# 学习目标

了解Git历史

能够安装Git客户端

能够使用Git提交和删除文件

能够搭建Git服务器

能够添加创建远程项目仓库

能够从仓库拉取代码

能够向仓库推送代码

能够创建分支

能够合并分支

能够解决冲突

# Git历史

同生活中的许多伟大事件一样，Git 诞生于一个极富纷争大举创新的年代。**Linux 内核开源项目**有着为数众广的参与者。绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上（1991－2002年间）。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。

到 2005 年的时候，开发 **BitKeeper** 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了免费使用 BitKeeper 的权力。这就迫使 Linux 开源社区（特别是 Linux的缔造者 Linus Torvalds ）不得不吸取教训，只有开发一套属于自己的版本控制系统才不至于重蹈覆辙。他们对新的系统订了若干目标：

• 速度

• 简单的设计

• 对非线性开发模式的强力支持（允许上千个并行开发的分支）

• 完全分布式

• 有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量）



# Git与svn对比

## Svn

SVN是集中式版本控制系统，版本库是集中放在**中央服务器**的，而干活的时候，用的都是自己的电脑，所以首先要从中央服务器哪里得到最新的版本，然后干活，干完后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，如果在局域网还可以，带宽够大，速度够快，如果在互联网下，如果网速慢的话，就郁闷了。

下图就是标准的集中式版本控制工具管理方式：



集中管理方式在一定程度上看到其他开发人员在干什么，而管理员也可以很轻松掌握每个人的开发权限。

但是相较于其优点而言，集中式版本控制工具缺点很明显：

* 服务器单点故障
* 容错性差

## Git

Git是**分布式**版本控制系统，那么它就没有中央服务器的，每个人的电脑就是一个完整的版本库，这样，工作的时候就不需要联网了，因为版本都是在自己的电脑上。既然每个人的电脑都有一个完整的版本库，那多个人如何协作呢？比如说自己在电脑上改了文件A，其他人也在电脑上改了文件A，这时，你们两之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

下图就是分布式版本控制工具管理方式：

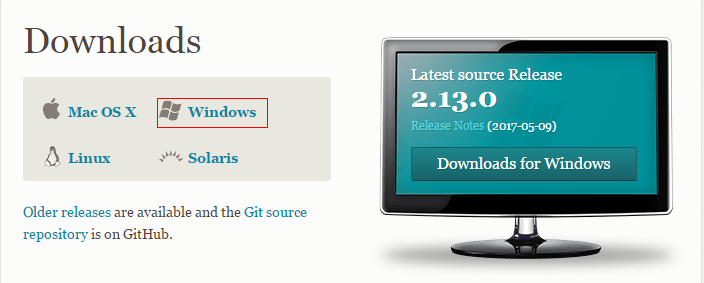


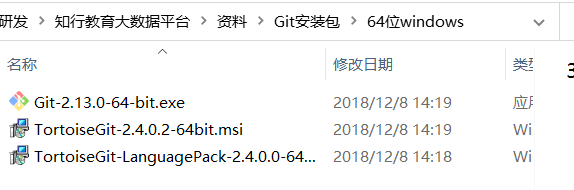
# Git客户端安装

最早Git是在Linux上开发的，很长一段时间内，Git也只能在Linux和Unix系统上跑。不过，慢慢地有人把它移植到了Windows上。现在，Git可以在Linux、Unix、Mac和Windows这几大平台上正常运行了。由于开发机大多数情况都是windows，所以我们只讲解windows下的git的安装及使用。

## 软件下载

下载地址：[https://git-scm.co](https://git-scm.com/download)[m/download](https://git-scm.com/download)



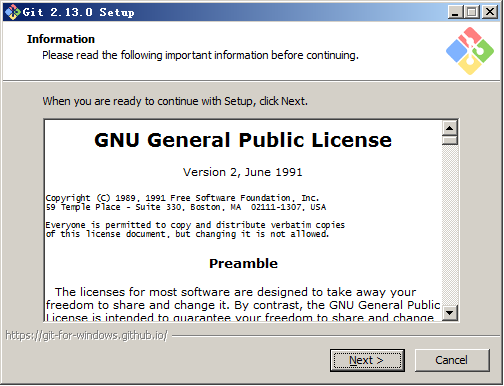


参考资料中安装包已经下载完毕，根据不同的操作系统选择对应的安装包。

## 软件安装

### 安装git for windows

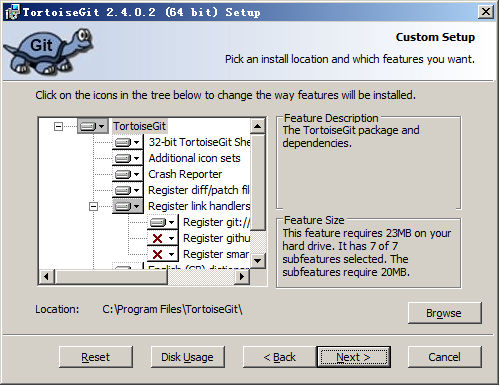




一路“下一步”使用默认选项即可。

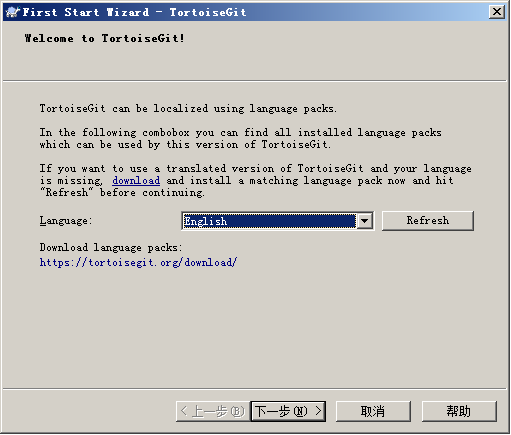
### 安装TortoiseGit





一路“下一步”使用默认选项即可。

默认选项下会启动配置画面：

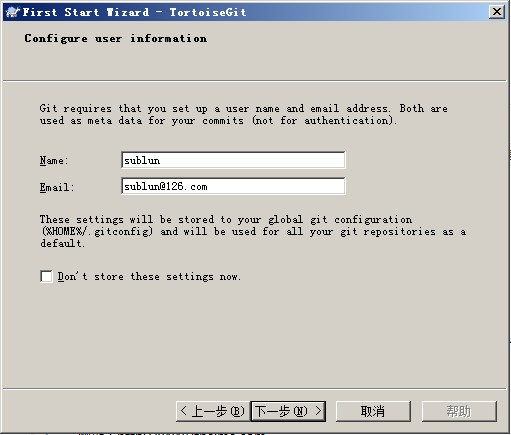


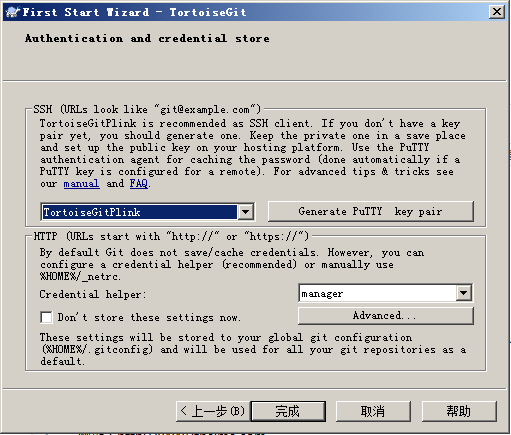
由于目前只有英文语言包，默认即可继续下一步。

配置git.exe，在4.2.1中已经安装过git-for-windows了所以在此找到git.exe所在的目录。



配置开发者姓名及邮箱，每次提交代码时都会把此信息包含到提交的信息中。





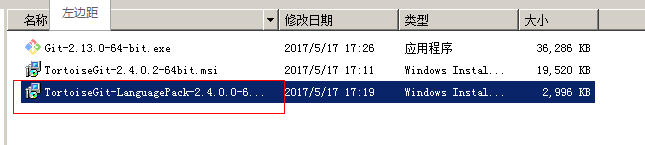
使用默认配置，点击“完成”按钮完成配置。

完整完毕后在系统右键菜单中会出现git的菜单项。



### 安装中文语言包

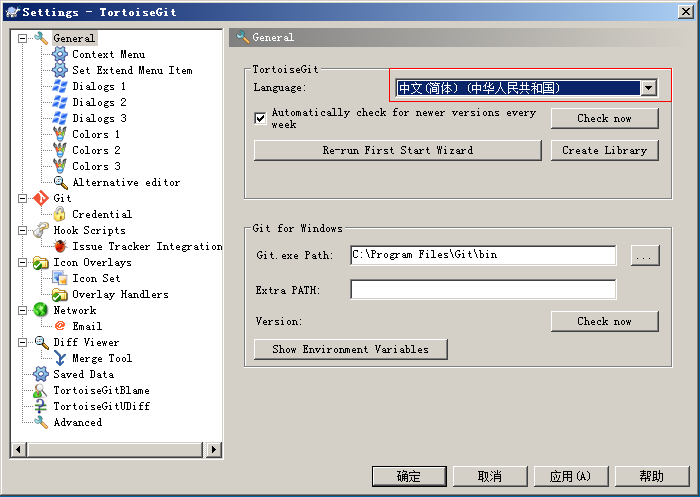
安装中文语言包并不是必选项。可以根据个人情况来选择安装。





直接“下一步”完整完毕。

语言包安装完毕后可以在TortoiseGit的设置中调整语言



# 使用git管理文件版本

## 创建版本库

什么是版本库呢？版本库又名仓库，英文名repository，你可以简单理解成一个目录（其实就是中央仓库），这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。由于git是分布式版本管理工具，所以git在不需要联网的情况下也具有完整的版本管理能力。

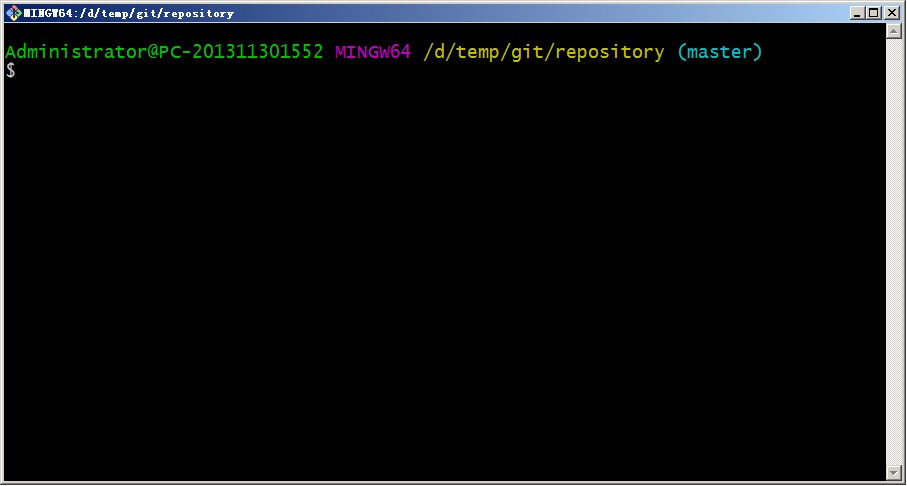
创建一个版本库非常简单，可以使用git bash也可以使用tortoiseGit。首先，选择一个合适的地方，创建一个空目录（D:\temp\git\repository）。

### 使用GitBash

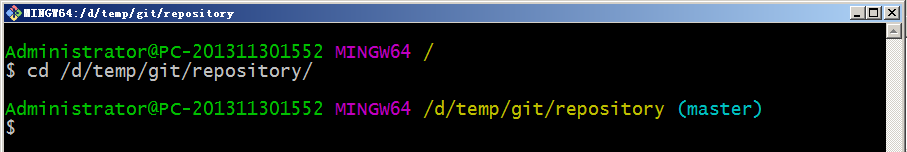
命令：

**$ git init**

在当前目录中点击右键中选择Git Bash来启动。



或者在开始菜单中启动。注意如果是从开始菜单启动的gitbash需要切换目录到仓库所在的目录。



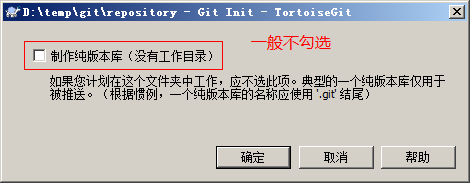
创建仓库执行命令：

**$ git init**

### 使用TortoiseGit

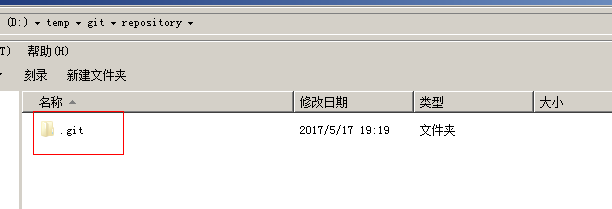
使用TortoiseGit时只需要在目录中点击右键菜单选择“在这里创建版本库”







版本库创建成功，会在此目录下创建一个.git的隐藏目录，如下所示：



在windows中如何显示隐藏目录隐藏目录请自行百度o(╯□╰)o

**概念：**

**版本库：“.git”目录就是版本库，将来文件都需要保存到版本库中。**

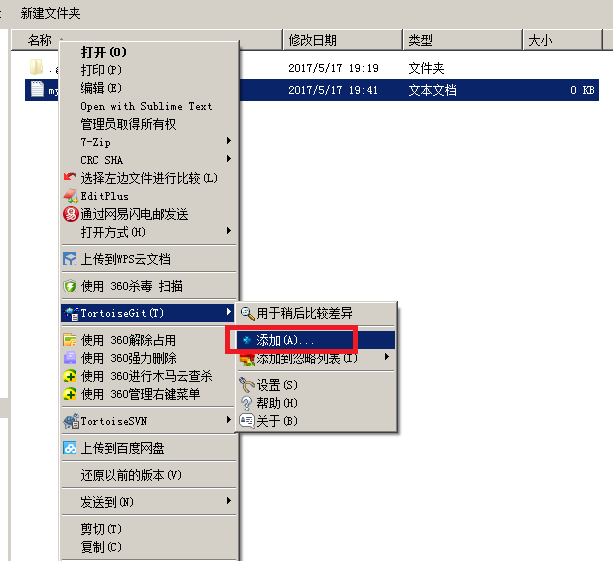
**工作目录：包含“.git”目录的目录，也就是.git目录的上一级目录就是工作目录。只有工作目录中的文件才能保存到版本库中。**

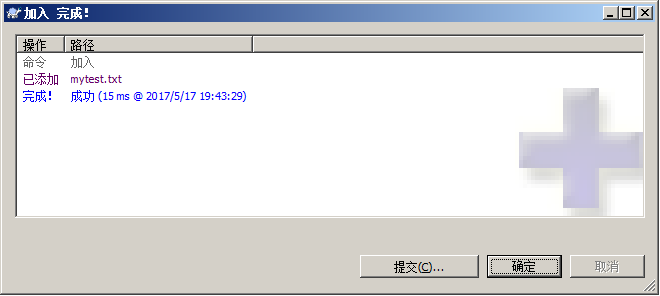
## 添加文件

### 添加文件过程

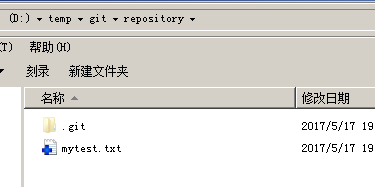
在D:\temp\git\repository目录下创建一个mytest.txt文件：



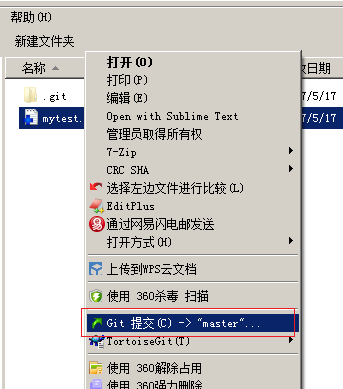


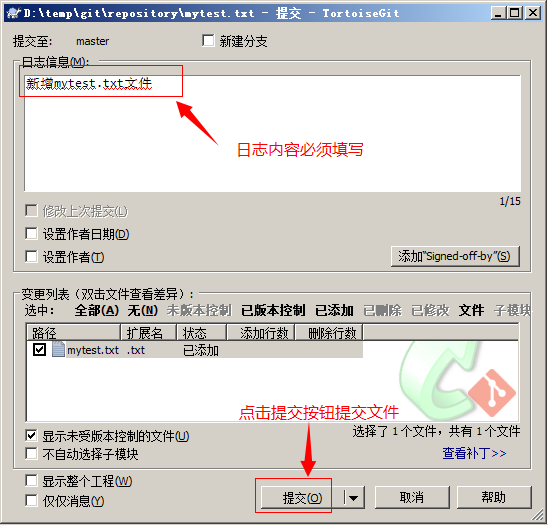


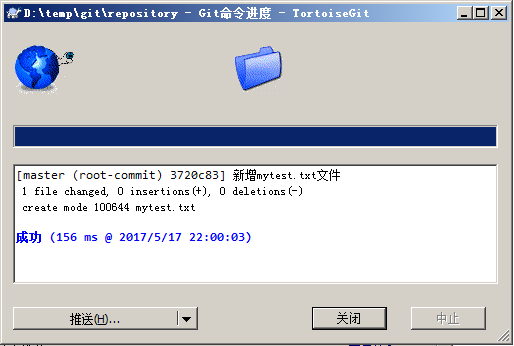
文本文件变为带“+”号的图标：



提交文件：在mytest.txt上再次点击右键选择**“提交”**，此时将文件保存至版本库中。







### 工作区和暂存区

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。

什么是工作区（Working Directory）？

工作区就是你在电脑里能看到的目录，比如我的reporstory文件夹就是一个工作区。

有的同学可能会说repository不是版本库吗怎么是工作区了？其实repository目录是工作区，在这个目录中的“.git”隐藏文件夹才是版本库。这回概念清晰了吧。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。

如下图所示：



分支和HEAD的概念我们稍后再讲。前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

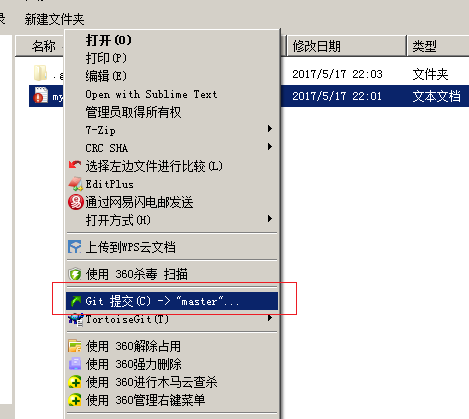
因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

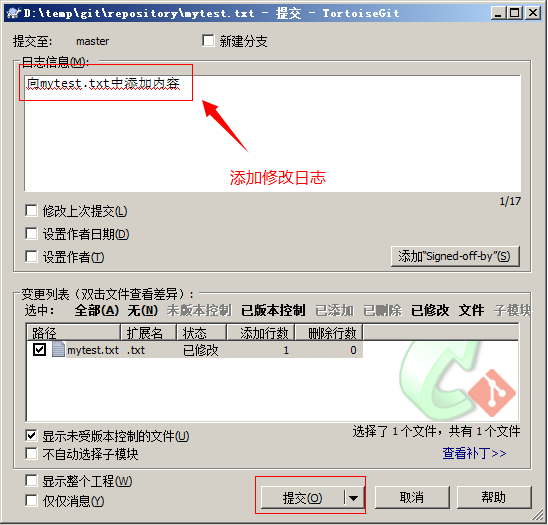
你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

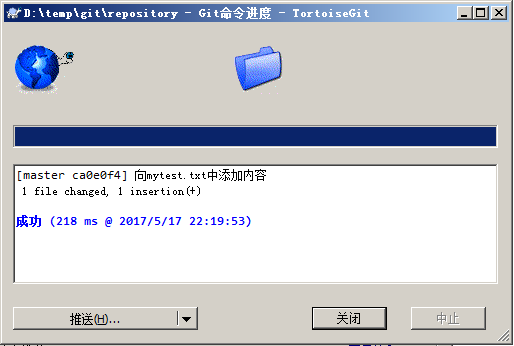
## 修改文件

### 提交修改

被版本库管理的文件不可避免的要发生修改，此时只需要直接对文件修改即可。修改完毕后需要将文件的修改提交到版本库。

在mytest.txt文件上点击右键，然后选择“提交”  




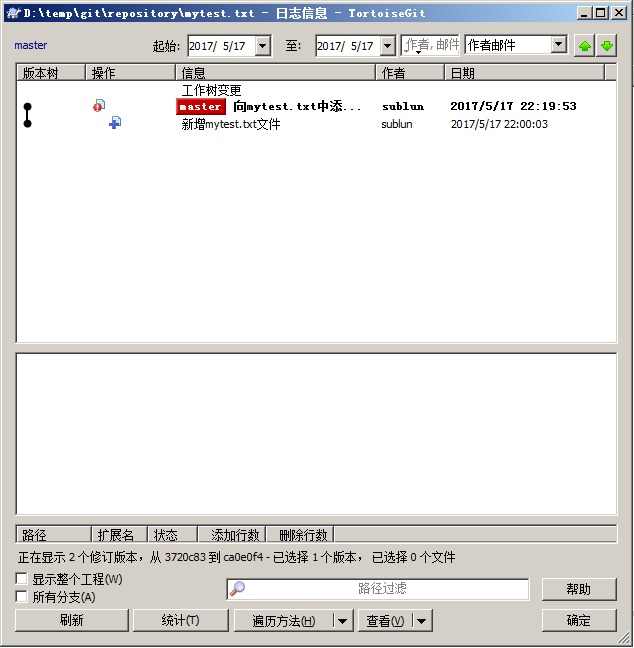


### 查看修改历史

在开发过程中可能会经常查看代码的修改历史，或者叫做修改日志。来查看某个版本是谁修改的，什么时间修改的，修改了哪些内容。

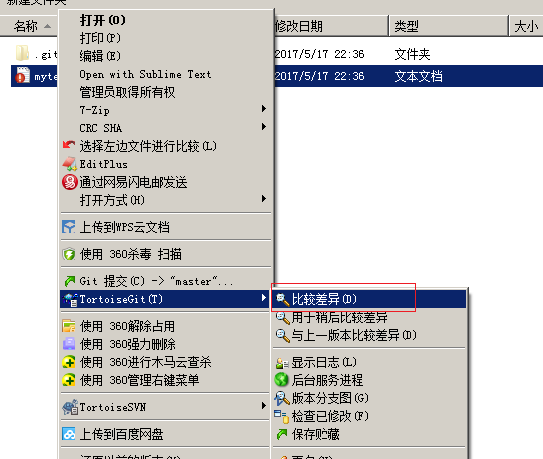
可以在文件上点击右键选择“显示日志”来查看文件的修改历史。

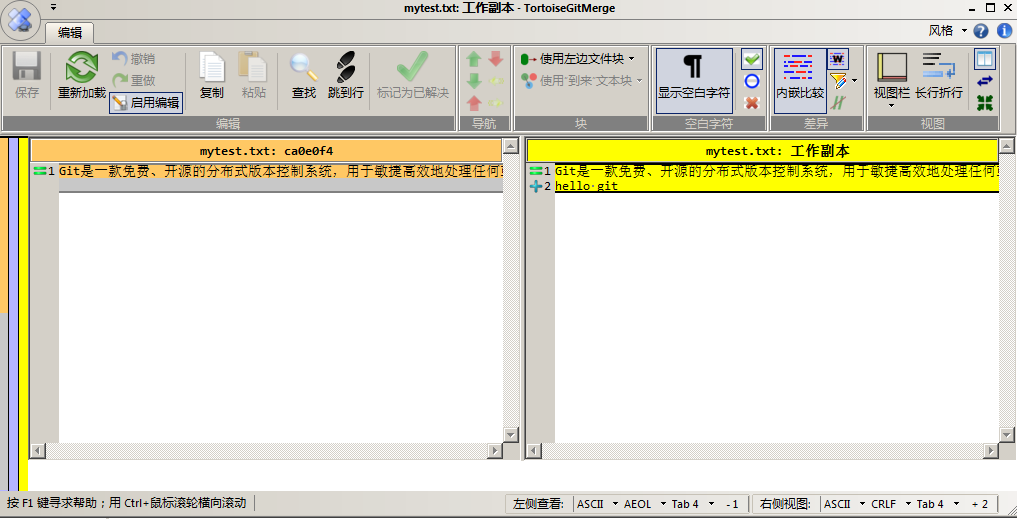




### 差异比较

当文件内容修改后，需要和修改之前对比一下修改了哪些内容此时可以使用“比较差异功能”

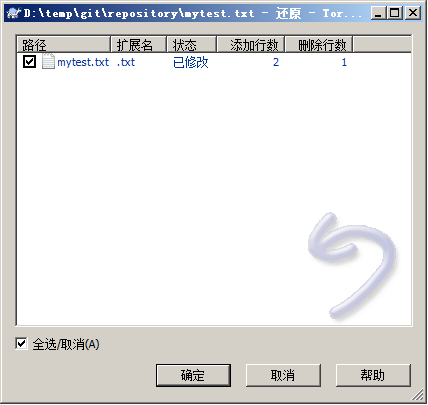


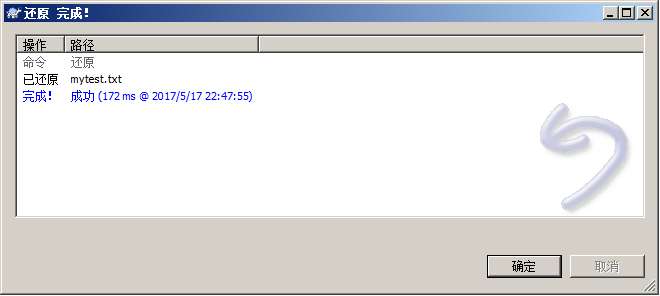


### 还原修改

当文件修改后不想把修改的内容提交，还想还原到未修改之前的状态。此时可以使用“还原”功能







**注意：此操作会撤销所有未提交的修改，所以当做还原操作是需要慎重慎重！！！**

## 删除文件

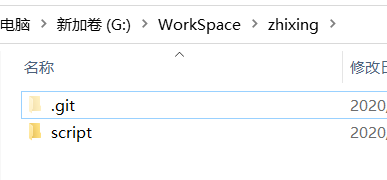
需要删除无用的文件时可以使用git提供的删除功能直接将文件从版本库中删除。



## 案例：将代码提交到版本库

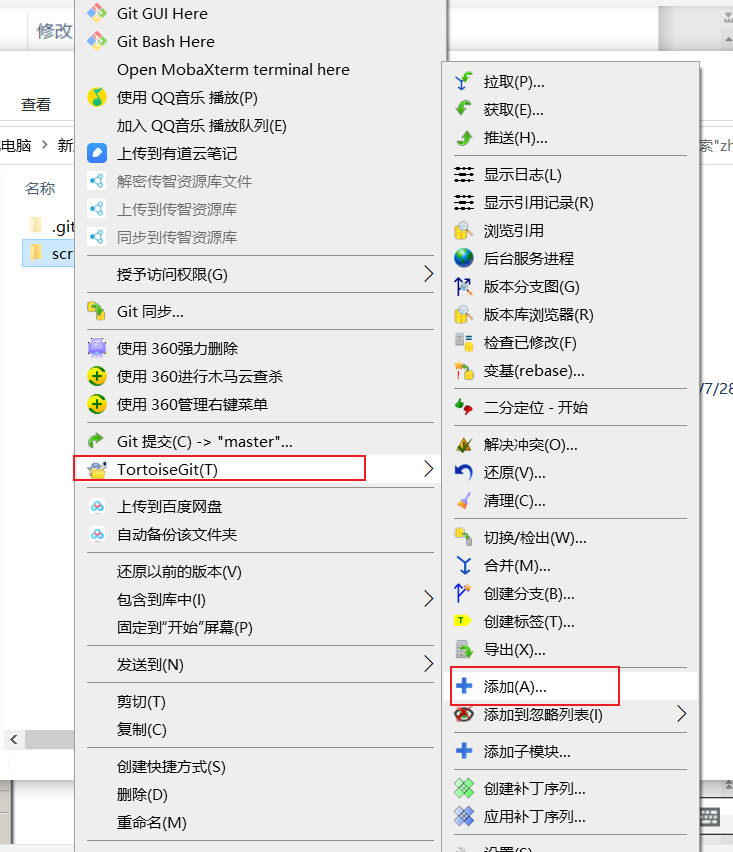
### 编写代码

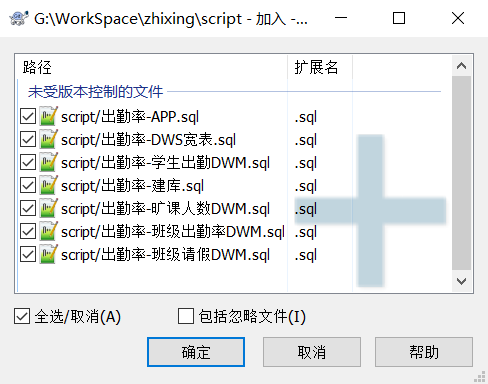
将代码复制到工作目录中

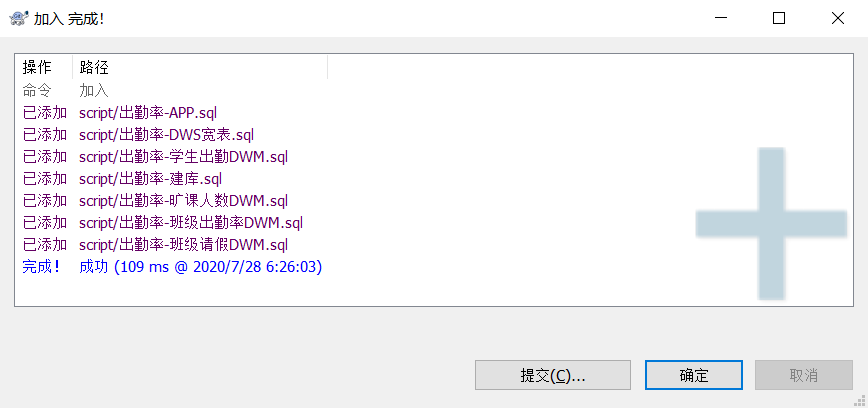


### 添加暂存区

将工程添加到暂存区。





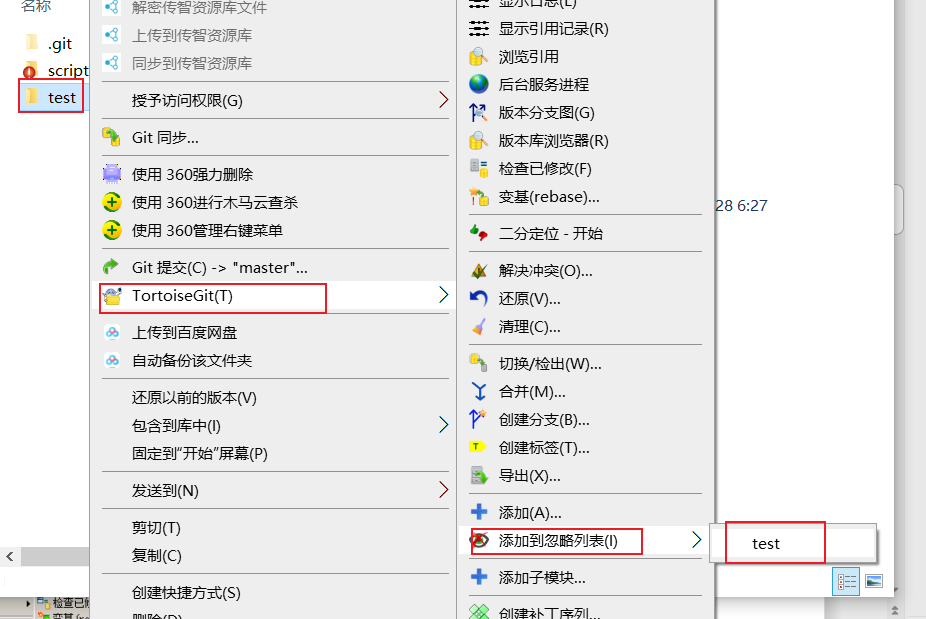


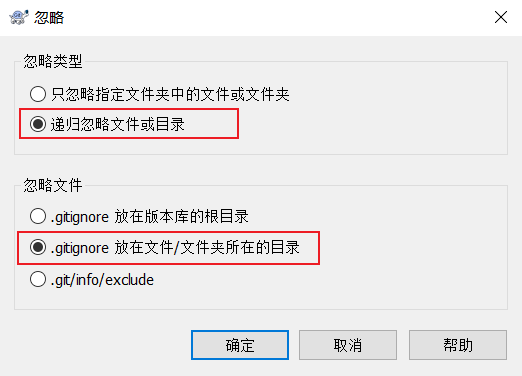
点击确定完成暂存区添加。

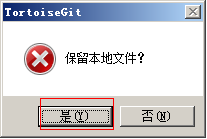
### 忽略文件或文件夹

在此工程中，并不是所有文件都需要保存到版本库中的例如“test”目录及目录下的文件就可以忽略。好在Git考虑到了大家的感受，这个问题解决起来也很简单，在Git工作区的根目录下创建一个特殊的.gitignore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。

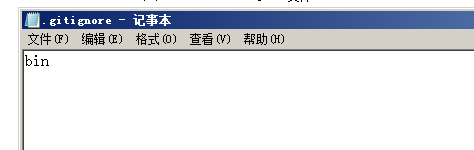
如果使用TortoiseGit的话可以使用菜单项直接进行忽略。







选择保留本地文件。完成后在此文件夹内会多出一个.gitignore文件，这个文件就是文件忽略文件，当然也可以手工编辑。其中的内容就是把test目录忽略掉。



### 提交代码

将代码添加到master分支上，其中.gitignore文件也需要添加到暂存区，然后提交到版本库。

# 远程仓库

## 搭建私有Git服务器

远程仓库实际上和本地仓库没什么不同，纯粹为了7x24小时开机并交换大家的修改。

GitHub就是一个免费托管开源代码的远程仓库。但是对于某些视源代码如生命的商业公司来说，既不想公开源代码，又舍不得给GitHub交保护费，那就只能自己搭建一台Git服务器作为私有仓库使用。

公司大多数都在使用带有**web界面**的**Gitlab**作为内部的Git服务器。

Gitlab安装步骤详见：【Home\讲义\第11章 Git版本控制\GitLab服务器安装.docx】

已安装虚拟机：【Home\资料\Git安装包\git服务器\GitLab.zip】



WEB访问地址：<http://192.168.52.239/>

账号：root

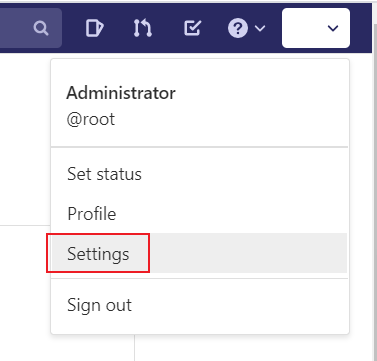
密码：12345678

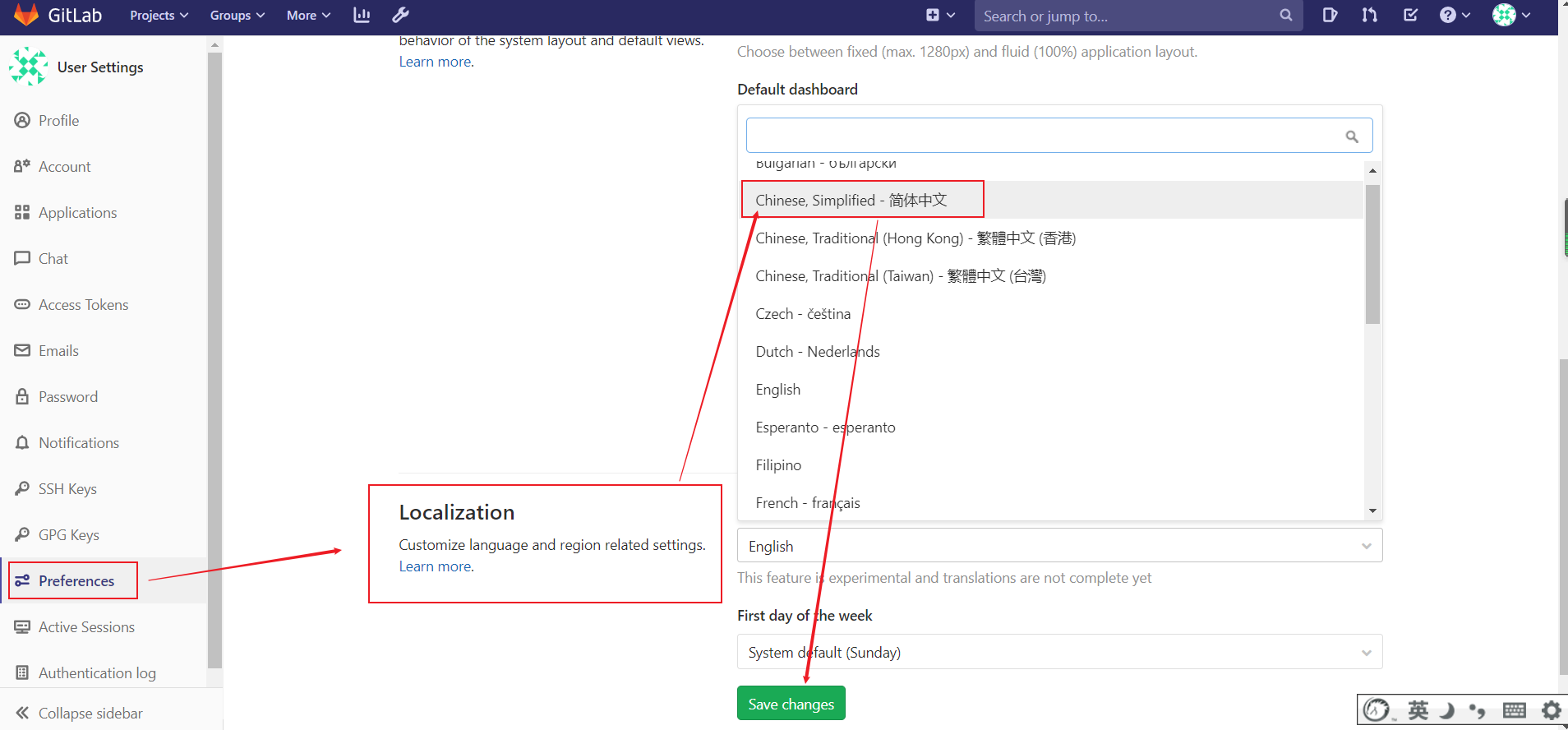
## 添加远程库

现在我们已经在本地创建了一个Git仓库，又想让其他人来协作开发，此时就可以把本地仓库同步到远程仓库，同时还增加了本地仓库的一个备份。

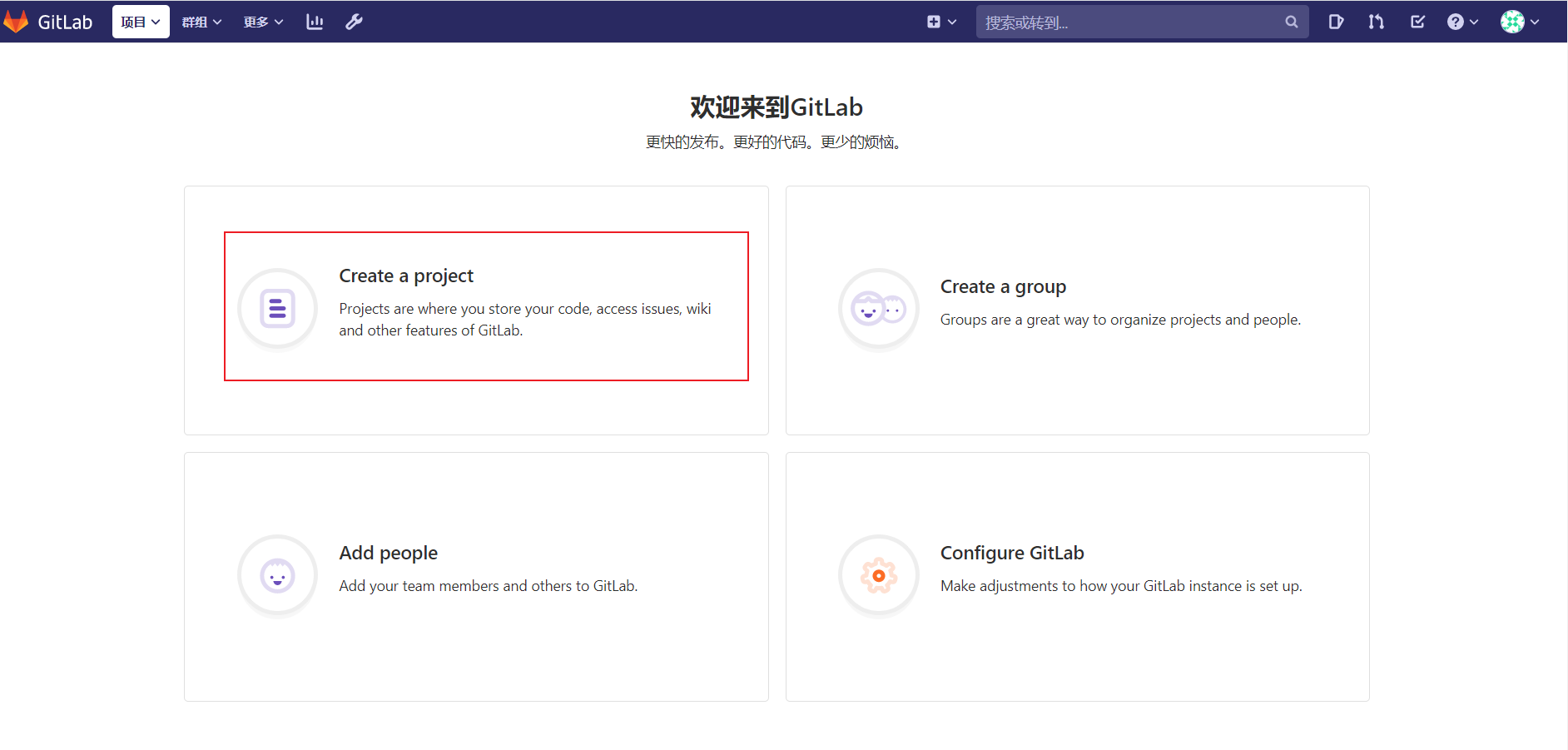
常用的远程仓库就是github(<https://github.com/>)及gitlab，接下来我们演示如何将本地代码同步到gitlab。

### 设置中文





### 创建仓库

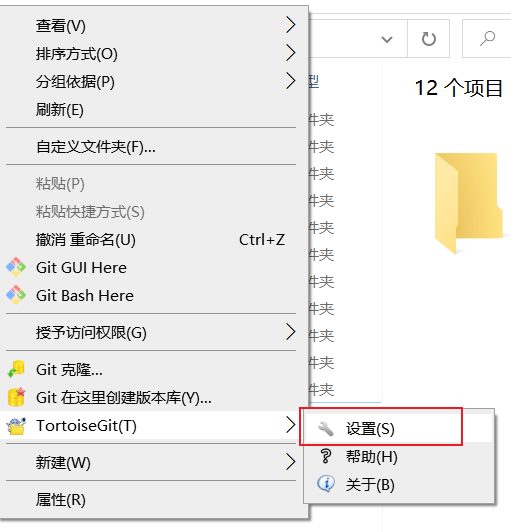




点击“新建项目”按钮仓库就创建成功了。

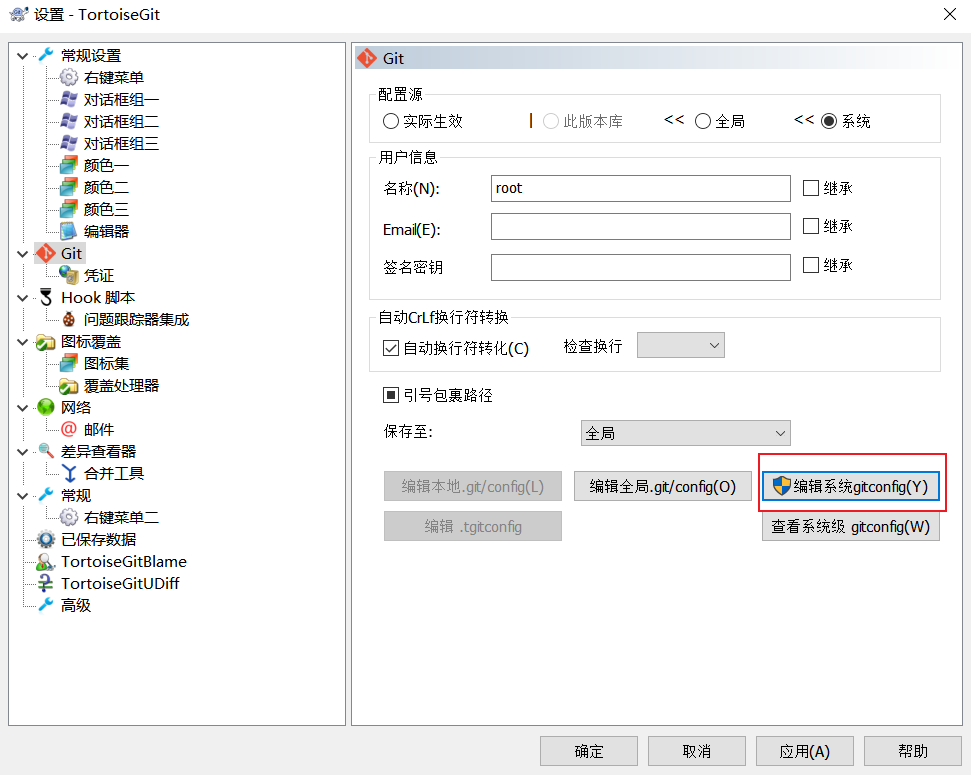
Gitlab支持两种同步方式“https/http”和“ssh”。如果使用https/http很简单基本不需要配置就可以使用，需要设置缓存密码，否则每次提交代码和下载代码时都需要输入用户名和密码。如果使用ssh方式就需要客户端先生成一个密钥对，即一个公钥一个私钥。然后还需要把公钥放到githib的服务器上。Http方式相对简单，用的较多。

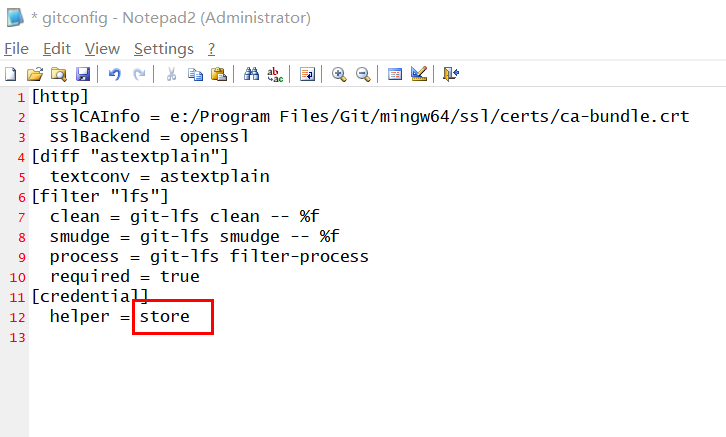
## 设置记住密码



在弹出的.gitconfig文件里，让[credential]的helper = store

|  |
| --- |
| [credential]  helper = store |



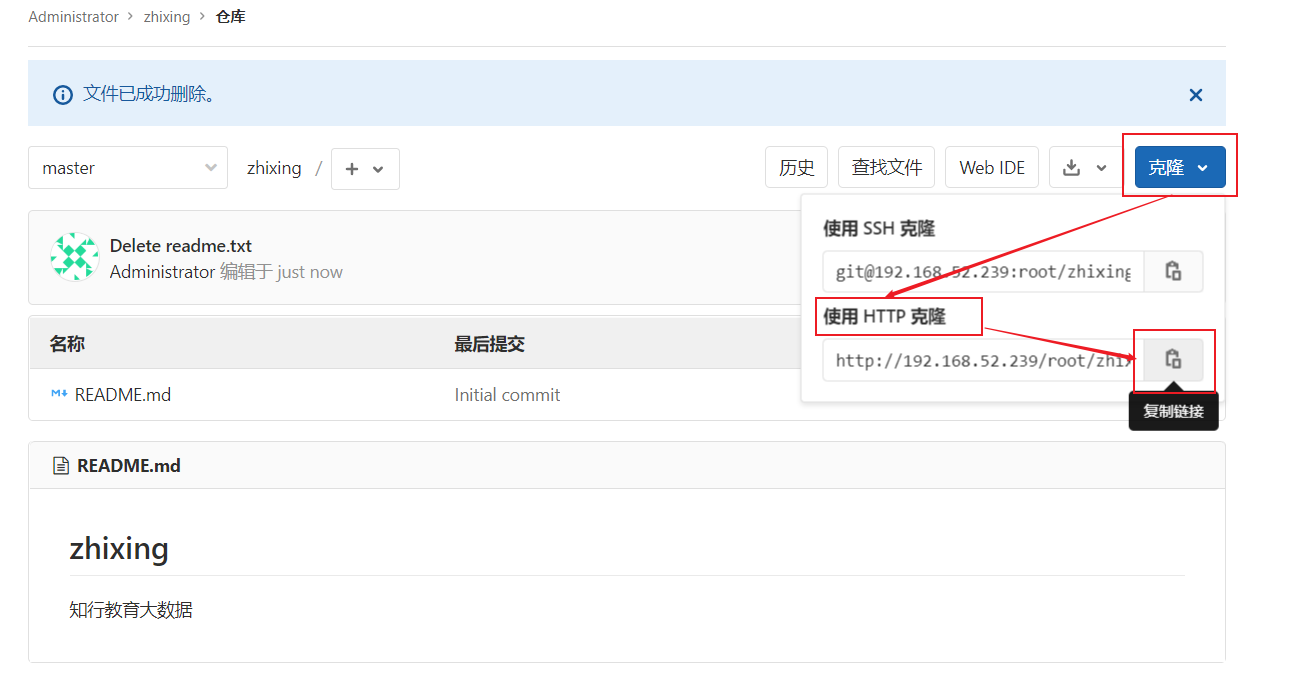


## 从远程仓库克隆

克隆远程仓库也就是从远程把仓库复制一份到本地，克隆后会创建一个新的本地仓库。选择一个任意部署仓库的目录，然后克隆远程仓库。

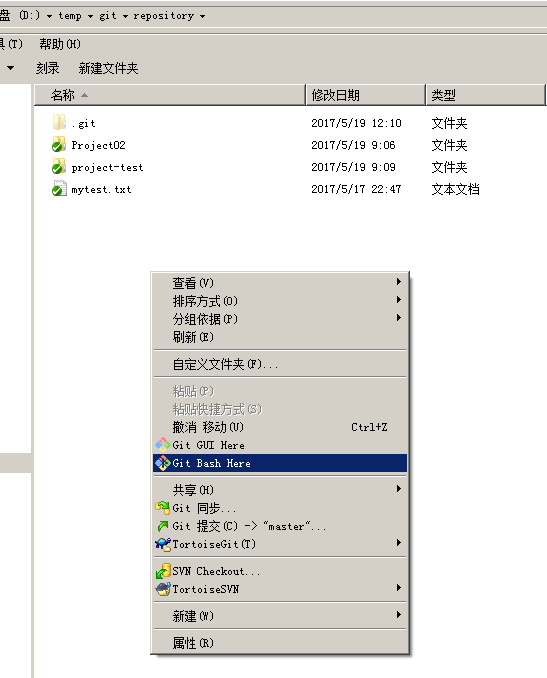
### 使用git bash：

1. 复制远程http地址：



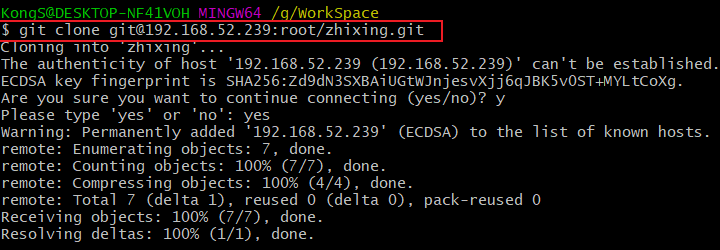
1. 执行克隆命令

找一个英文目录作为本地仓库（G:\WorkSpace）点击右键选择“Git Bash Here”，启动git bash程序。



然后在git bash中执行如下语句：

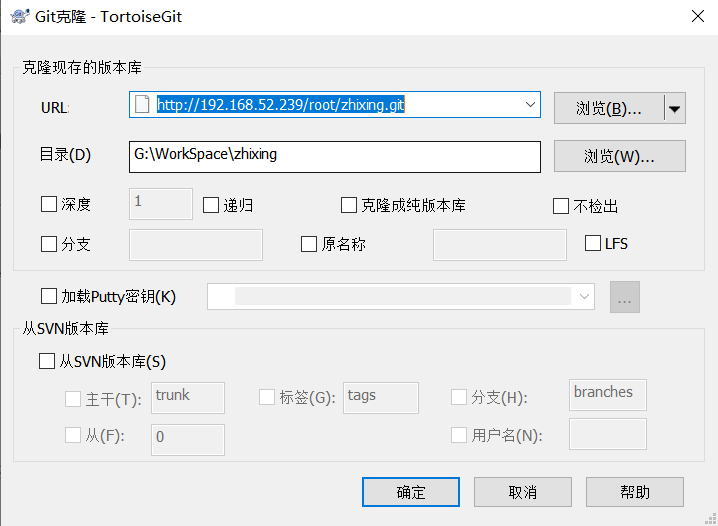
git clone http://192.168.52.239/root/zhixing.git

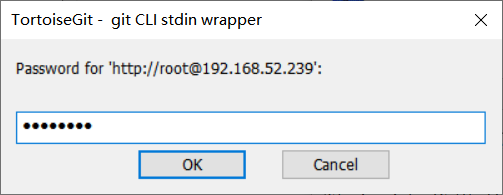


### 使用TortoiseGit：

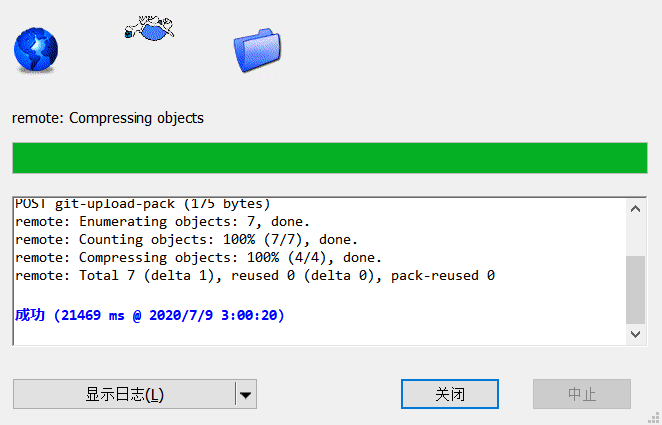
再次克隆之前，记得删除之前下载好的目录。

在任意目录点击右键：





输入Gitlab仓库的账号和密码，点击OK。



下载完成。

## 从远程仓库取代码

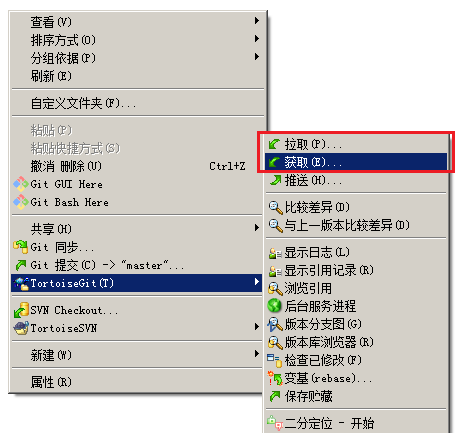
Git中从远程的分支获取最新的版本到本地有这样2个命令：

1. git fetch：相当于是从远程获取最新版本到本地，不会自动merge（合并代码）
2. git pull：相当于是从远程获取最新版本并merge到本地

上述命令其实相当于git fetch 和 git merge

在实际使用中，git fetch更安全一些，git pull更智能快速一些。

如果使用TortoiseGit的话可以从右键菜单中点击“拉取”（pull）或者“获取”（fetch）



## 向远程仓库推送代码

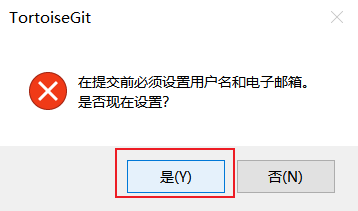
### 新建目录和文件



### 提交修改

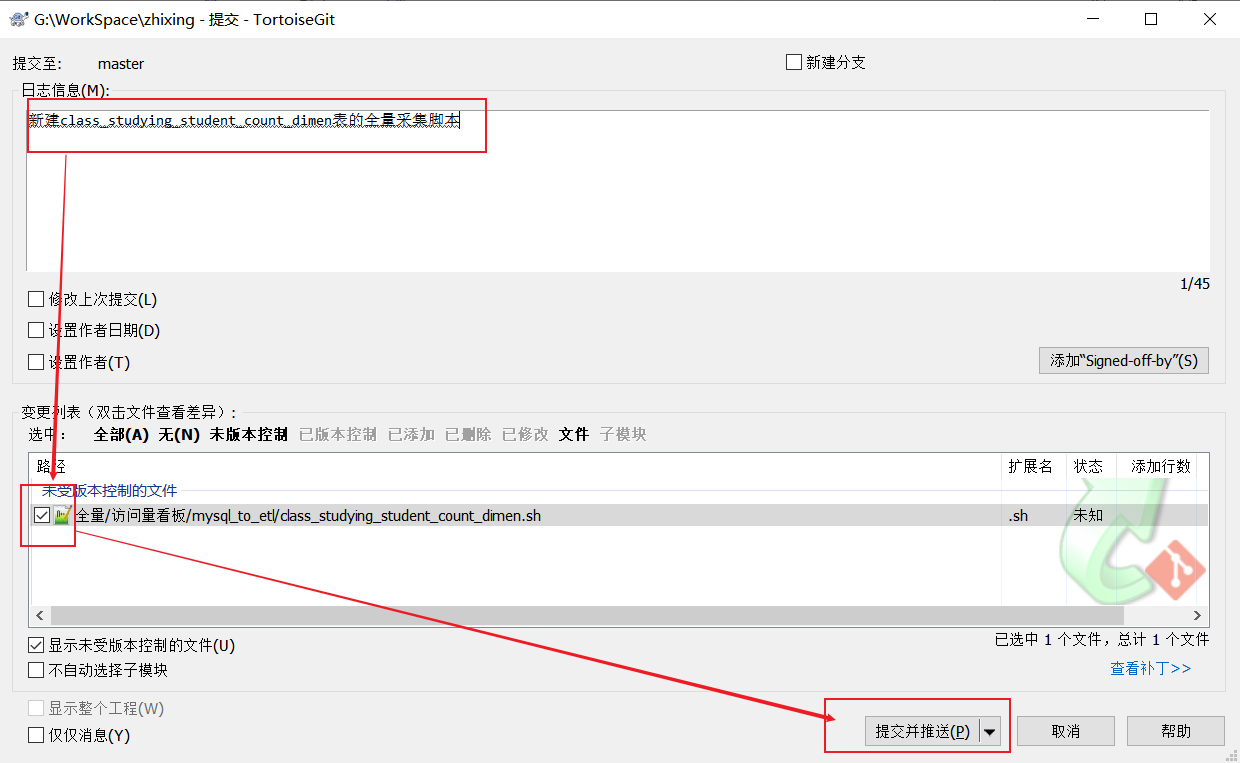


提示设置邮箱

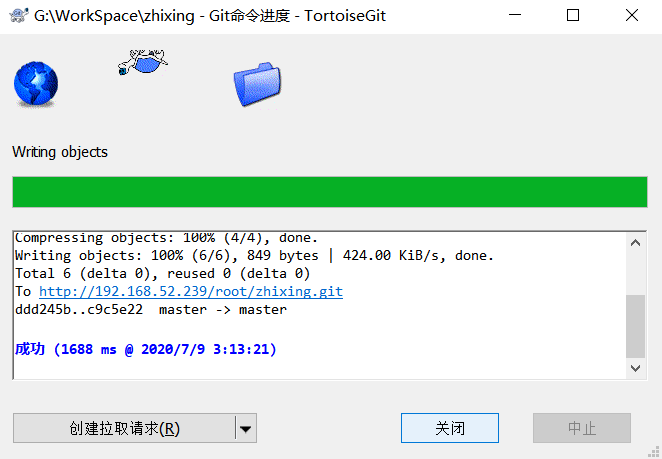




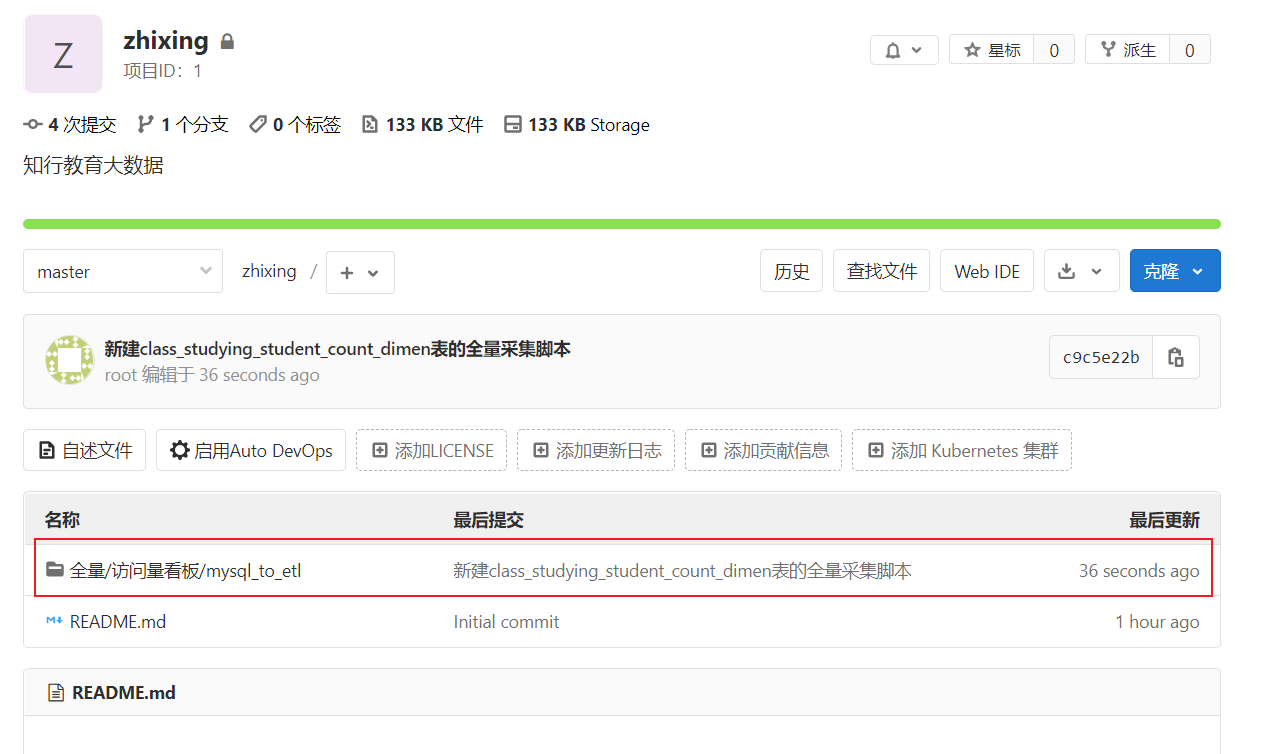
### 提交前填写备注信息



### 提交成功



### 在仓库中验证文件是否存在





# 分支管理

## 分支的实现原理

在我们每次的提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD指针指向的就是当前分支。

一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点：



每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长。

当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！

不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！

合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：

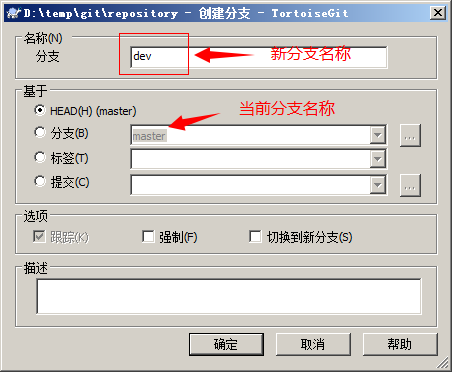


## 使用TortoiseGit实现分支管理

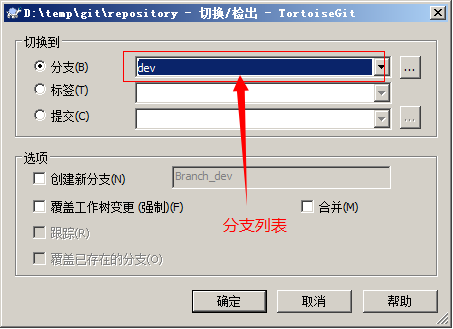
使用TortoiseGit管理分支非常简单。

### 创建分支

在本地仓库文件夹中点击右键，然后从菜单中选择“创建分支”：



如果想创建完毕后直接切换到新分支可以勾选“切换到新分支”选项或者从菜单中选择“切换/检出”来切换分支：

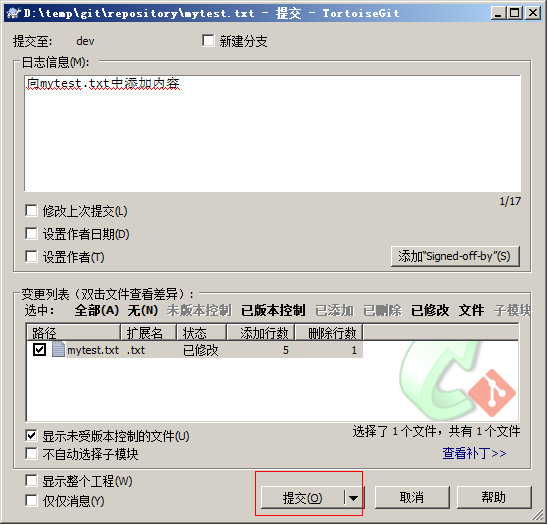


### 合并分支

#### Dev修改提交

分支切换到dev后就可以对工作区的文件进行修改，然后提交到dev分支，原来的master分支不受影响。例如我们修改mytest.txt中的内容，然后提交到dev分支。

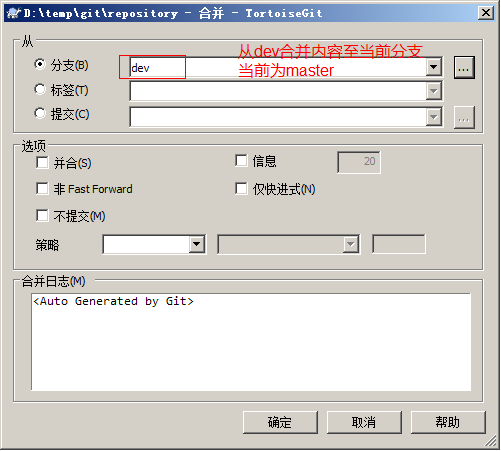




#### 合并dev更新到master

切换到master分支后还是原来的内容：



将dev分支的内容合并到master分支，当前分支为master。从右键菜单中选择“合并”：

再查看mytest.txt的内容就已经更新了：



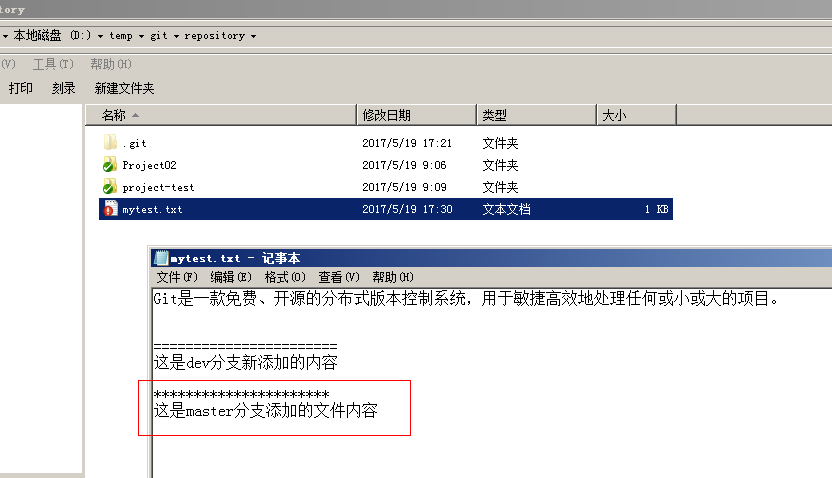
## 解决冲突

刚才两个分支中编辑的内容都是相互独立互不干扰的，那么如果在两个分支中都对同一个文件进行编辑，然后再合并，就有可能会出现冲突。

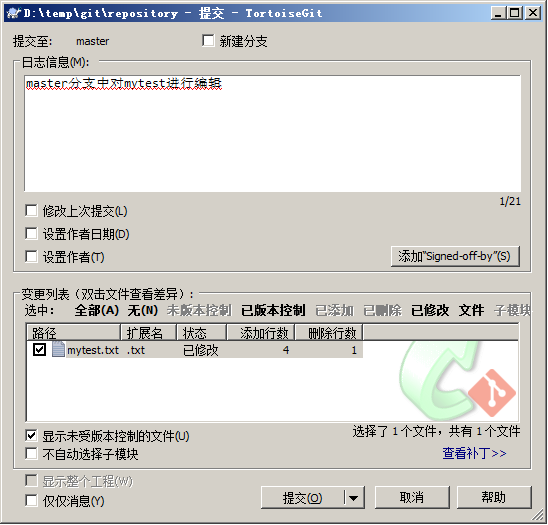
### 冲突场景

#### Master分支编辑提交

在master分支中对mytest.txt进行编辑：

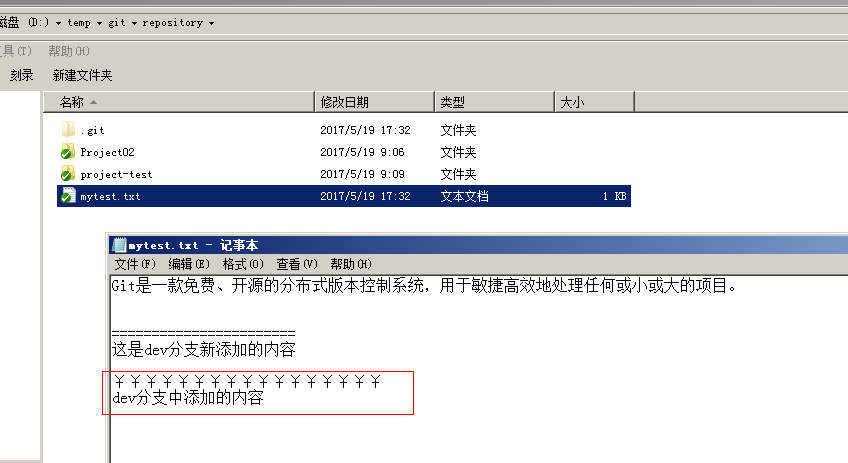


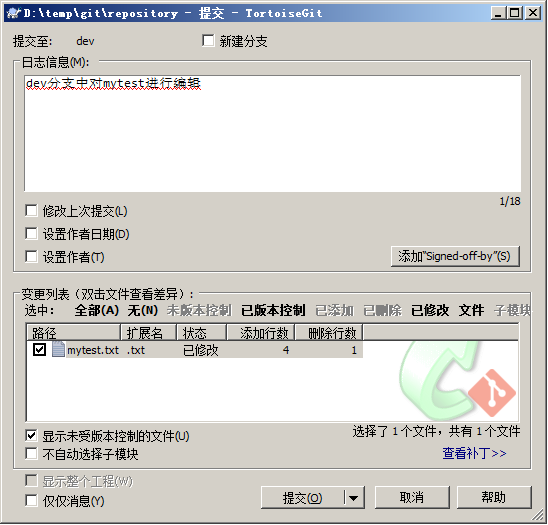
然后提交到版本库。



#### Dev分支编辑提交

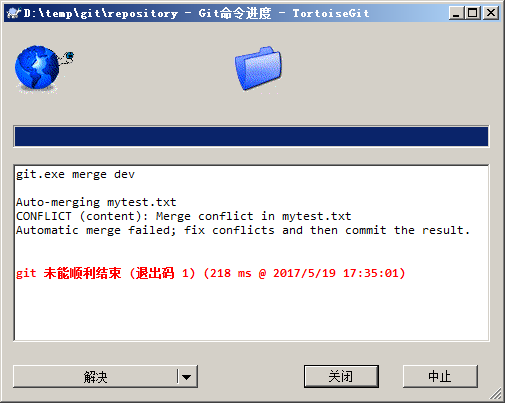
切换到dev分支，对mytest.txt进行编辑：



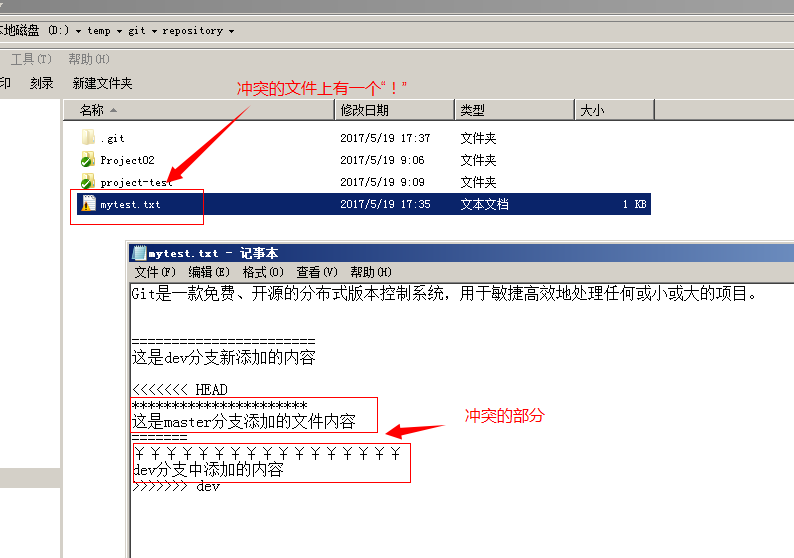


#### 合并分支

最后进行分支合并，例如将dev分支合并到master分支。需要**先切换到master**分支然后进行分支合并。

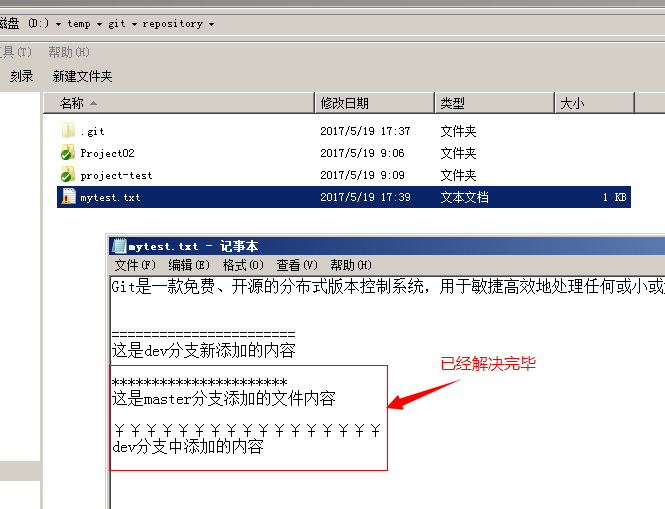


出现版本冲突。



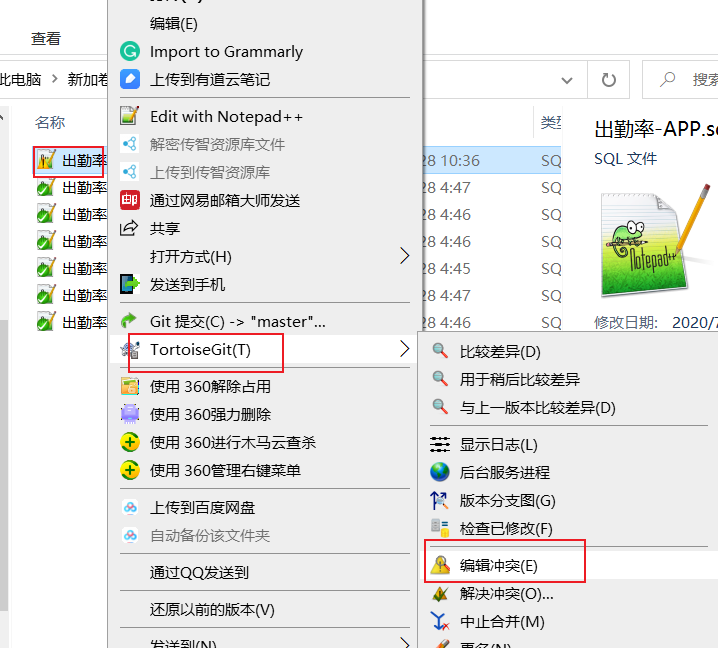
### 解决冲突

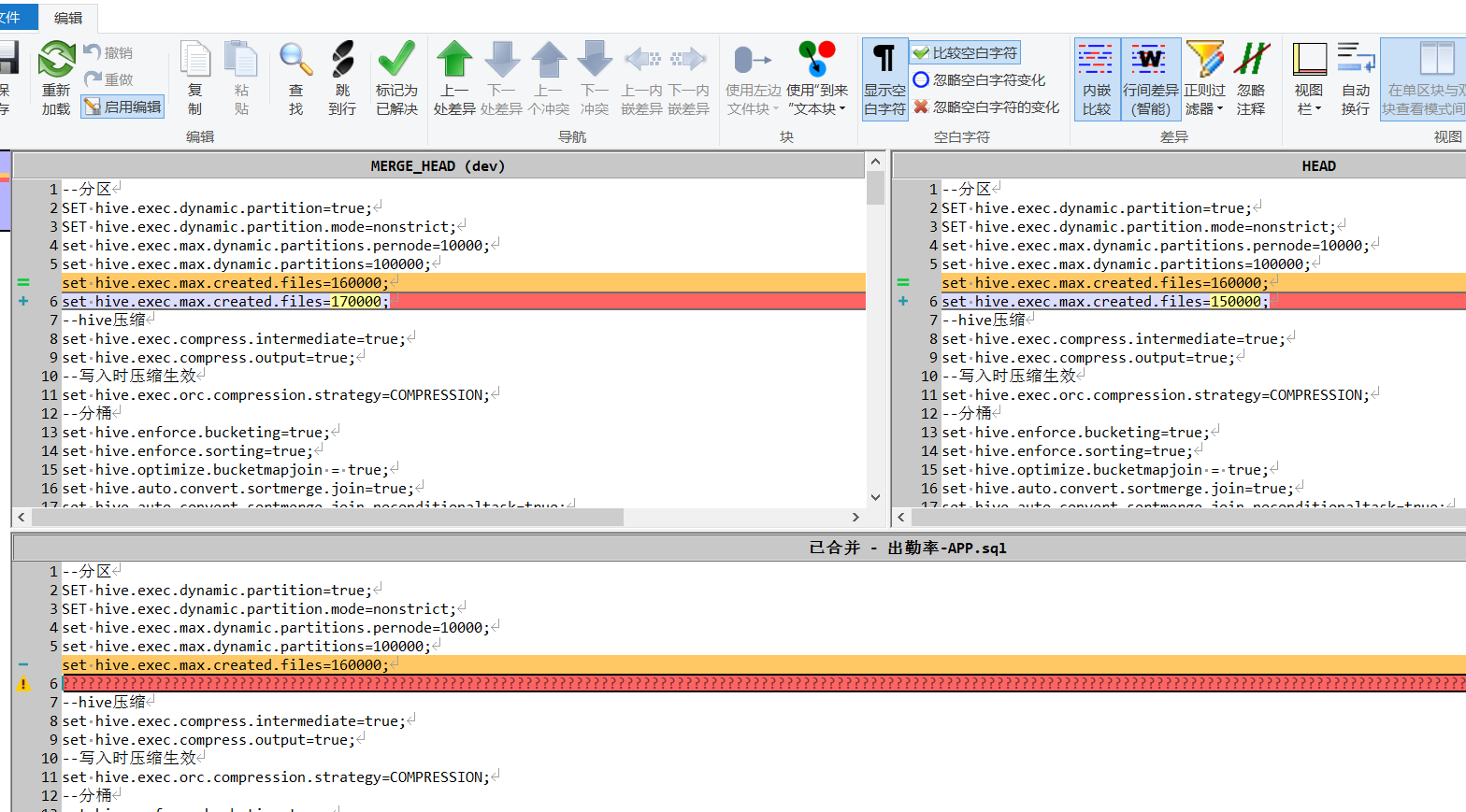
#### 手动解决



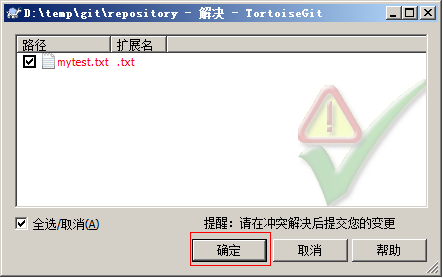
#### Tortoise可视化解决

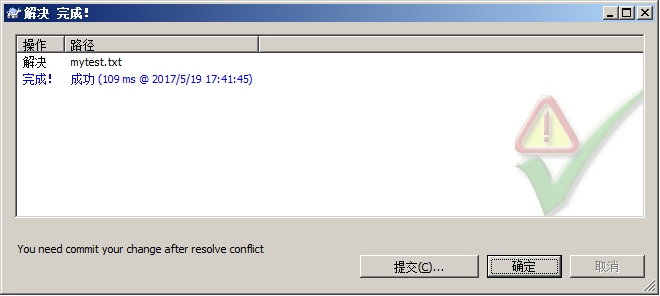
也可以通过Tortoise来编辑冲突：





解决冲突后，在冲突文件上单机右键选择“解决冲突”菜单项：





把冲突解决完毕的文件提交到版本库就可以了。