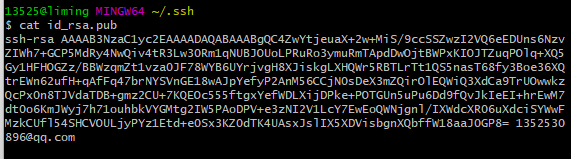


• ll

• 查看id\_rsa.pub文件内容

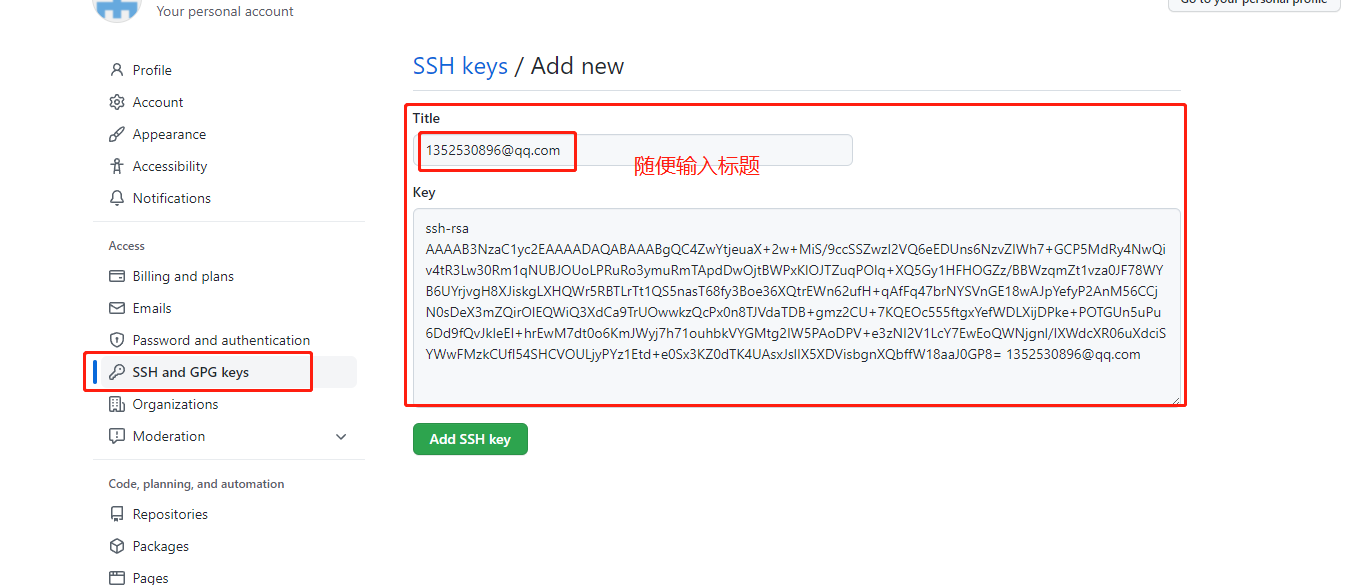
• cat id\_rsa.pub

•



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

•

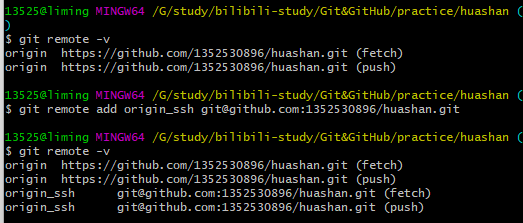


• New SSH Key

• 输入复制的秘钥信息

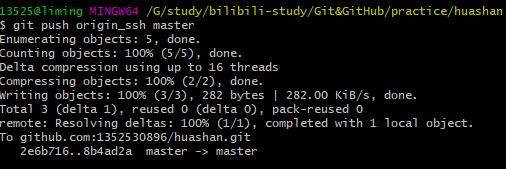
• 回到Git bash创建远程地址别名

•



• 推送的时候使用ssh别名推送，就不需要登录

•



# 4.Git图形化界面操作

## Eclipse的Git操作

### 详细内容见：https://www.bilibili.com/video/BV1pW411A7a5?p=43&spm\_id\_from=pageDriver

# 5.Git工作流

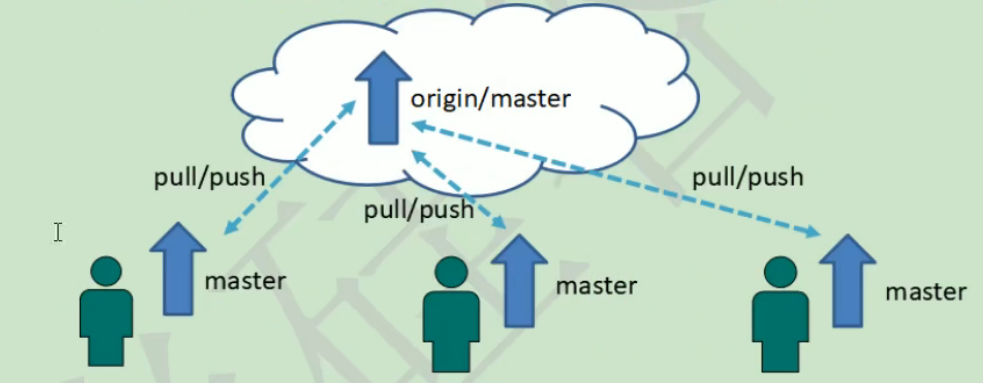
## 1.概念

### 在项目开发过程中使用Git的方式

## 2.分类

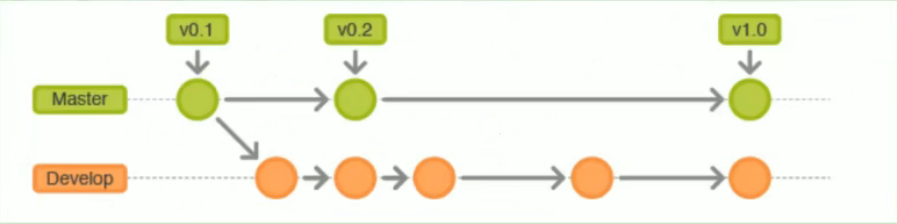
### 1.集中式工作流

像SVN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。



### 2.GitFlow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。



* 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。永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，视情况可以删除改分支。

• 准生产分支（预发布分支）release

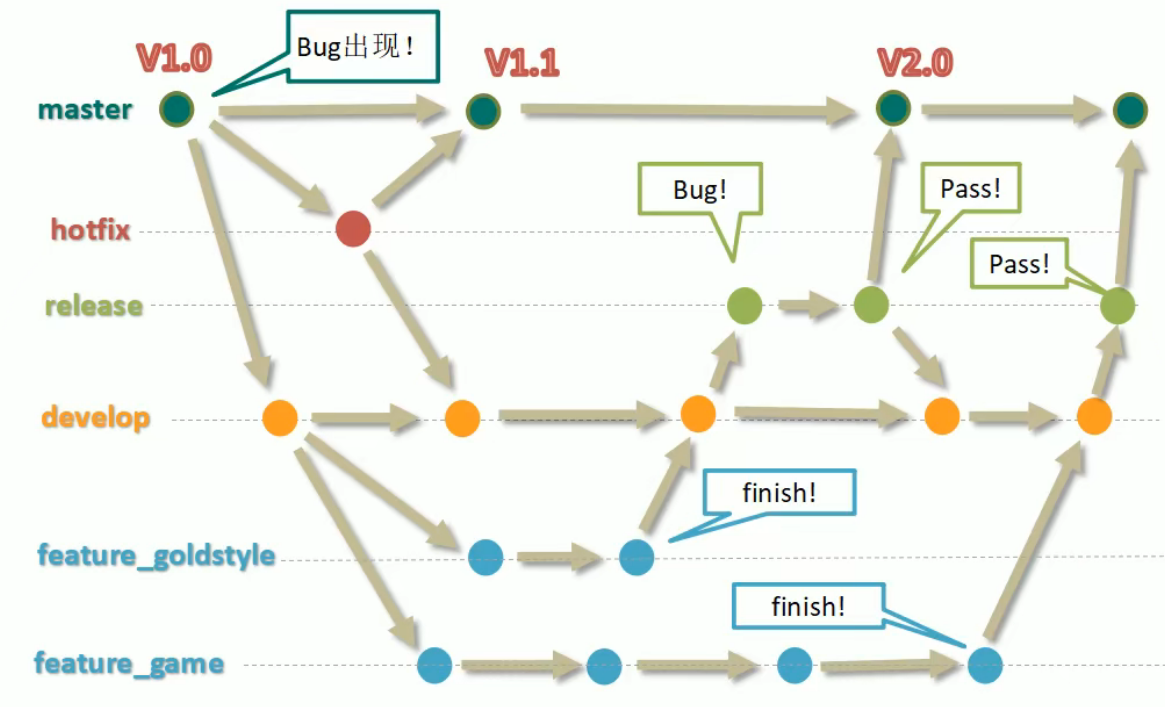
较大的版本上线前，会从开发分支中分出准生产分支，进行最后阶段的集成测试。该版本上线后，会合并到主干分支。生产环境运行一阶段较稳定后可以视情况删除。

• 功能分支feature

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

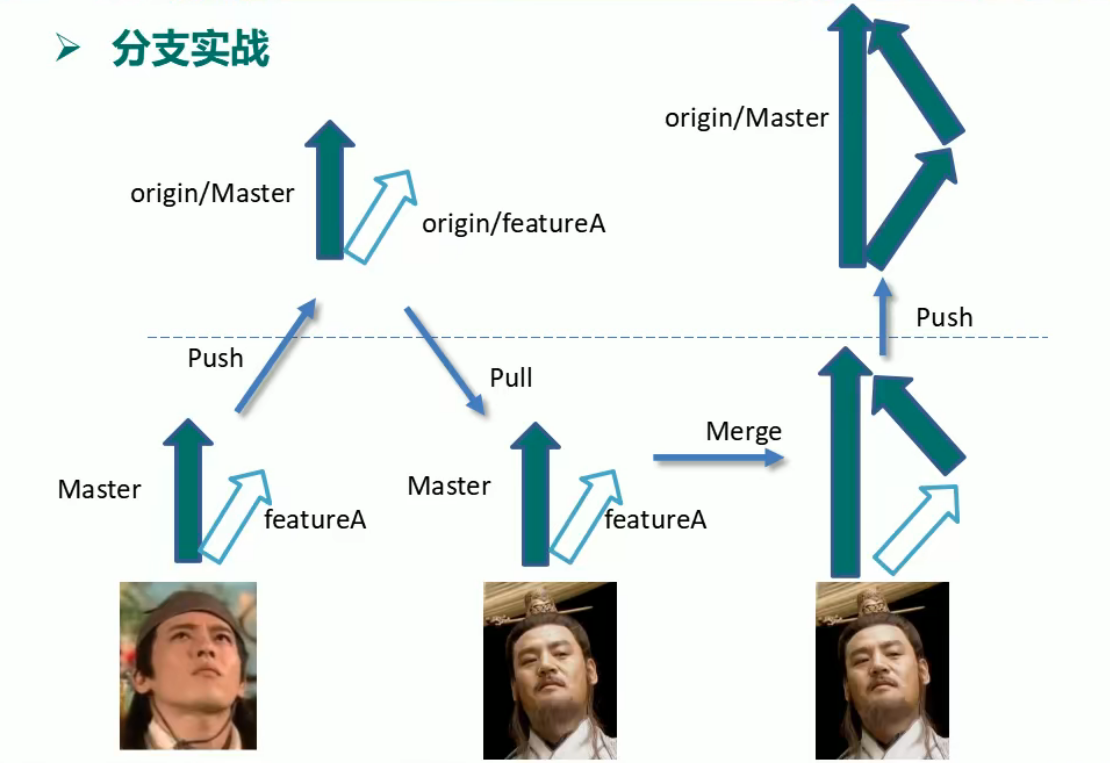
* 2.GitFlow工作流举例

•



* 3.分支实战

•



• 具体操作

• 1.创建分支

• 2.修改内容并提交推送

• 3.另一个人员切换分支审查代码

• 4.检出远程分支测试

• 5.切换回master

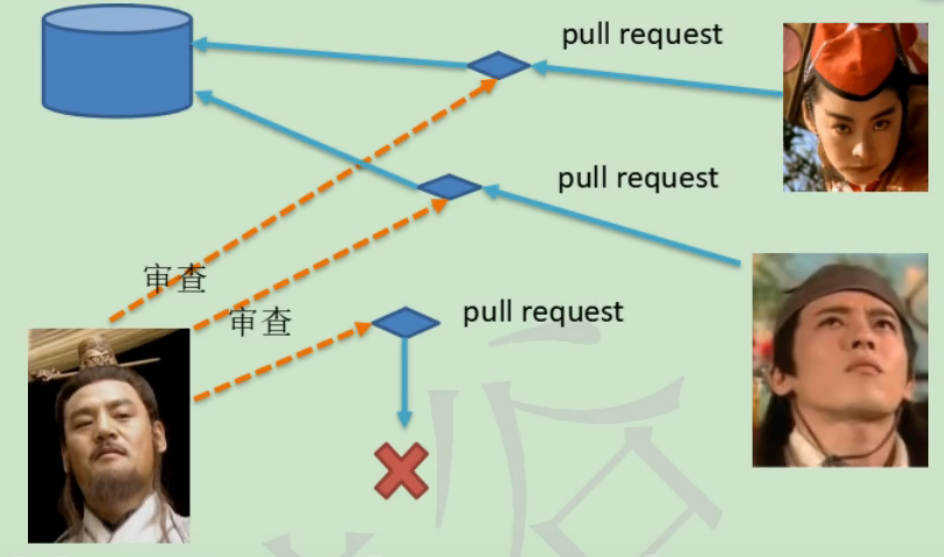
• 6.合并分支

• 7.合并结果，有冲突解决冲突

• 8.合并成功后推送至远程分支

### 3.Forking工作流

Forking工作流是在GitFlow基础上，充分利用了GIt和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。



# 6.Gitlab服务器环境搭建

## 1.官网地址

### 首页

* https://about.gitlab.cn/

### 安装说明

* https://about.gitlab.cn/install/

## 2.安装步骤

### 1.安装和配置必须的依赖项

* 在 CentOS 7上，下面的命令也会在系统防火墙中打开 HTTP、HTTPS 和 SSH 访问。这是一个可选步骤，如果您打算仅从本地网络访问极狐GitLab，则可以跳过它。

• sudo yum install -y curl policycoreutils-python openssh-server perlsudo systemctl enable sshdsudo systemctl start sshdsudo firewall-cmd --permanent --add-service=httpsudo firewall-cmd --permanent --add-service=httpssudo systemctl reload firewalldsudo firewall-cmd --permanent --add-service=httpsudo firewall-cmd --permanent --add-service=httpssudo systemctl reload firewalld

* （可选）下一步，安装 Postfix 以发送电子邮件通知。如果您想使用其他解决方案发送电子邮件，请跳过此步骤并在安装极狐GitLab 后配置外部 SMTP 服务器。

• sudo yum install postfixsudo systemctl enable postfixsudo systemctl start postfix

* 在安装 Postfix 的过程中可能会出现一个配置界面，在该界面中选择“Internet Site”并按下回车。把“mail name”设置为您服务器的外部 DNS 域名并按下回车。如果还有其它配置界面出现，继续按下回车以接受默认配置

### 2.下载极狐GitLab 安装包并安装极狐GitLab

* 下载极狐GitLab 安装包。

• wget https://omnibus.gitlab.cn/el/7/gitlab-jh-14.7.2-jh.0.el7.x86\_64.rpm

* 接下来，安装极狐GitLab。确保您已正确设置您的 DNS，并更改 https://gitlab.example.com 为您要访问极狐GitLab 实例的 URL。安装包将在该 URL 上自动配置和启动极狐GitLab。建议将极狐GitLab 实例的域名以环境变量的形式注入（请将命令中 URL 替换为您要访问极狐GitLab 实例的 URL）：

• export EXTERNAL\_URL=https://gitlab.example.com

* 对于 https 站点，极狐GitLab 将使用 Let's Encrypt 自动请求 SSL 证书，这需要有效的主机名和入站 HTTP 访问。您也可以使用自己的证书或仅使用 http://（不带s）。如果您想为初始管理员用户(root)指定自定义密码，请查看文档。如果未指定密码，将自动生成随机密码。接着执行如下命令开始安装：

• sudo rpm -Uvh gitlab-jh-14.7.2-jh.0.el7.x86\_64.rpm

### 3.访问极狐GitLab 实例并登录

* 除非您在安装过程中指定了自定义密码，否则将随机生成一个密码并存储在 /etc/gitlab/initial\_root\_password 文件中(出于安全原因，24 小时后，此文件会被第一次 gitlab-ctl reconfigure 自动删除，因此若使用随机密码登录，建议安装成功初始登录成功之后，立即修改初始密码）。使用此密码和用户名 root 登录。

### 4.后续配置请参考网站

* https://docs.gitlab.cn/jh/install/next\_steps.html

### 也可打开官网安装说明进行安装

## 3.gitlab服务操作

### 初始化配置gitlab

* gitlab-ctl reconfigure

### 启动gitlab服务

* gitlab-ctl start

### 停止gitlab服务

* gitlab-ctl stop

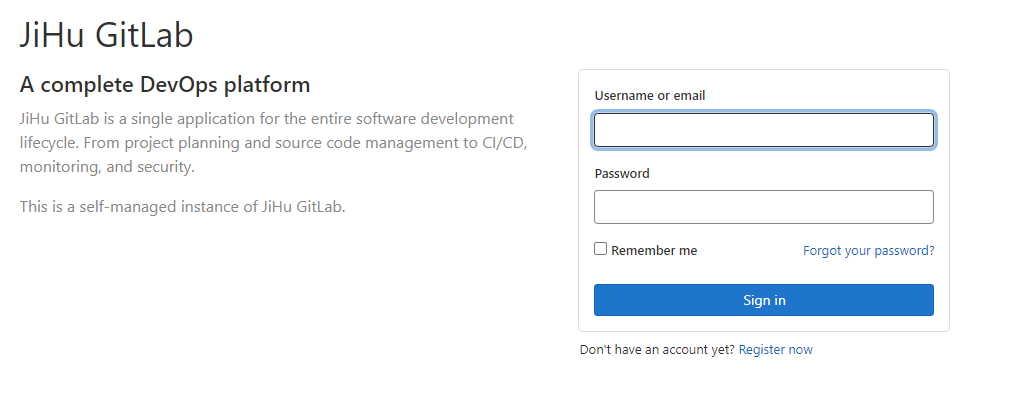
## 4.浏览器访问

### 访问Linux服务器IP地址即可，如果想访问EXTERNAL\_URL指定的域名还需要配置域名服务器或本地hosts文件。

### 初次登录时需要为gitlab的root用户设置密码

### 如果因为防火墙拦截导致网站打不开，需要停止防火墙，停止防火墙服务：service firewalld stop

### 



### 初始密码随机生成，如练习是随机生成的密码为：ey+QNx41nqymlyA1PjXJcQtYUDz3nKVv40i/6O0CWB4=查看密码路径：/etc/gitlab/initial\_root\_password，但是在命令gitlab-ctl reconfigure执行24小时后自动删除该文件需要存储该密码或者登录后立即修改密码

# GitBash中多屏显示控制方式

## 空格向下翻页

## b向上翻页

## q退出

# vim模式下：按ESC然后:set nu查看行号

# 切换账号需要删除Windows凭据

# 本地库B

# 远程库A(代码托管中心)

# 本地库A

# 远程库B

# 本地库C