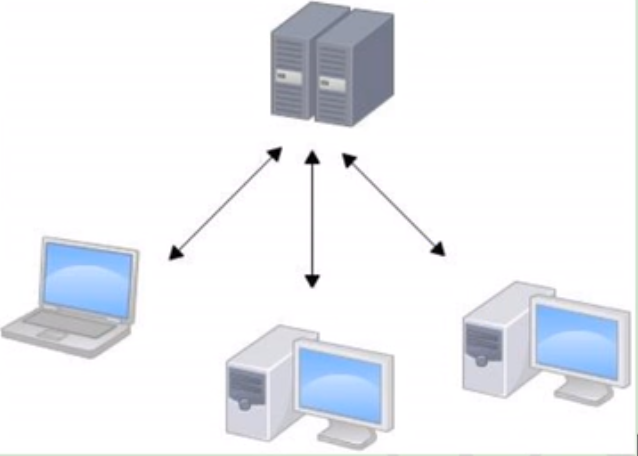
# Git笔记

# 1 版本控制介绍

## 1.1 集中式

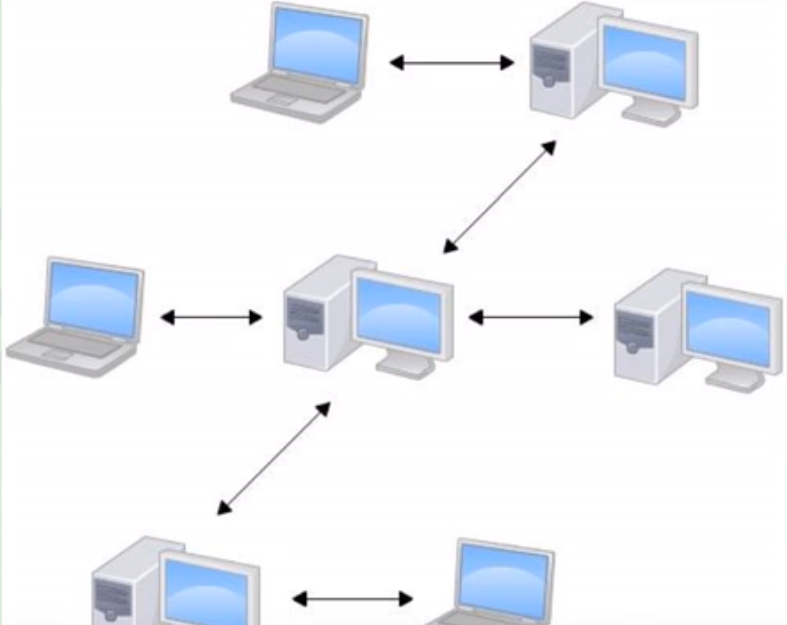
**案例工具**：CVS、**SVN**、VSS......



缺点：一旦服务器宕机，本地只有本地代码，历史版本就没有了，俗称单点故障。

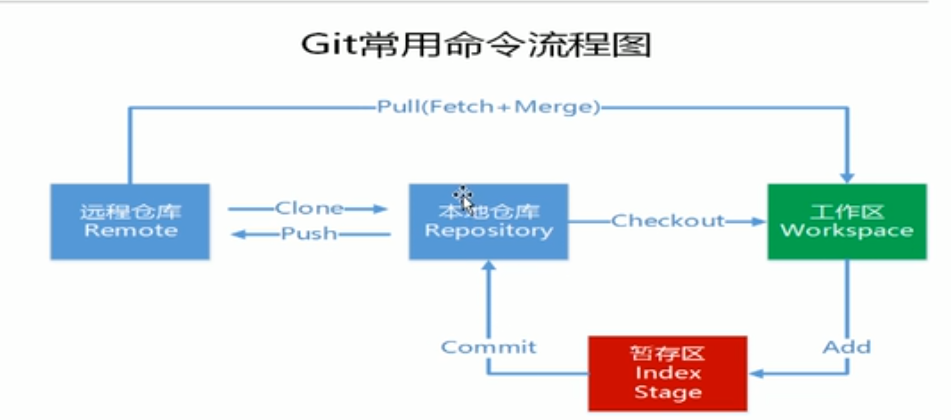
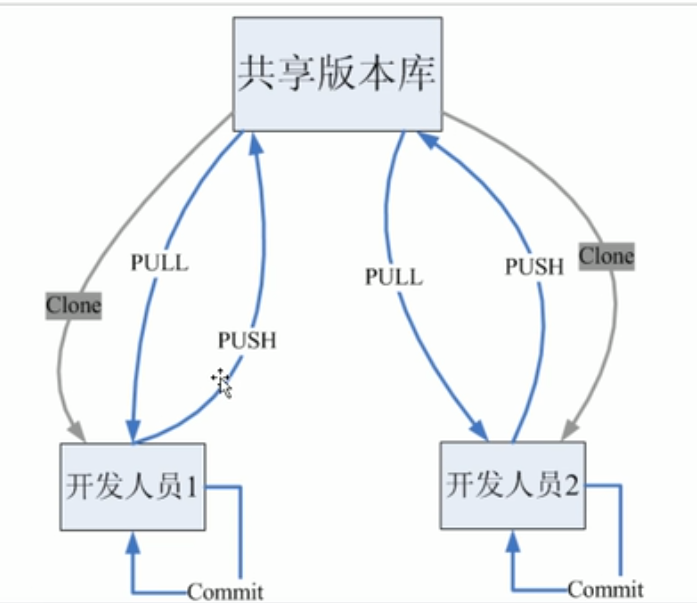
## 1.2 分布式

**案例工具**：**Git**、Mercurial、Bazaar、Darcs.......



每个人除了有本地代码，还有历史信息等，张三宕机了，可以拿李四的数据恢复。但实际上还是有一个远程库，避免单点故障。

Git可以实现本地闭环，远程的来源本地。



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linux命令全面兼容

# 2 Git 历史

Linux内核开源项目有着数量众多的参与者，然后绝大多数的linux内核维护工作都花在了提交不定和保存归档的繁琐事务上（1991-2002年间），然后linus手动合并代码。到2002年，整个项目组开始启用分布式版本控制系统BitKeeper来管理和维护代码，出于人道主义，授权linux社区免费试用版本控制，但要求不能进行破解。

2005年，因为linux社区不能像商业公司那样对参与开发者有强有力的约束；开发Samba的Andrew师徒破解BitKeeper的协议，被发现，BitMover要收回免费使用权。然后linus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。

# 3 Git安装

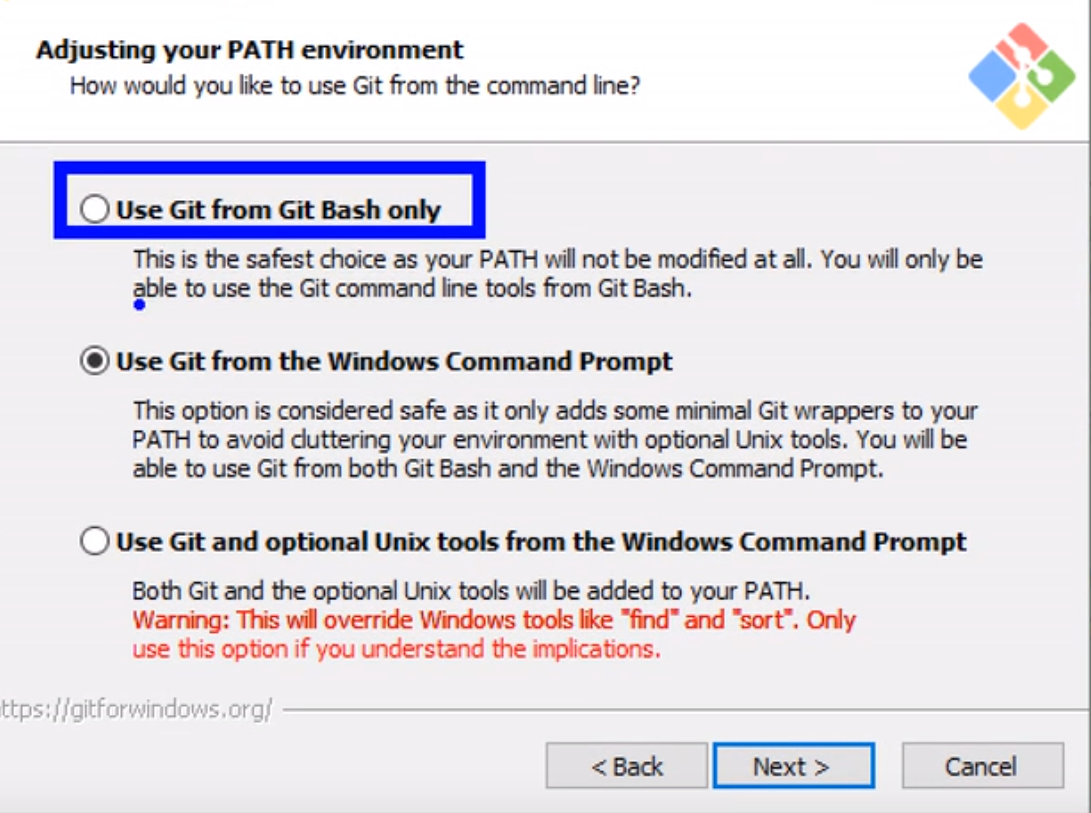
## 3.1 下载

版本：Git-2.22.0-64-bit.exe

地址：H:\软件安装包\Windows\编程

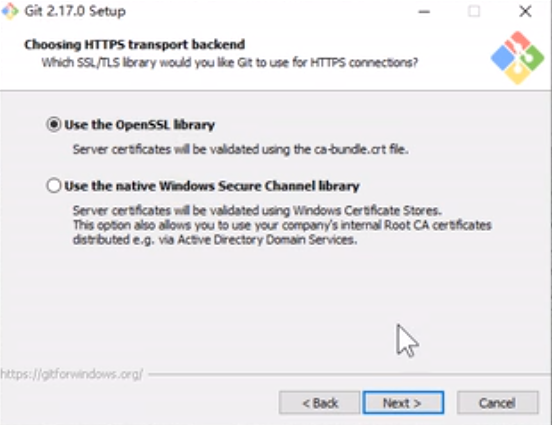
## 3.2 安装

安装地址：E:\development\Git\Git2.22\Git

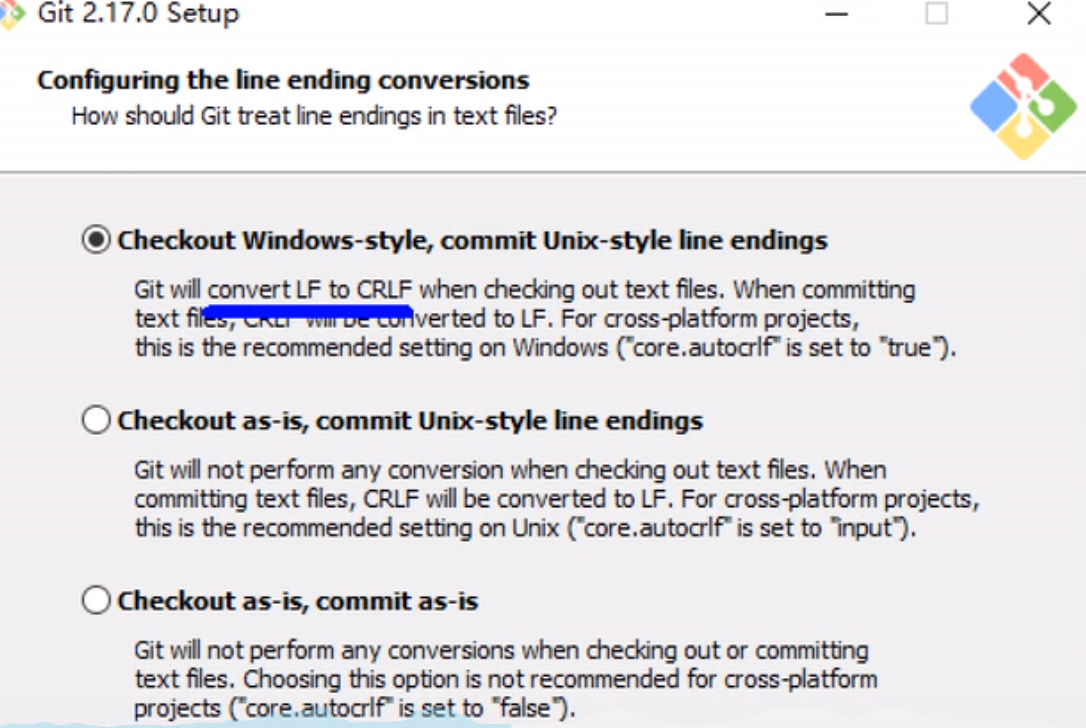
选第一个就好，第二个和第三个不要用。修改path变量



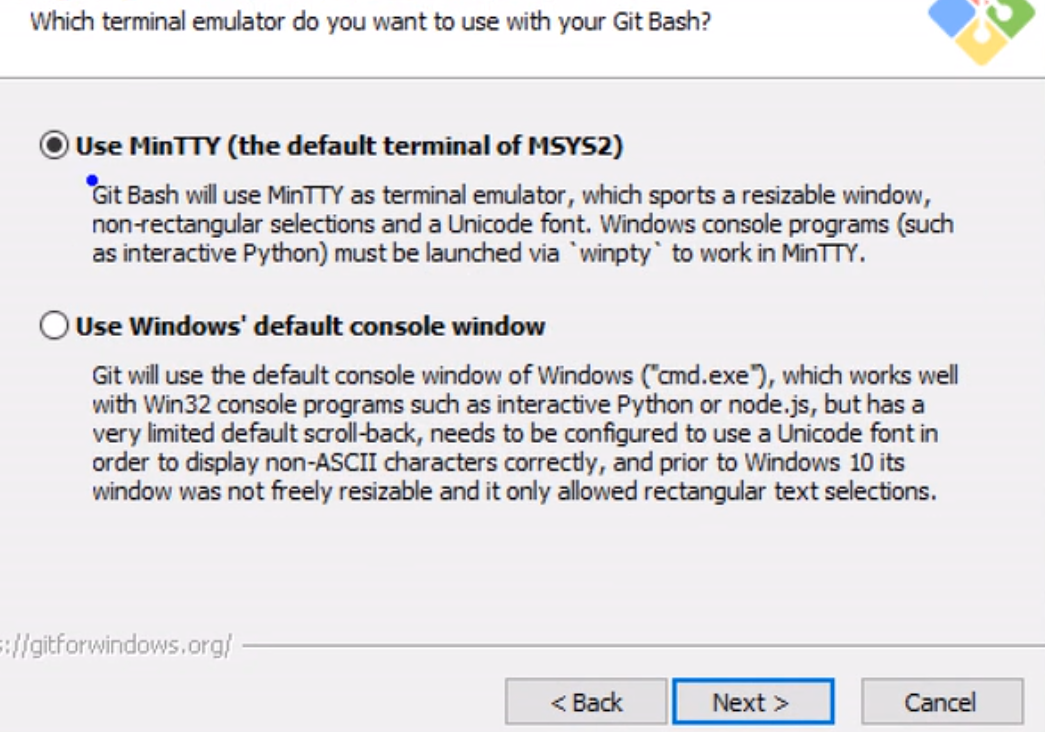
本地库和远程库的连接方式：

使用第一个

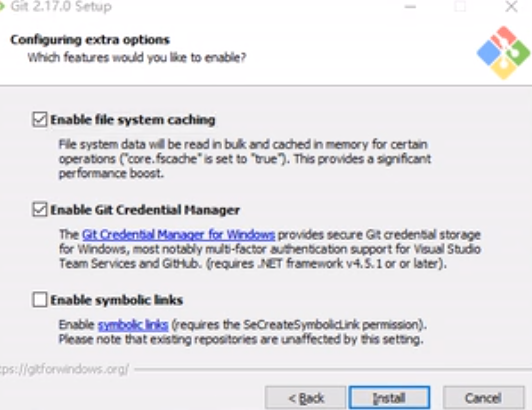
因为linux和Windows的换行不一致，选择一个转换

选第一个

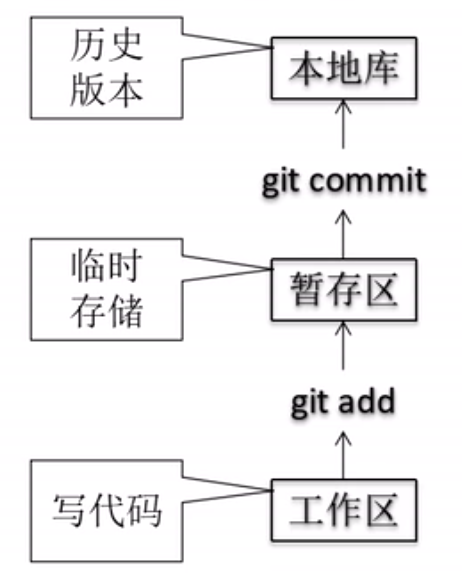
linux的命令行有linux的，不要混着用，命令行终端

，选第一个

启用缓存、授权管理器

选择默认，然后install；

## 3.3 Git结构（本地的）



# 4 Git和代码托管中心

代码托管中心的目的：维护远程库

## 4.1 局域网下

安装GitLab服务器

## 4.2 外网环境下

GitHub

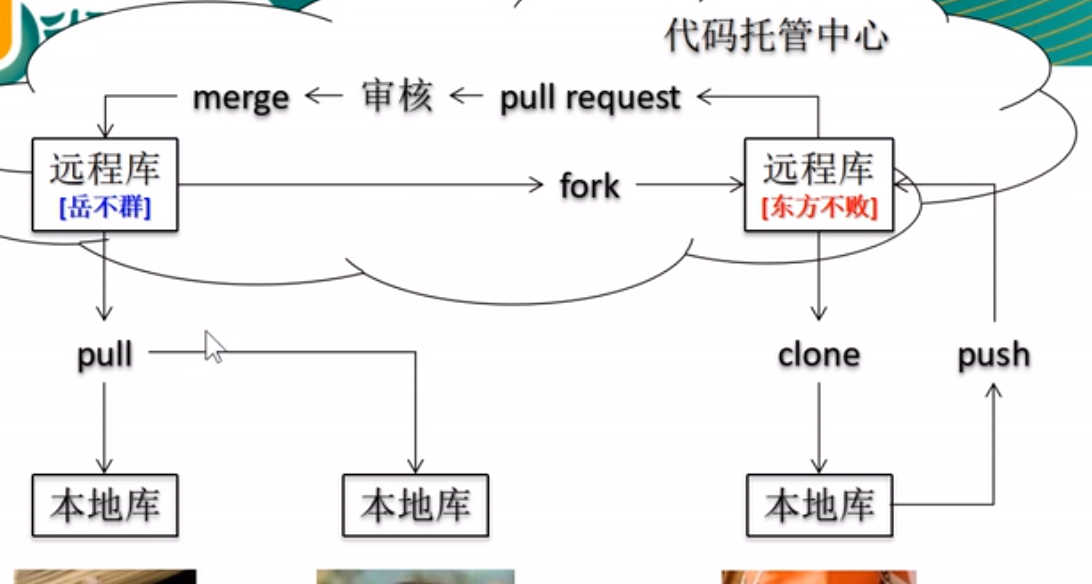
码云

## 4.3 本地库和远程库

* 内部团队协作



* 跨团队协作（比如两个公司，linux）



# 5 Git命令行操作

## 5.1 本地库操作

### 5.1.1 本地库初始化

* 初始化：就是建出来
* 命令：git add

总目录：**/f/project/LocaGitlRepository**；初始化一个test

mkdir test

cd test

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，
* 设置签名
* 形式：

用户名：tom

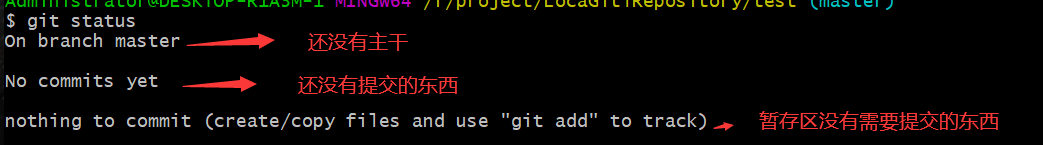
Email地址：[goodmo@www.com](mailto:goodmo@www.com) 可以不和用户名一样，也可以不填

* 作用：区分不同开发人员身份，就是每一个版本是谁提交修的等等
* 辨析：登录Git的用户名密码和这里的签名（用户名）没有一点关系。
* 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 [goodmo\_pro@11.com](mailto:goodmo_pro@11.com)
* 信息保存的位置：本地库下的./git/config文件里
* 系统用户级别：登录系统的那个用户
* git config **--global** user.name tom\_glb
* git config **--global** user.email [goodmo\_pro@11.com](mailto:goodmo_pro@11.com)
* 信息保存的位置：家目录，cd ~下的.gitconfig文件下
* 优先级：两个都有，就近原则（项目级别优先于系统用户）；如果都没有则没有访问权限

#### 5.1.2 基本操作

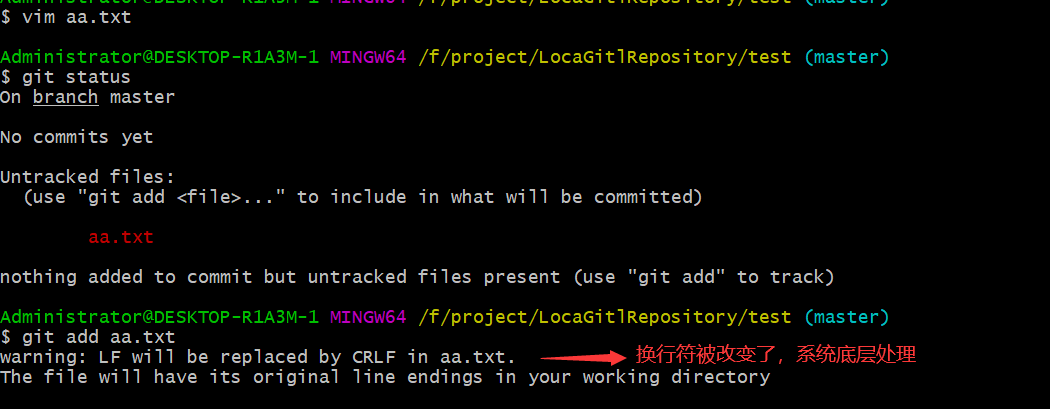
##### 1）git status

首先进到一个本地仓库下，敲下这个命令



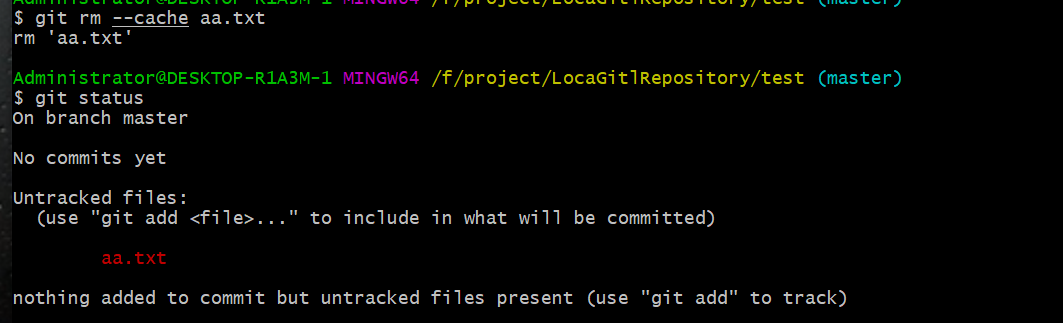
##### 2）git add [file name]

新建一个文件，然后放到缓存区（交给Git管理）**git add aa.txt**



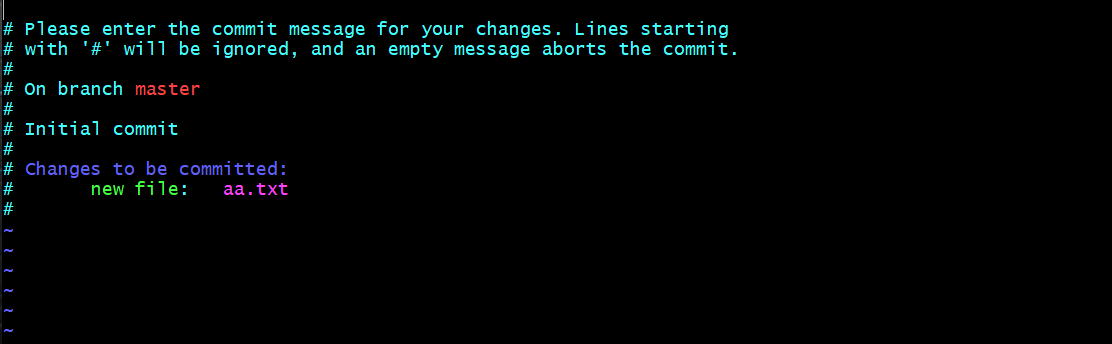
##### 3）git rm –cache [file nae]

从缓存区撤回：**git rm –cached aa.txt**



##### 4）git commit [file name]

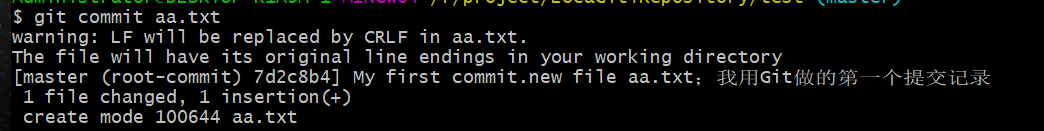
提交到本地库：git commit aa.txt 进入vim编辑器



给提交的文件加个注释：其实是在vim编辑器里面

:set nu

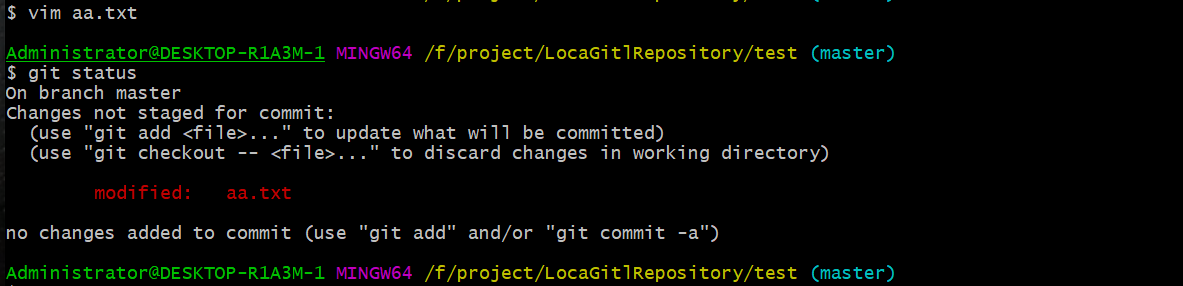
然后写入自己的提交记录



再次查看：



修改这个文件：



##### 5）git commit –m “BUG修复” [file name]

放到暂存区后提交或者可以直接提交：got commit –m “BUg修复” aa.txt

带上-m “” 就不用了进入vim编辑器了，直接添加记录

##### 6）git log



HEAD就是一个指针，指向当前的版本

最全的log；多屏显示时，空格向下翻页，b向上翻页，q退出

##### 7）git log –pretty=oneline

每个提交只显示一行

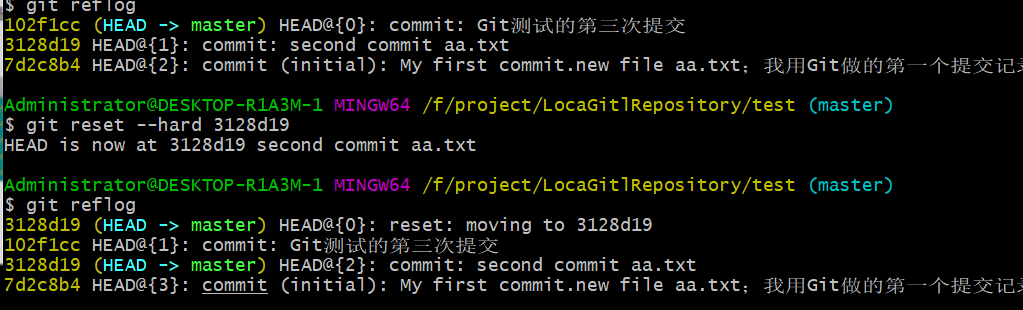
##### 8）git log –-oneline

hash值只显示一部分

##### 9）git reflog

多了指针的数目，有利于移动指针

##### 10）前进后退版本

* 基于索引值（最好的方式，前后都可以）
* 
* git reset --hard 3128d19 就回到指定的版本了
* 
* 使用^符号（异或符号，只能往后推，不能往前）
* git reset --hard HEAD^ 退一个版本
* git reset --hard HEAD^^ 退两个两版本，以此类推
* 使用~符号(只能后退)
* git reset --hard HEAD~n 后退n个版本

##### 11）hard、soft、mixed参数对比

* soft：仅仅在本地库移动HEAD指针
* mixed：在本地库移动HEAD指针，同时重置缓存区
* hard：在本地库移动HEAD指针，同时重置工具去和缓存区

##### 12）提交到本地库永久删除文件后找回文件

* 前提：删除前，提交到了本地库

rm [file name]

git add [file nname]

其实就是从本地库换到另一个版本

##### 13）添加到缓存区删除后找回文件

* 前提：删除前，提交到了本地库

git reset --hard HEAD

前提还是已经提交过的，第二次放到缓存区，是找回文件，不是新编辑的内容

##### 14）比较文件

git diff [file name] 是那本地文件和缓存比较

git diff HEAD [file name] 和指定的版本比较，不写就是上一个版本

##### 15）分支操作

查看所有分支：git branch –v

创建一个分支：git branch [分支名]

切换分支：git checkout [分支名]

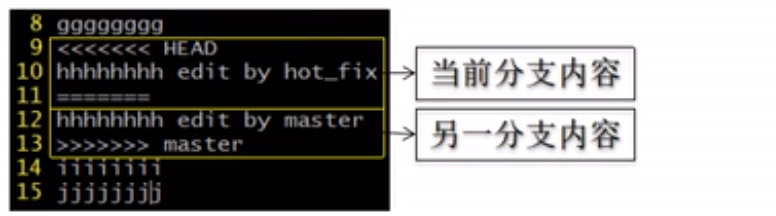
合并分支：

第一步：切换到接受修改的分支上

第二步：执行git merge [变动的分支（它的内容将被合并出去）]

解决冲突：

冲突的表现：

人去修改，然后保存

把合并后的文件添加到缓存区：git add [文件名]

再提交本地库：这里不能带文件名 git commit –m “log” 不要带文件名

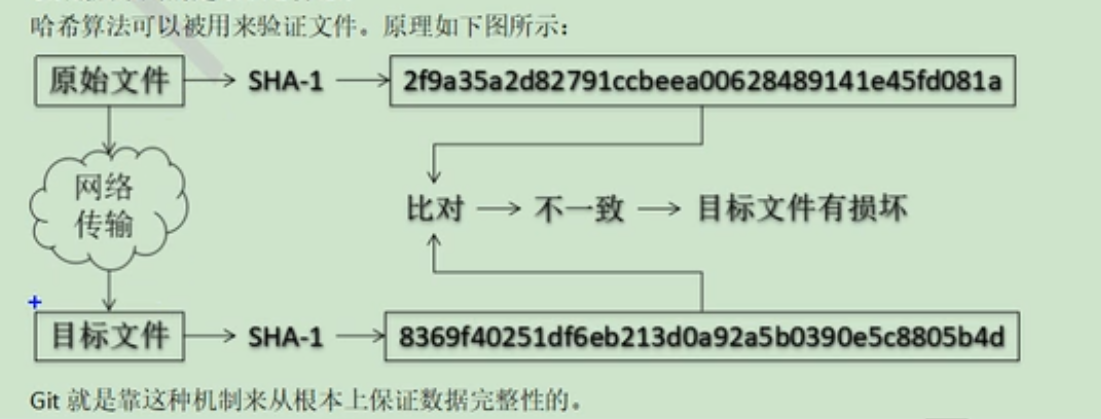
#### 5.1.3 Git基本原理

##### 1）hash

Hash是一个加密算法，有很多种hash加密算法，共同点如下：

1. 结果长度固定
2. hash算法确定，入参确定，则出参不变
3. hash算法确定，入参有变化，出参一定有变化，通常变化很大
4. hash算法不可逆

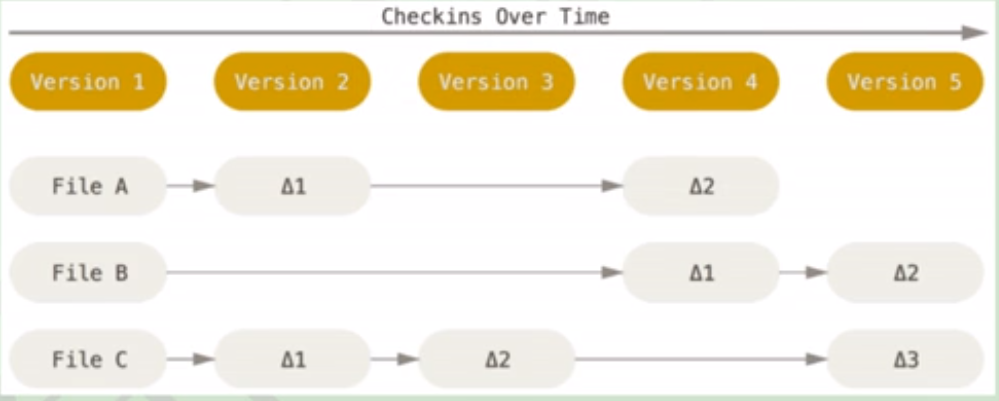
Git底层采用的SHA-1哈希算法。



##### 2）git保存版本的机制

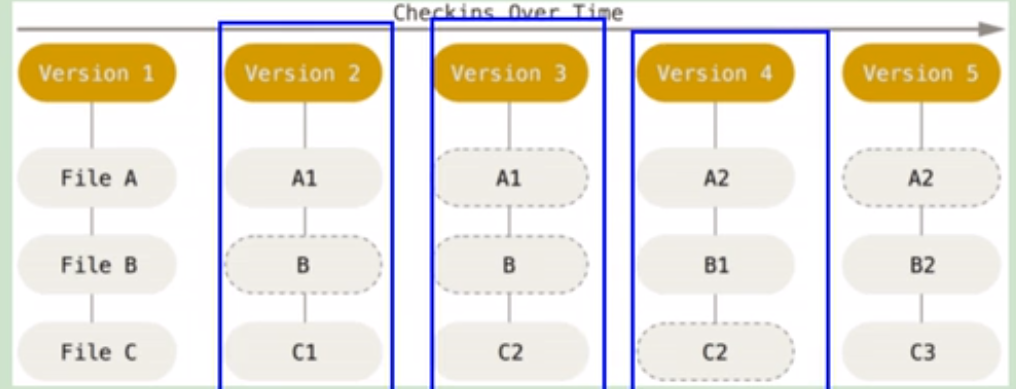
* 集中式版本控制工具的文件管理机制

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做是一组基本文件和每个文件随时间逐步积累的差异。

想要某个版本的信息，就把原始问价和之间的改动拼在一起，就成了，缩成增量式（节约存储空间）

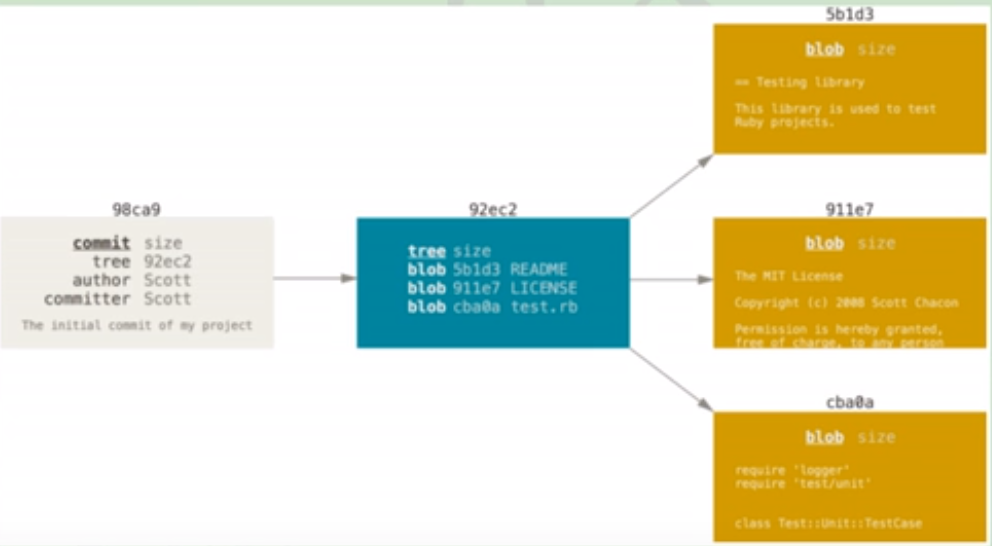
* Git的文件管理机制

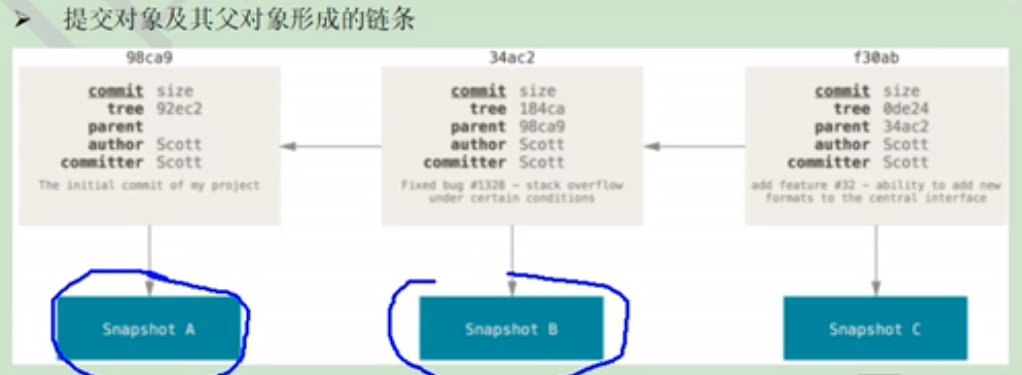
Git把数据看做是小型文件系统的一组快照。每次提交更新时都会对当前全部文件制作一个快照并保存这个快照的索引。为了高效，如果文件没有修改，Git不再重新存储该文件，只是保留一个链接指向之前存储的文件。俗称快照流。



##### 3）git 文件管理机制细节

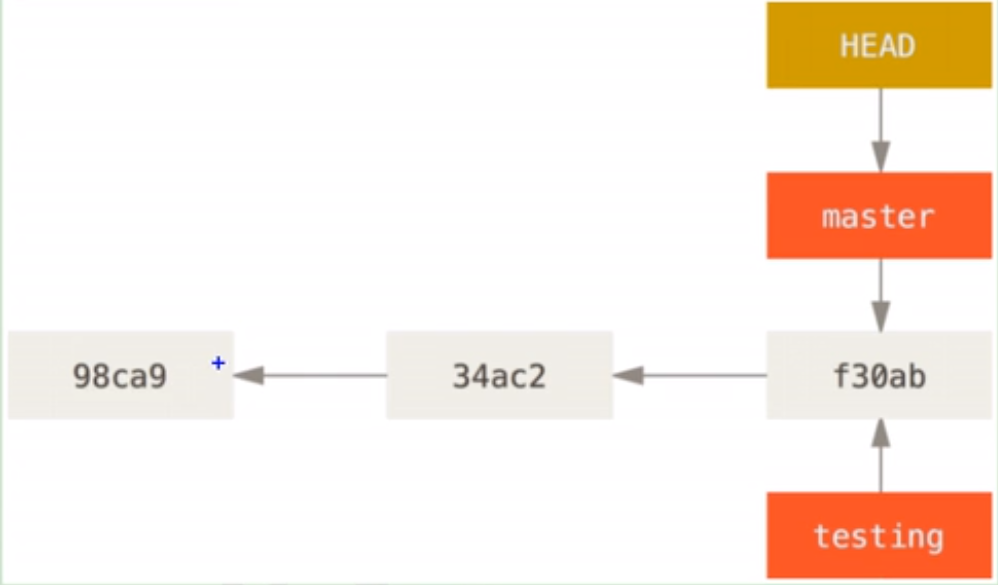
* Git的“提交对象”





##### 4）git 分支管理机制

* 分支的创建



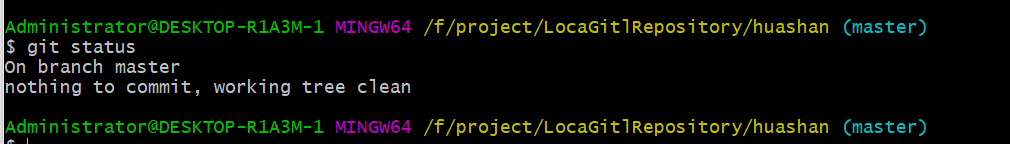
* 分支的切换



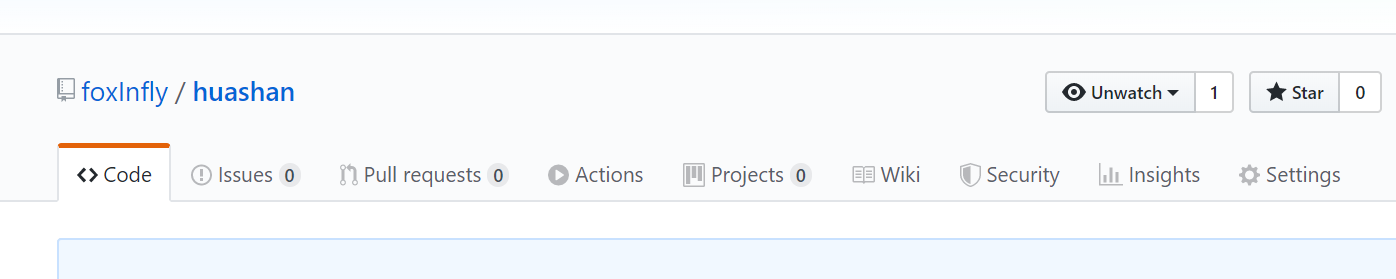
### 5.2 远程库操作

#### 5.2.1 本地库和远程库的交互

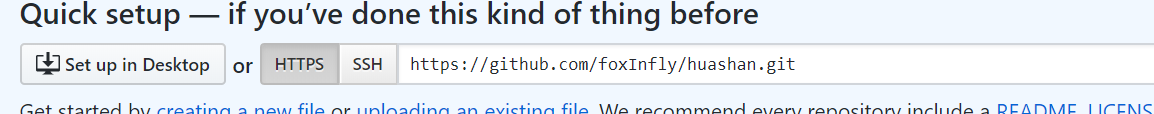
##### 1）初始化本地库



##### 2）初始化远程库



##### 3）本地库push远程库

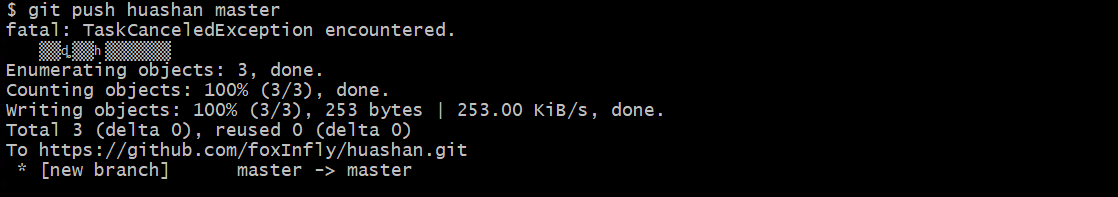
远程库地址：

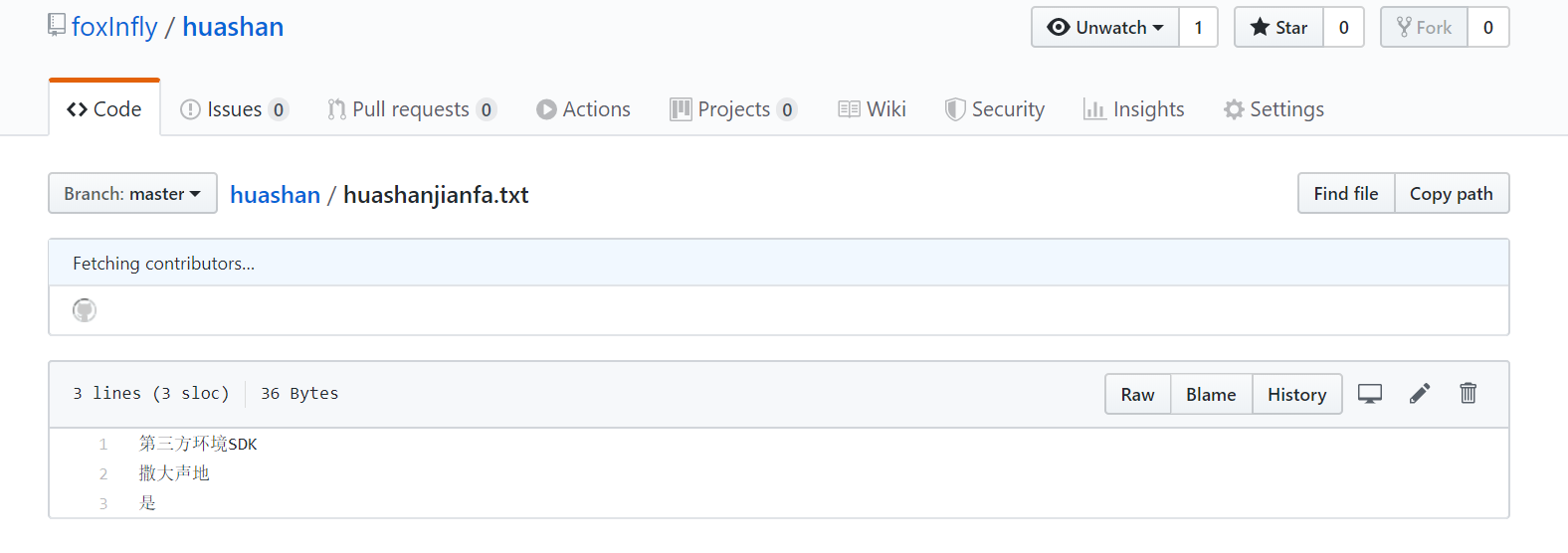
https://github.com/foxInfly/huashan.git

起别名：

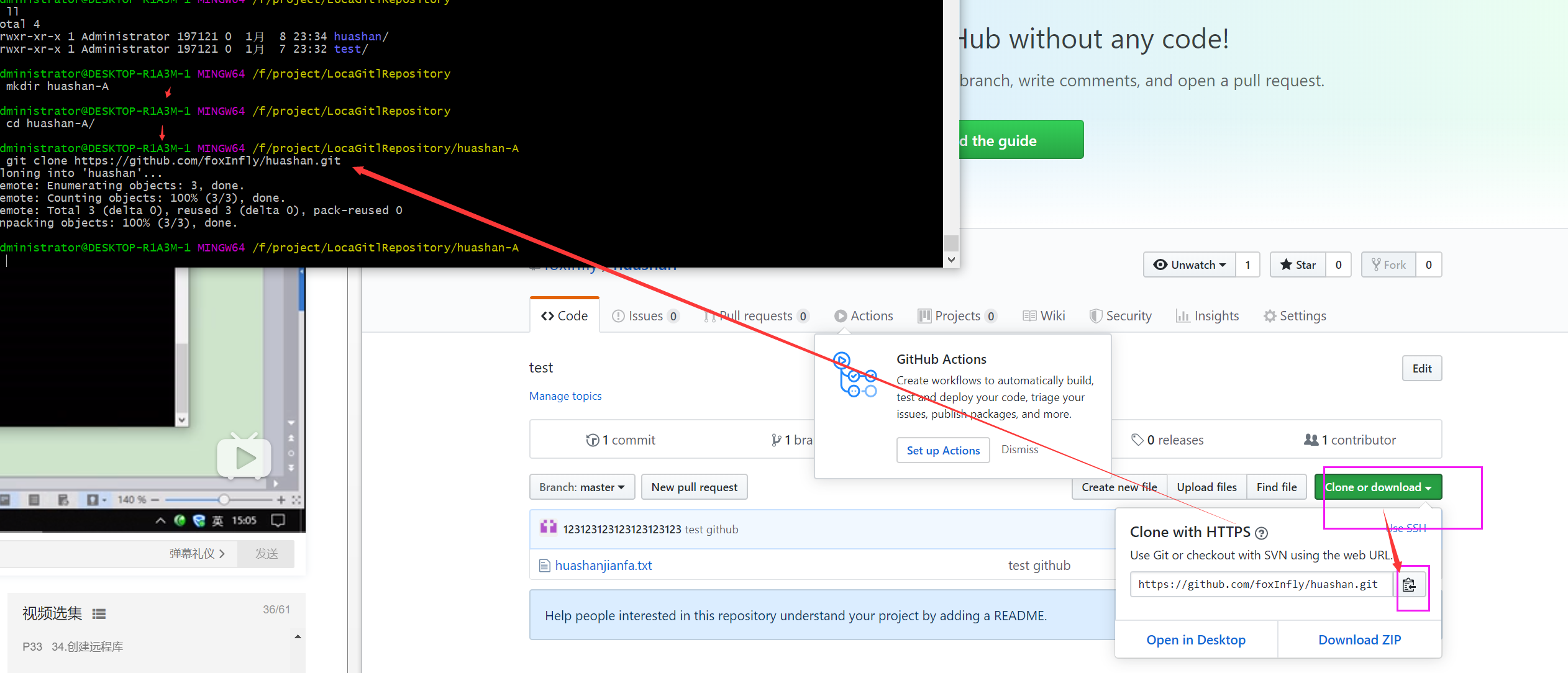
##### 4）推送push：

git push huashan master 推送某个分支到指定的远程库

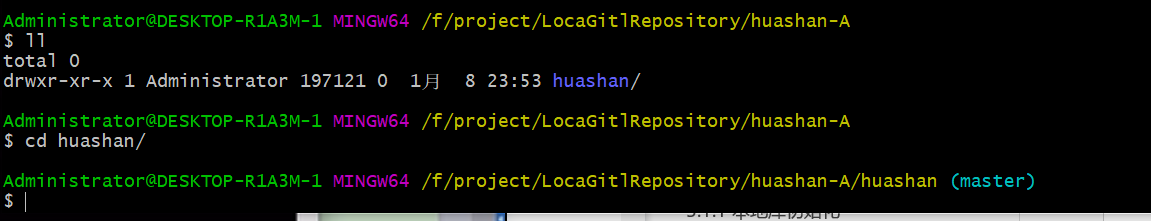




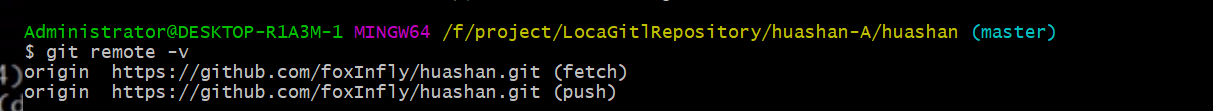
##### 5）远程库clone到本地库



查看



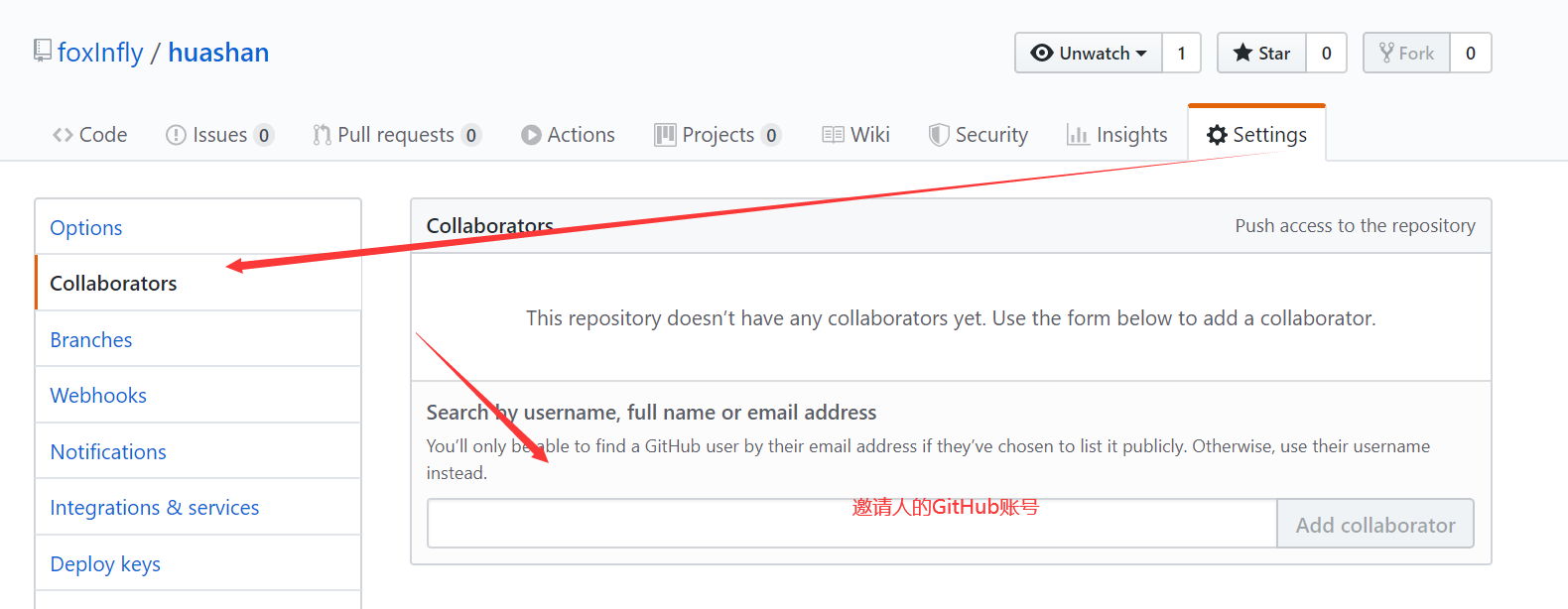
1. 完整的把远程库下载到本地
2. 创建远程地址别名
3. 初始化本地库



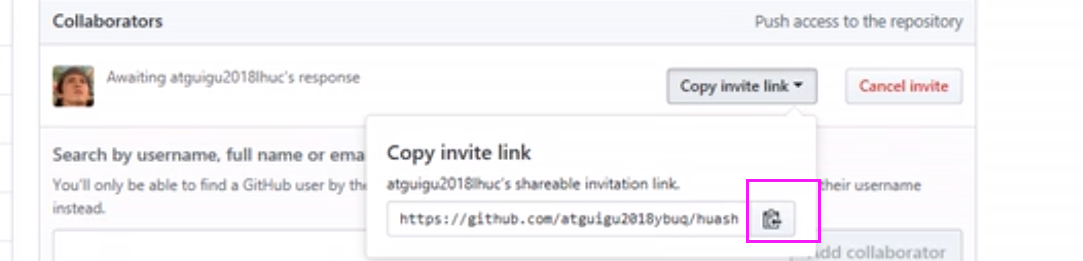
##### 7）邀请B加入成为团队成员

如果没有权限，会报403

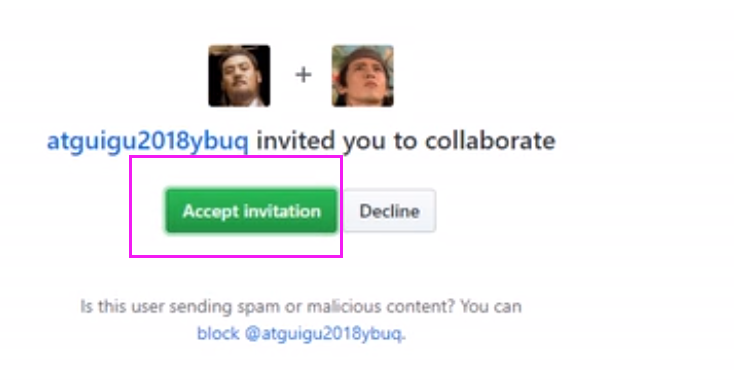
床架邀请连接：



复制链接给邀请人：



邀请人访问地址后，点击接受：



后面就可以提交操作了。

##### 8）远程库修改的拉取pull

pull=fetch(抓取)+merge（合并）

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]

##### 9）冲突解决

准备push时，已经有人push了，只能先pull，解决冲突后再push

#### 5.2.2 跨团队操作

需要两个远程库

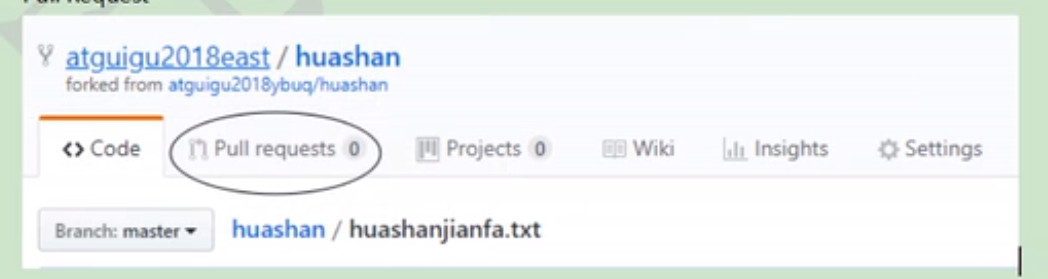
fork

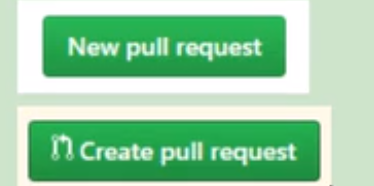


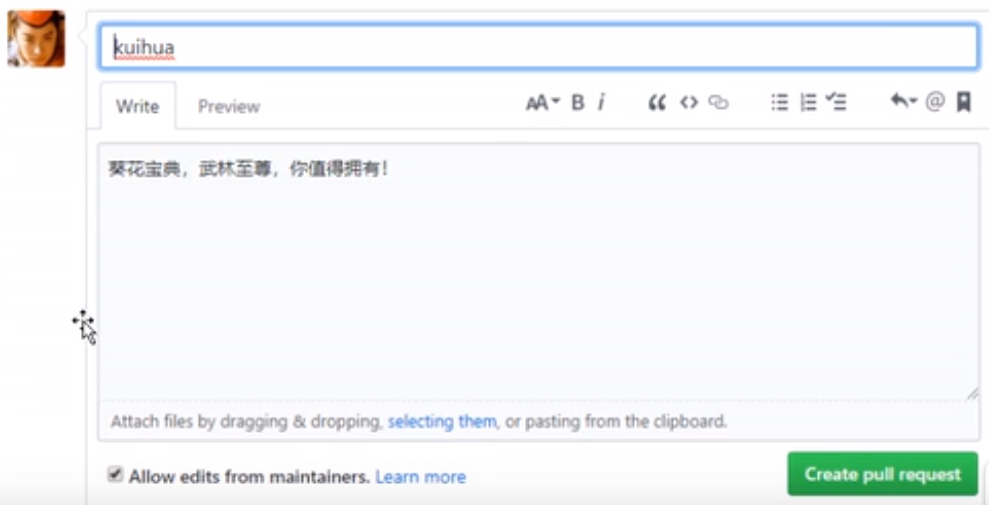
修改

发给第一个远程库，让他审核

pull request











对话；

合并

再拉取到本地，

协作完成。

#### 5.2.3 SSH

### 5.4

# 6 Gitlab服务器搭建

## 6.1 官网

首页：https://about.gitlab.com/

安装说明：https://about.gitlab.com/installation

linux版的Git；C语言开发的；是源代码

https://packges.gitlab.com/gitlab/gitlab-ce/packages/el/7

## 6.2 安装命令摘录

Centos7以上

1. 安装一些依赖软件包，SSH一般系统是默认安装好的，不过也不排除一些最小安装的系统没有sshd服务。

sudo yum install -y curl policycoreutils-python openssh-server cronie

sudo systemctl enable sshd

sudo systemctl start sshd

如果提示lokkit找不到命令先安装

yum install lokkit

1. 开放端口82
2. 安装邮件服务器，当gitlab想要通过邮件通知，也可以另外配置其它的邮件服务器

sudo yum install postfix

sudo systemctl enable postfix

sudo systemctl start postfix

1. 从官网获取一件安装脚本，当然自己手动安装也是可以的；使用官网脚本会简单一些。执行这一步会如果使用CentOS系统，会添加gitlab的yum源

//输出到文件里是为了看下下载的脚本内容

curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.rpm.sh > rpm.sh

chmod +x rpm.sh

./rpm.sh

路径：/pl/gitlab

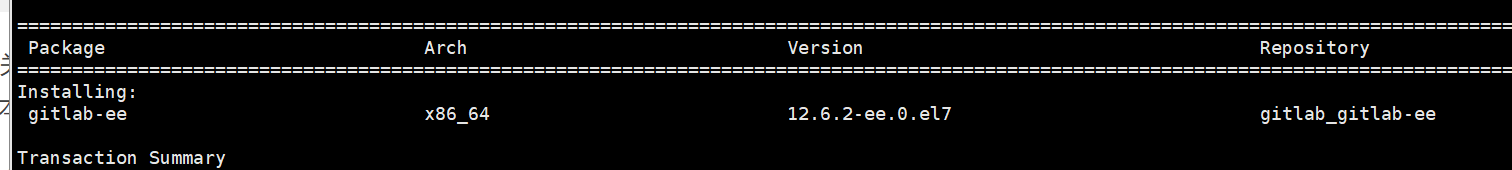
5）安装gitlib

//使用yum安装gitlab

yum install -y gitlab-ee

//可以看下gitlab-ee包的内容，看到gitlab安装在/opt/gitlab目录下

rpm -ql gitlab-ee | less



6）上面已经安装好了gitlab，不过可以稍作一些配置，配置gitlab监听的地址与端口，gitlab的配置文件在/etc/gitlab/目录下，主要配置文件为gitlab.rb

我修改了下gitlab.rb文件中的nginx监听地址,

external\_url 'http://gitlab.ai-he.me'

nginx['listen\_addresses'] = ['0.0.0.0', '[::]']

# 系统端口冲突，我把端口改为了82

nginx['listen\_port'] = 82

7）先重启，运行gitlib命令，

//重新配置gitlab

sudo gitlab-ctl reconfigure

//重启gitlab

gitlab-ctl restart

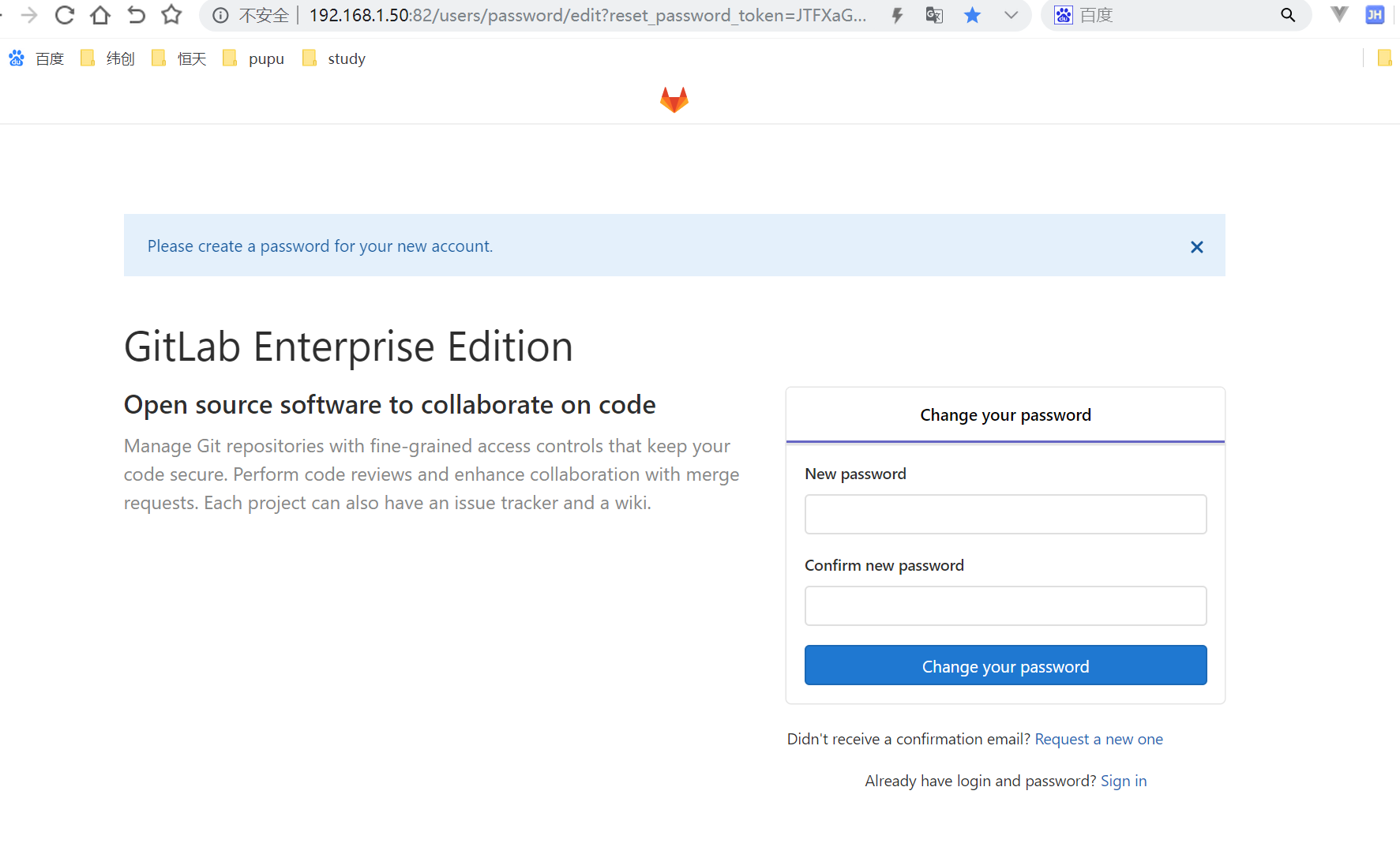
// 查看gitlab-ctl命令的帮助信息

gitlab-ctl –help

start stop

8）打开浏览器查看效果,第一次打开页面会让我们设置root用户的密码。记住自己设置的密码，再次刷新进入登录页面。

192.168.1.50:82计入如下页面



root/gitlab123

9）以管理员身份登录，默认的用户是root，密码是刚才设置的。

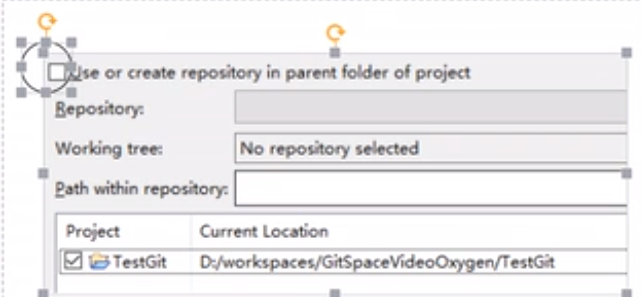
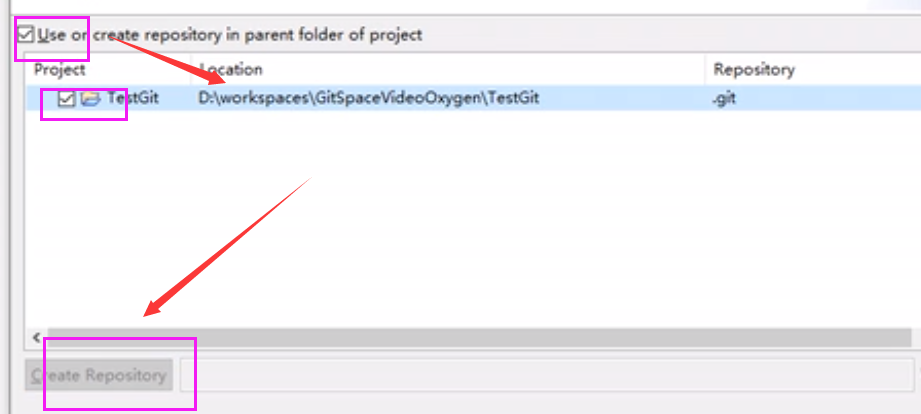
## 6.3 访问

# 7 IDEA集成Git

## 7.1 Eclipse集成Git

### 7.1.1 工程初始化城本地库

* 工程上右键→Team→Share Project→Git

* Create Repository→finish

### 7.1.2 Git图标

### 7.1.3 忽略特定文件

.classpath

.project

.settings目录下的文件

以上都是Eclipse为了管理工程而创建的文件，跟开发没有一点关系。最好不要在Git中追踪，忽略（防止每个人的IDE版本不一样，可能会冲突）

#############################################

# Eclipse.gitignore global

#############################################

.metadata

bin/

tmp/

\*.tmp

\*.bak

\*.swp

\*~.nib

local.properties

.settings/

.loadpath

.recommenders

# External tool builders

.externalToolBuilders/

# Locally stored "Eclipse launch configurations"

\*.launch

# PyDev specific (Python IDE for Eclipse)

\*.pydevproject

# CDT-specific (C/C++ Development Tooling)

.cproject

# CDT- autotools

.autotools

# Java annotation processor (APT)

.factorypath

# PDT-specific (PHP Development Tools)

.buildpath

# sbteclipse plugin

.target

# Tern plugin

.tern-project

# TeXlipse plugin

.texlipse

# STS (Spring Tool Suite)

.springBeans

# Code Recommenders

.recommenders/

# Annotation Processing

.apt\_generated/

.apt\_generated\_test/

# Scala IDE specific (Scala & Java development for Eclipse)

.cache-main

.scala\_dependencies

.worksheet

在~/.gitconfig文件中引入上述文件

[core]

excludesfile=C:/user/xxx/Java.gitignore

### 7.1.4 工程初始化城本地库

## 7.2 IDEA集成Git

### 7.2.1 IDEA配置Git

### 7.2.2 加入项目到本地项目

### 7.2.3 忽略特定文件

1、Git，在项目跟目录下增加.gitignore文件，内容如下：

target/

!.mvn/wrapper/maven-wrapper.jar

### STS ###

.apt\_generated

.classpath

.factorypath

.project

.settings

.springBeans

### IntelliJ IDEA ###

.idea

\*.iws

\*.iml

\*.ipr

.mvn

mvnw\*

### NetBeans ###

nbproject/private/

build/

nbbuild/

dist/

nbdist/

.nb-gradle/

=================================================================================================

===================#############################################

# JetBrains.gitignore global

#############################################

# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio and WebStorm

# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839

# User-specific stuff

.idea/\*\*/workspace.xml

.idea/\*\*/tasks.xml

.idea/\*\*/usage.statistics.xml

.idea/\*\*/dictionaries

.idea/\*\*/shelf

# Generated files

.idea/\*\*/contentModel.xml

# Sensitive or high-churn files

.idea/\*\*/dataSources/

.idea/\*\*/dataSources.ids

.idea/\*\*/dataSources.local.xml

.idea/\*\*/sqlDataSources.xml

.idea/\*\*/dynamic.xml

.idea/\*\*/uiDesigner.xml

.idea/\*\*/dbnavigator.xml

# Gradle

.idea/\*\*/gradle.xml

.idea/\*\*/libraries

# Gradle and Maven with auto-import

# When using Gradle or Maven with auto-import, you should exclude module files,

# since they will be recreated, and may cause churn. Uncomment if using

# auto-import.

# .idea/modules.xml

# .idea/\*.iml

# .idea/modules

# \*.iml

# \*.ipr

# CMake

cmake-build-\*/

# Mongo Explorer plugin

.idea/\*\*/mongoSettings.xml

# File-based project format

\*.iws

# IntelliJ

out/

# mpeltonen/sbt-idea plugin

.idea\_modules/

# JIRA plugin

atlassian-ide-plugin.xml

# Cursive Clojure plugin

.idea/replstate.xml

# Crashlytics plugin (for Android Studio and IntelliJ)

com\_crashlytics\_export\_strings.xml

crashlytics.properties

crashlytics-build.properties

fabric.properties

# Editor-based Rest Client

.idea/httpRequests

# Android studio 3.1+ serialized cache file

.idea/caches/build\_file\_checksums.ser

忽略配置Maven规则

#############################################

# Maven.gitignore global

#############################################

target/

pom.xml.tag

pom.xml.releaseBackup

pom.xml.versionsBackup

pom.xml.next

release.properties

dependency-reduced-pom.xml

buildNumber.properties

.mvn/timing.properties

# https://github.com/takari/maven-wrapper#usage-without-binary-jar

.mvn/wrapper/maven-wrapper.jar

在Java项目中添加.gitignore文件

#################################################

# Java.gitignore

#################################################

# Compiled class file

\*.class

# Log file

\*.log

# BlueJ files

\*.ctxt

# Mobile Tools for Java (J2ME)

.mtj.tmp/

# Package Files #

\*.jar

\*.war

\*.nar

\*.ear

\*.zip

\*.tar.gz

\*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error\_hotspot.xml

hs\_err\_pid\*

### 7.2.4 和私有库交互

## xx001 常识

### 1 BitKeeper

BitMover是BitKeeper的开发厂商，创始人和CEO是Larry McVoy。Larry期望BitKeeper 能帮助Linus 免于陷入不断加重的Linux内核管理工作中－－事实上，自从Linus 3年前开始使用BitKeeper之后，Linux的开发步伐加快了两倍。但是收费