

# Git

## 安装

<https://git-scm.com/download>

在安装过程中，除了修改安装路径，一路默认即可

## git 功能

Git GUI Here

git 图形界面

Git Bash Here

git 命令行（可以操作一些linux命令）

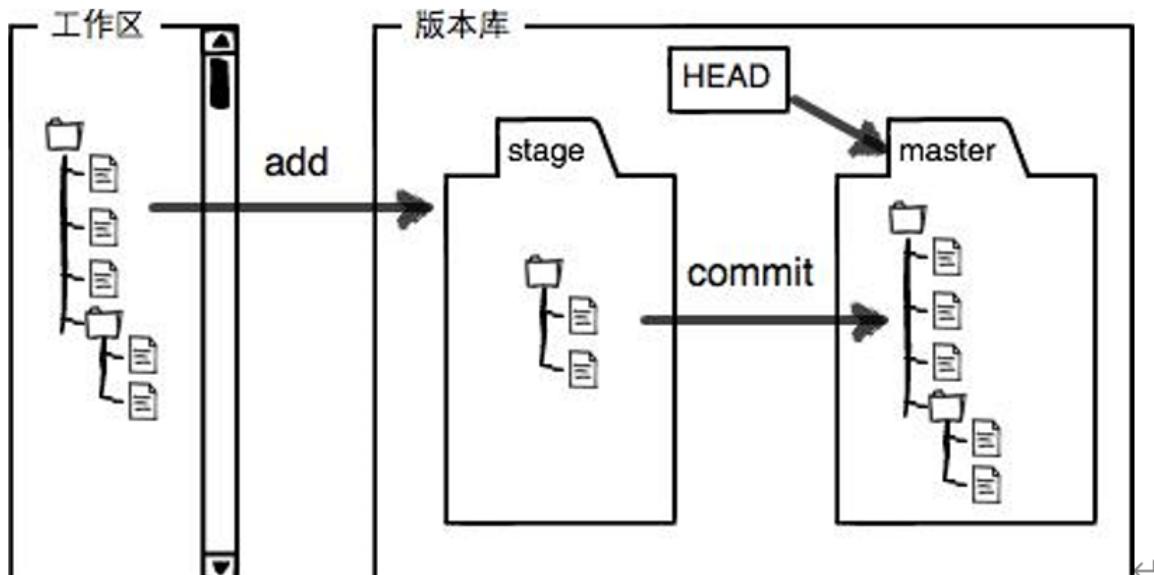
## git命令

git --version    查看git版本命令

## 工作区和暂存区

包含本地仓库的目录就是‘工作目录（工作区）’

stage 暂存区，在版本库中



## TortoiseGit

## 安装

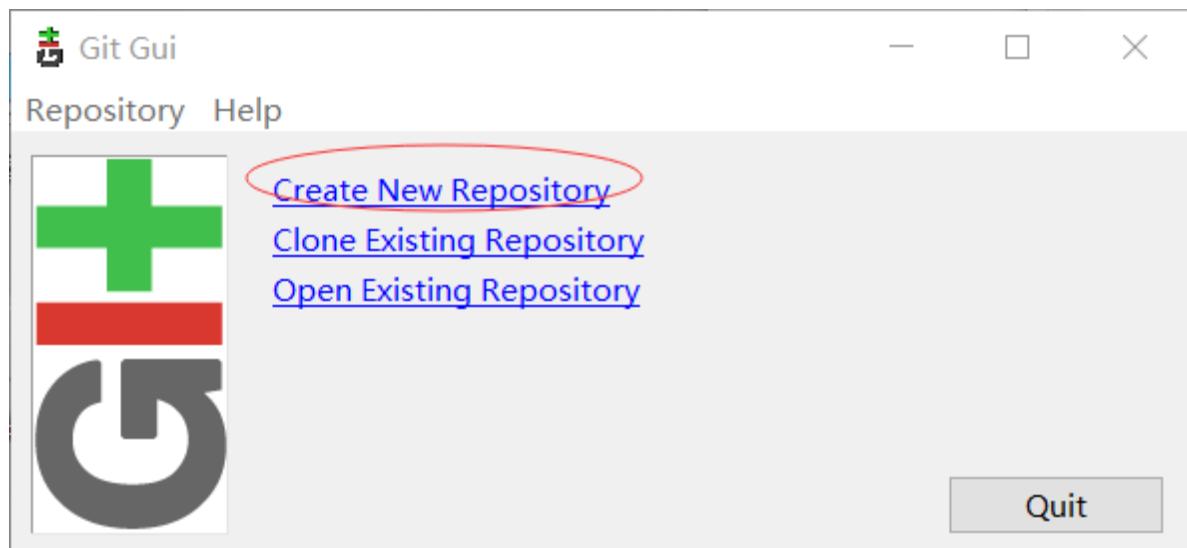
TortoiseGit-2.4.0.2-64bit 除了修改安装路径一路默认即可

可以安装中文汉化

# git使用方法

## 1. 创建本地仓库

<1>



<2>



<2>命令行

创建仓库执行命令：

```
$ git init
```

<3>小乌龟 (TortoiseGit)

右击Git在这里创建版本库

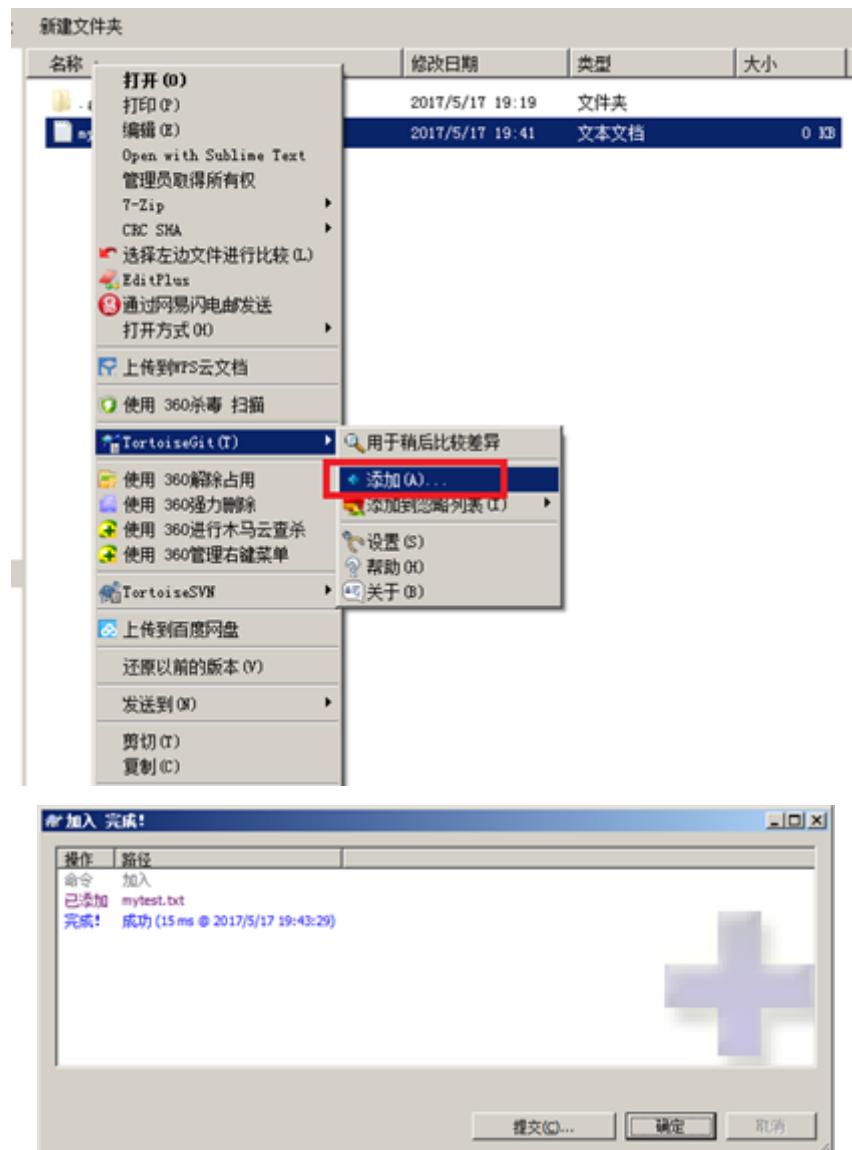
不选 制作纯版本库---->确定

## 2. 添加文件

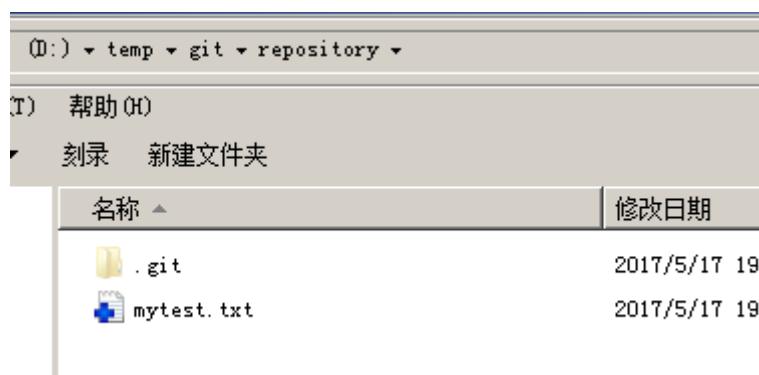
**版本库**: ".git" 目录就是版本库，将来文件都需要保存到版本库中。

**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。

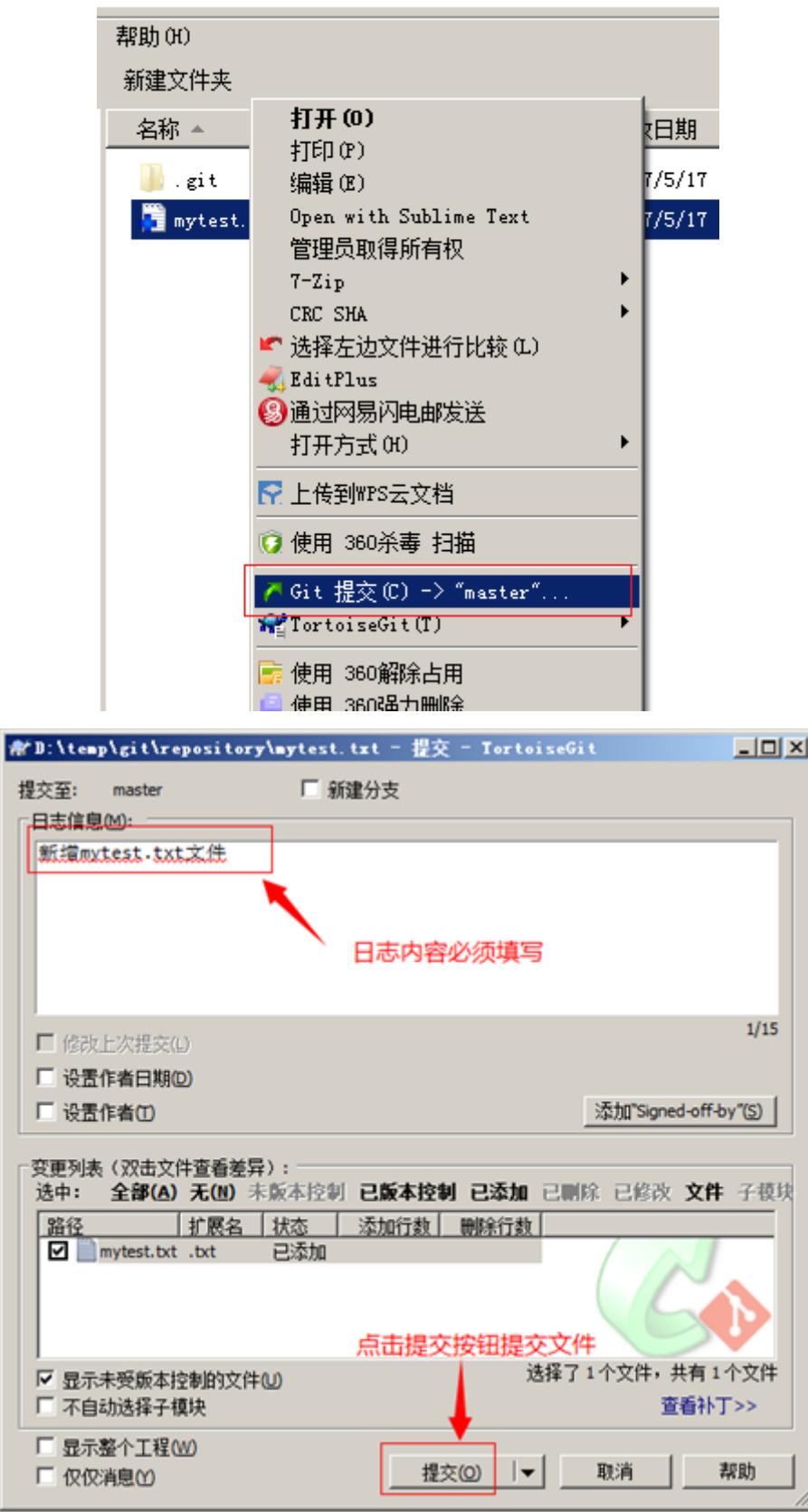
在 F:\mi\_git\_storage\repo1 下创建 hello.txt 然后添加到版本库中

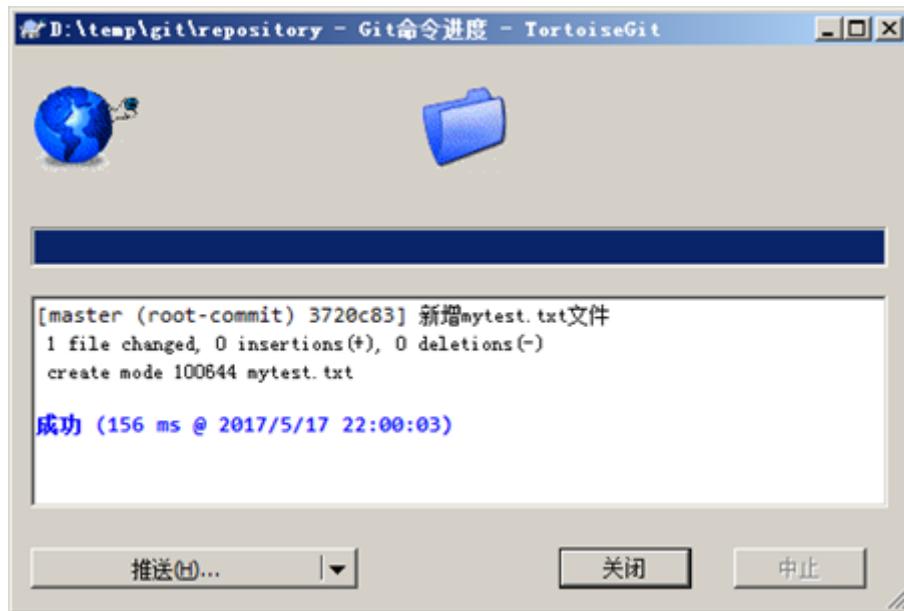


文本文件变为带 '+' 号的图标



需要commit 把文件从暂存区 (stage) 提交到本地仓库中





### 3.删除文件并且还原

<1>当delete删除之后，可以还原

但是，在提交完毕后，删除的文件将不能还原，彻底删除了出去

可以通过版本库浏览器查看

<2>使用小乌龟进行删除

删除 和 使用delete删除效果一样（只删除了本地文件，而本地版本库中文件还存在）

删除并保留本地副本 只是打上删除标记，当提交之后删除版本库中的对应文件，本地文件保存

### 4.将java工程添加到本地版本库

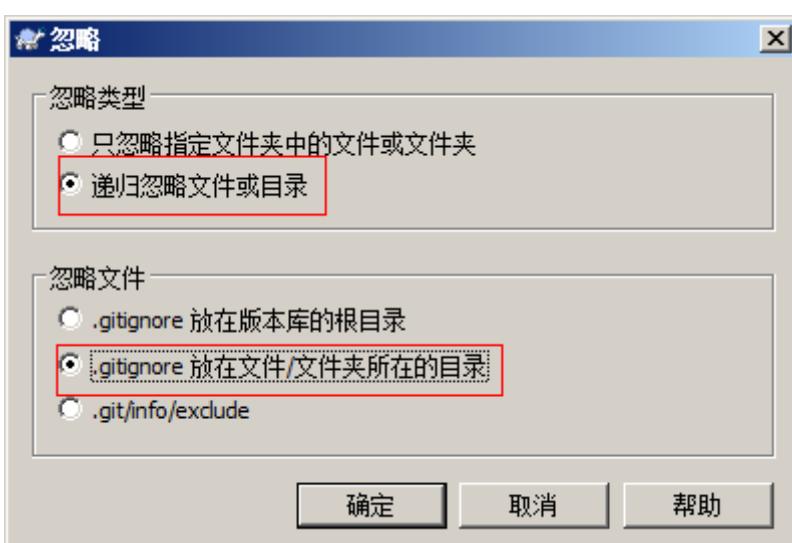
<1>将java工程 xxx 复制到工作目录中（工作区）

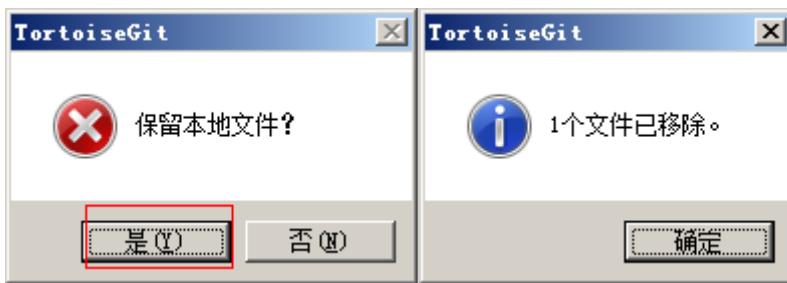
<2>将工程‘add’到暂存区

<3>忽略文件

这是为了将一部分没必要的文件不上传

xxx 文件选中后，右键删除并添加到忽略列表，根据名称删除和忽略





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bin目录忽略掉。

#### <4>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。

## 5. 使用github创建一个远程仓库

### 在github上创建一个仓库

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: sublun / Repository name: mytest

Great repository names are short and memorable. Need inspiration? How about crispy-fortnight.

Description (optional):

Public  
Anyone can see this repository. You choose who can commit.

Private  
You choose who can see and commit to this repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None | ⓘ

Create repository

点击“create repository”按钮仓库就创建成功了。

Github支持两种同步方式“https”和“ssh”。如果使用https很简单基本不需要配置就可以使用，但是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github的服务器上。这两种方式在实际开发中都用应用，所以我们都需要掌握。

## ssh协议

### 什么是ssh?

SSH 为 Secure Shell (安全外壳协议) 的缩写，由 IETF 的网络小组 (Network Working Group) 所制定。SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

### 基于密钥的安全验证

使用ssh协议通信时，推荐使用基于密钥的验证方式。你必须为自己创建一对密匙，并把公用密匙放在需要访问的服务器上。如果你要连接到SSH服务器上，客户端软件就会向服务器发出请求，请求用你的密匙进行安全验证。服务器收到请求之后，先在该服务器上你的主目录下寻找你的公用密匙，然后把它和你发送过来的公用密匙进行比较。如果两个密匙一致，服务器就用公用密匙加密“质询”(challenge) 并把它发送给客户端软件。客户端软件收到“质询”之后就可以用你的私人密匙解密再把它发送给服务器。

### ssh密钥的生成

一台电脑只需要一个密钥即可与github服务器进行连接

当新项目要推送到github时，需要在github新建好对应的仓库，然后设置好对应的URL即可。

在windows下我们可以使用 Git Bash.exe来生成密钥，可以通过开始菜单或者右键菜单打开Git Bash。



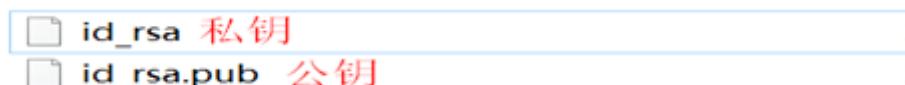
git bash 执行命令,生命公钥和私钥

命令: ssh-keygen -t rsa

```

MINGW64 : /d/temp/git/repository
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa): 跪回车即可
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Administrator/.ssh/id_rsa. 使用默认配置
Your public key has been saved in /c/Users/Administrator/.ssh/id_rsa.pub. 置
The key fingerprint is:
SHA256:8a0c1rulhk5j8WN7+pRouB4Jpv4XINugNO2ZJFRG3A Administrator@PC-201311301552
The key's randomart image is:
+---[RSA 2048]---+
| .oE
| .. o
| +.
| o o o o
| o So= + +
| o *o+. *o+.o.
| .= +=+=o.o
| .o= .o=.
| o+..+o.o...
+---[SHA256]---+
Administrator@PC-201311301552 MINGW64 /d/temp/git/repository (master)
$
```

执行命令完成后，在window本地用户.ssh目录C:\Users\用户名.ssh下面生成如下名称的公钥和私钥



## ssh密钥配置

密钥生成后需要在github上配置密钥本地才可以顺利访问。

The screenshot shows the GitHub user interface for managing SSH keys. The top navigation bar includes 'Signed in as sublun'. The main menu has options like 'Watch', 'Graphs', and 'Settings'. The 'Settings' option is highlighted with a red box. Below the menu, the 'Personal settings' sidebar lists 'Profile', 'Account', 'Emails', 'Notifications', 'Billing', 'SSH and GPG keys' (which is selected and highlighted with a red box), 'Security', 'Blocked users', 'Repositories', 'Organizations', 'Saved replies', 'Authorized OAuth Apps', 'Authorized Integrations', and 'Installed Integrations'. The main content area is titled 'SSH keys' and contains a message: 'There are no SSH keys with access to your account.' A red box highlights the 'New SSH key' button. Below it, a new key is being added with the title 'subl' and the public key content:

```

-----BEGIN RSA PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAQIDQD971OpGhz3wbnx1486+QDx5hiCMCeh+45cGM/bcLH5opjWf5g
HxGtq7m+8xs/QG5alw+4KAf-d9WbUe+envvokS0logKgqL57Ajh+QWtk3T20f1+4+1U1Mkyv5FY5MZhx06
XYalssbaTbCnogY7+wv68MP979Q47radhwpojh5/WfAxApDUjMAzpolc374016oiU40fe3oWAlUgPU2s2
Nljy+uyTo+FgNhULmLjcvbDIG335efzhWlgDmnvwZeZKX0+eOPgw45BFJOnaeRbJspel9s05NstQ0000H5
nAdACMWB1Y7LD2DikV Administrator@PC-201311301552
-----END RSA PUBLIC KEY-----

```

At the bottom of the 'SSH keys' section, there is a note: 'Check out our guide to generating SSH keys or troubleshoot common SSH Problems.' A red box highlights the 'Add SSH key' button.

在key部分将id\_rsa.pub文件内容添加进去，然后点击“Add SSH key”按钮完成配置。

## 同步到远程仓库

同步到远程仓库可以使用git bash也可以使用tortoiseGit

### 1.ssh方式进行同步

<1>使用git bush here 命令进行操作

在仓库所在的目录点击右键选择“Git Bash Here”，启动git bash程序。

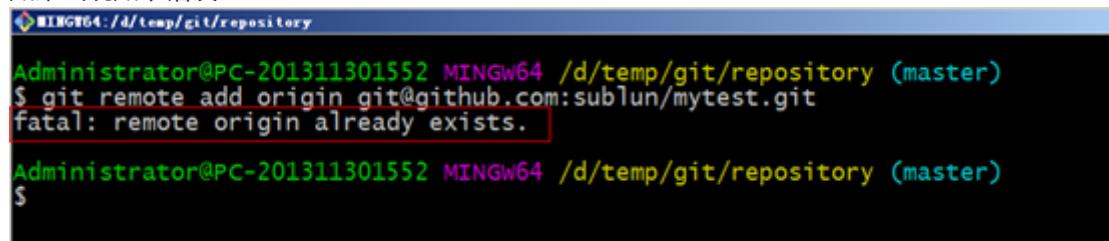
然后在git bash中执行如下语句：

```
git remote add origin git@github.com:sublun/mytest.git
```

```
git push -u origin master
```

注意：其中加粗字体部分需要替换成个人的用户名。

如果出现如下错误：



```
MINGW64 /d/temp/git/repository
Administrator@PC-201311301552 MINGW64 /d/temp/git/repository (master)
$ git remote add origin git@github.com:sublun/mytest.git
fatal: remote origin already exists.

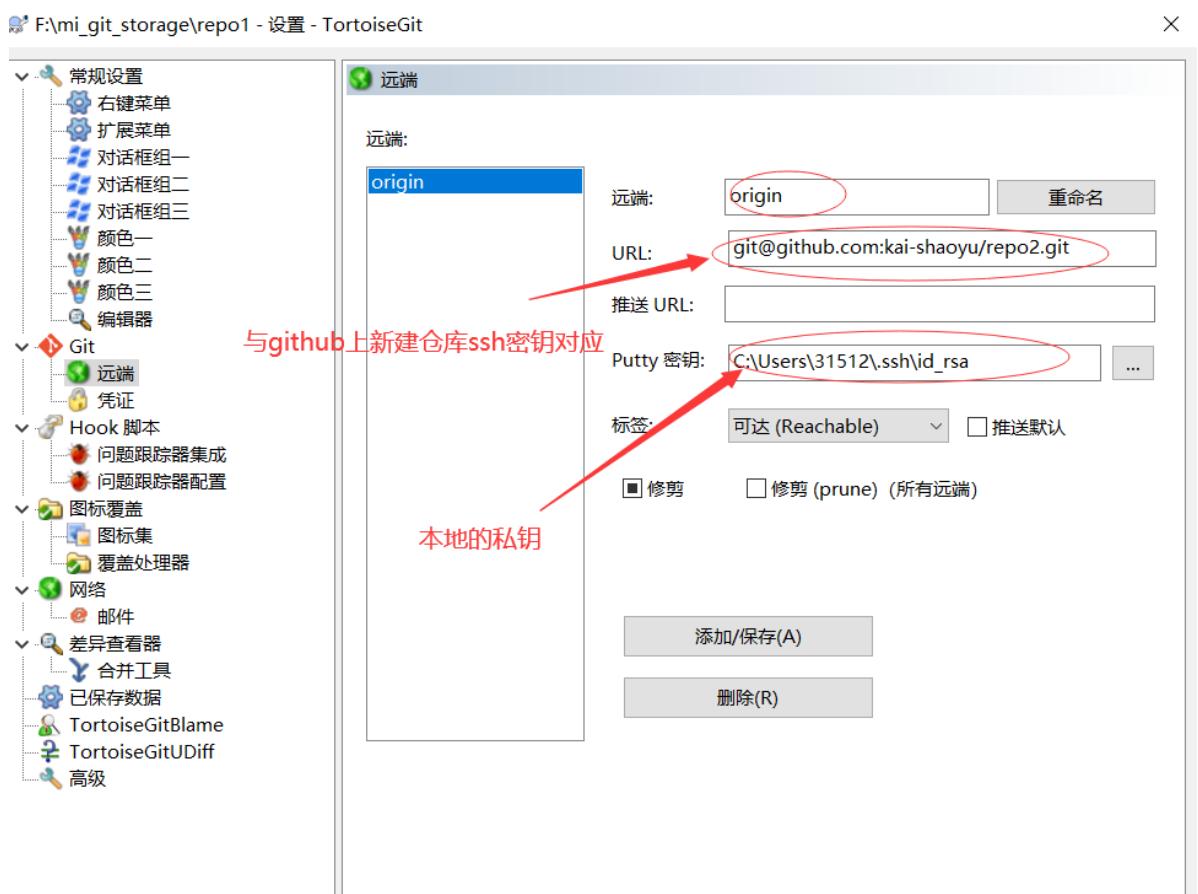
Administrator@PC-201311301552 MINGW64 /d/temp/git/repository (master)
$
```

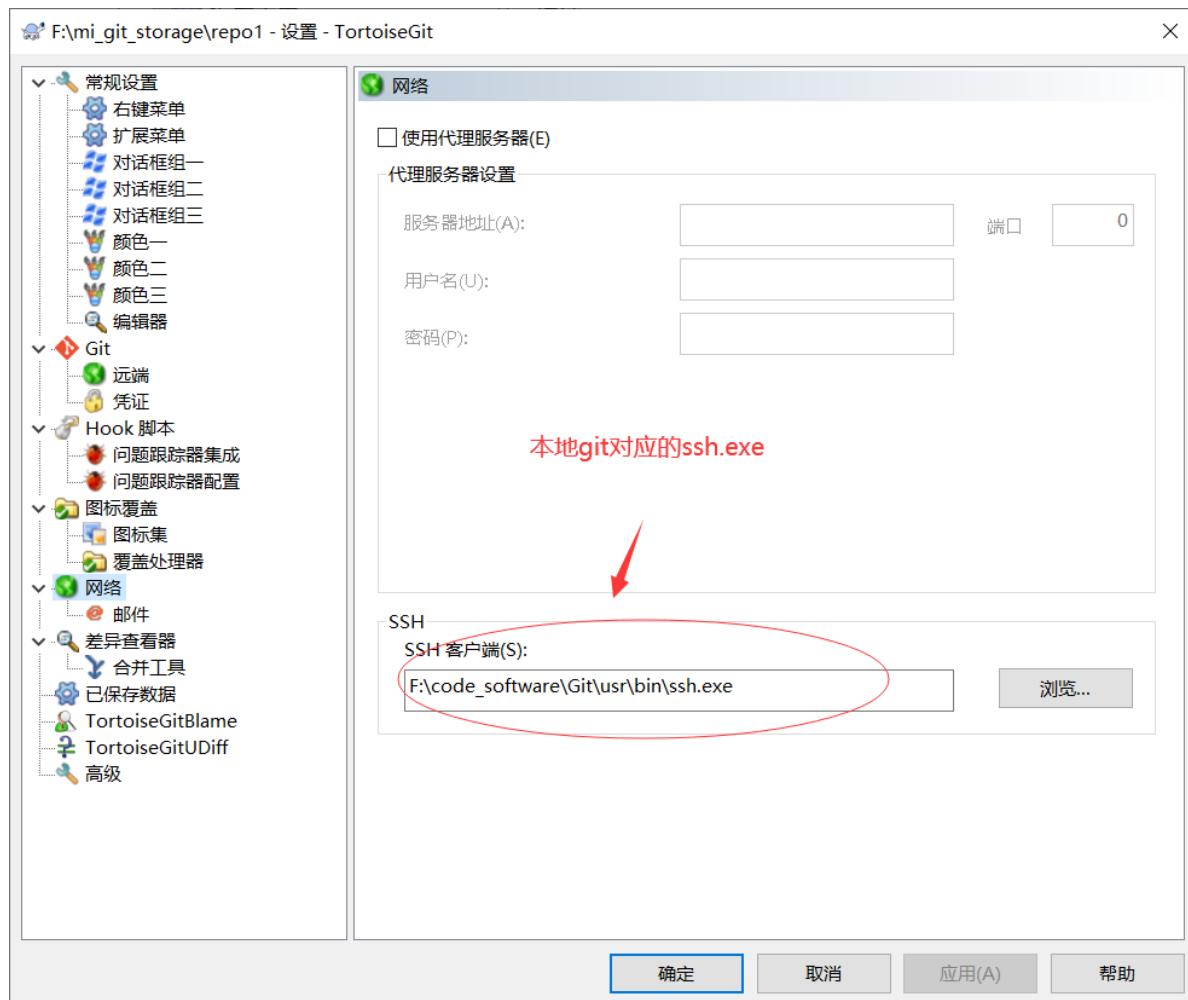
可以先执行如下命令，然后再执行上面的命令

```
$ git remote rm origin
```

<2>使用小乌龟同步





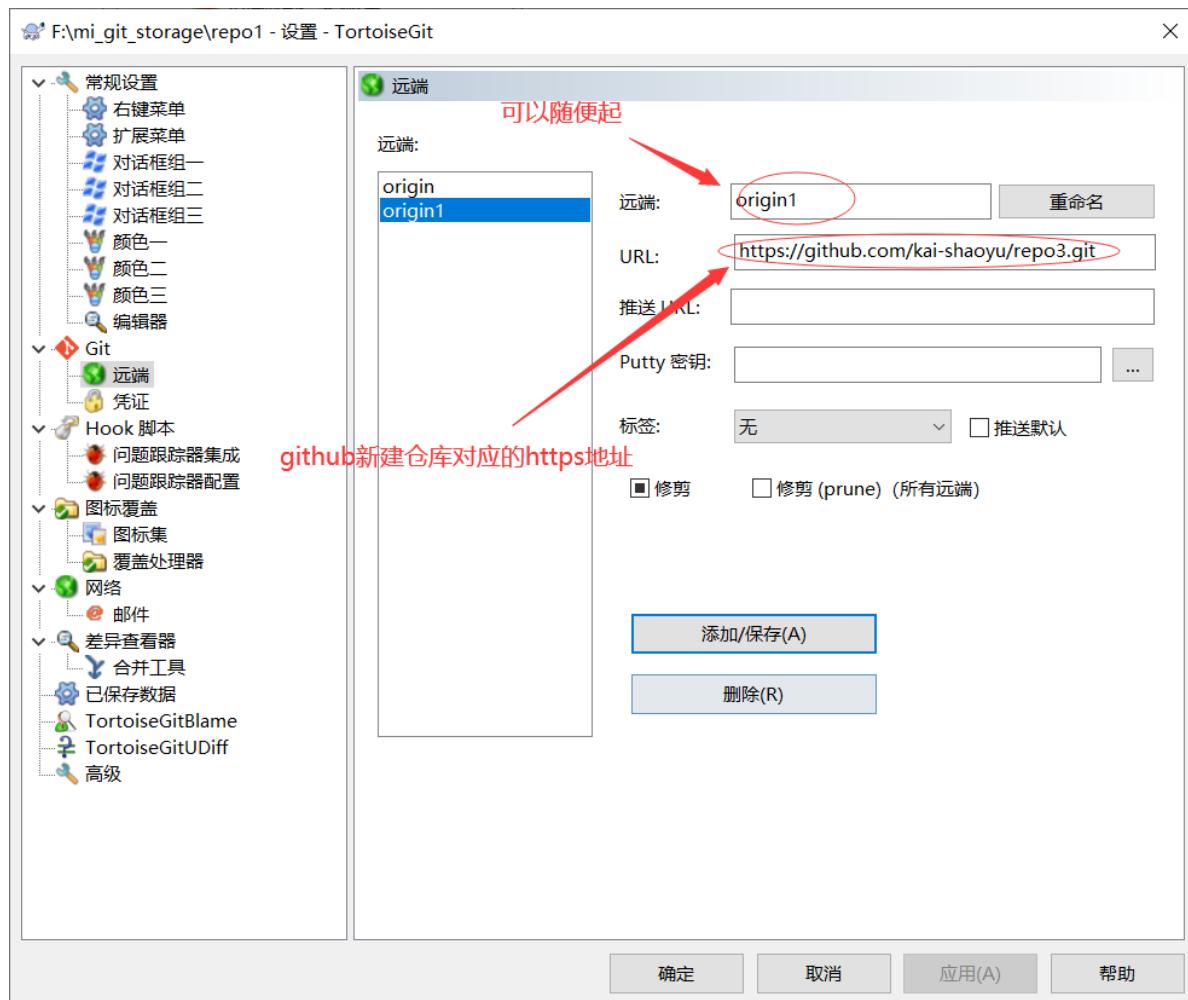


设置好上述之后，即可在git同步框进行推送，待推送完毕，刷新github页面即可显示。

## 2.https方式进行同步（不推荐）

需要用户的用户名和密码，而不是密钥

先进行远端设置



确定后，进入到git同步框，点击推送，填写用户名，密码，等到推送完毕，刷新github页面即可。

## 克隆到本地仓库

### 1.ssh方式

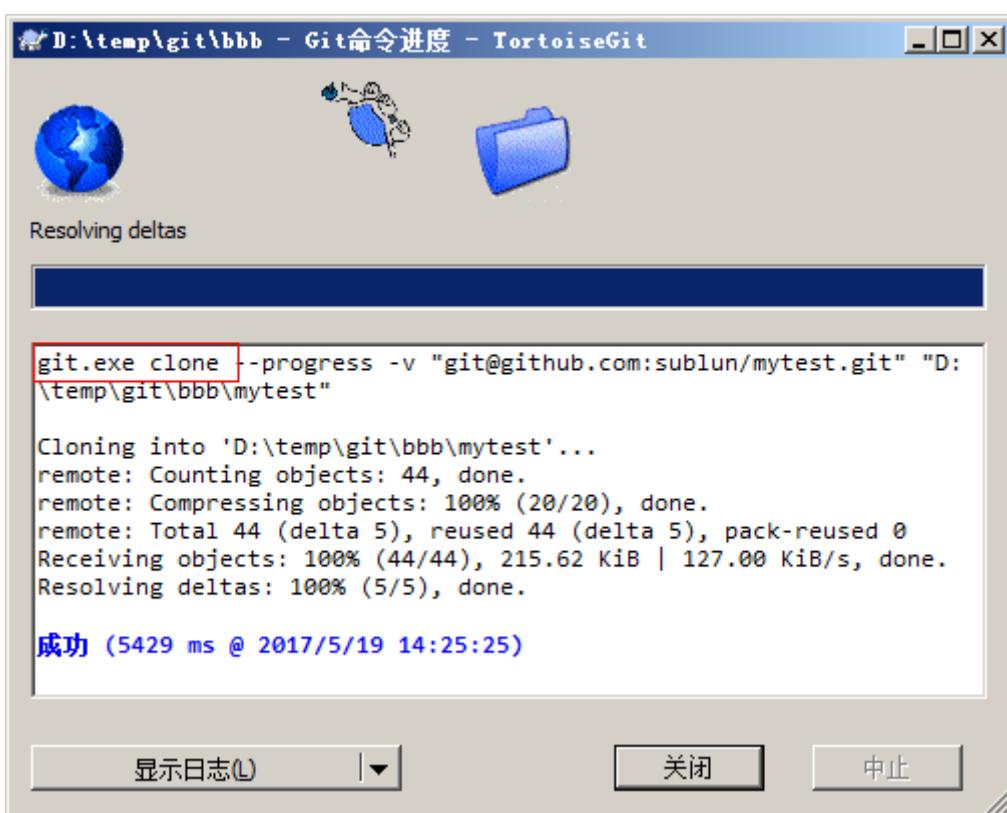
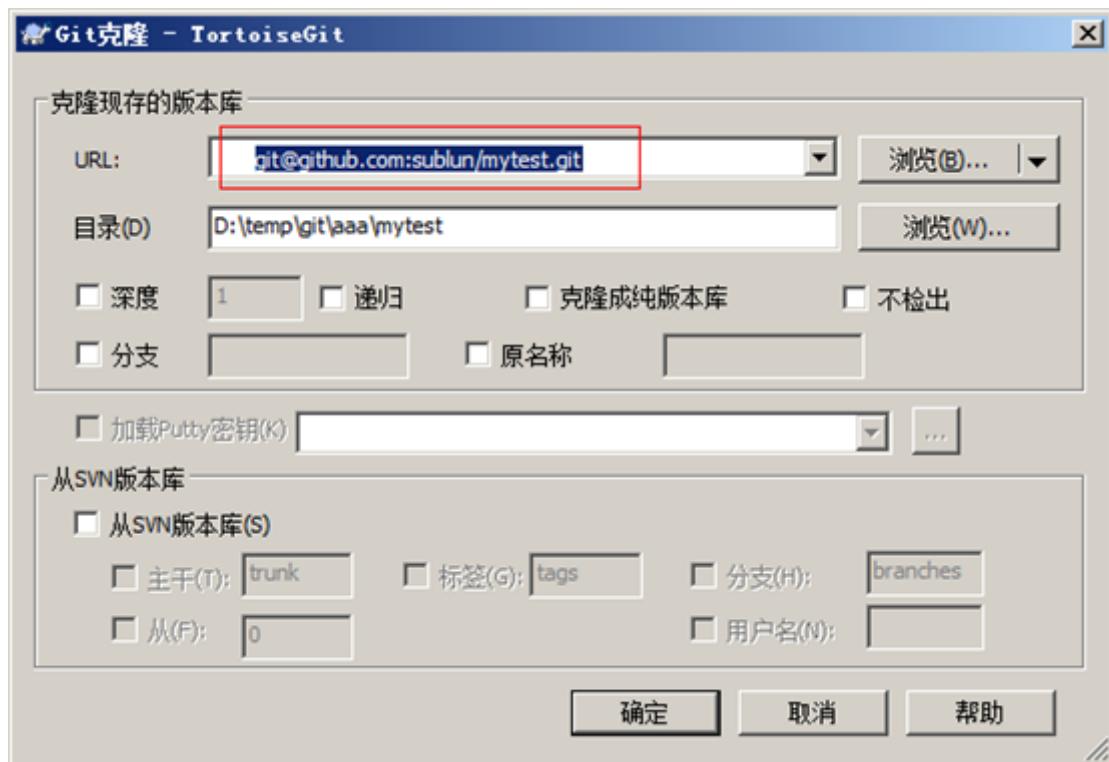
<1>git bash

```
$ git clone git@github.com:kai-shaoyu/repo3.git
```

这个命令，在github对应仓库下载按钮，选择ssh即可获取成功

<2>小乌龟

在任意目录点击右键：选择git 克隆



## 2.https

<1>git bash

```
$ git clone https://github.com/kai-shaoyu/repo3.git
```

<2>小乌龟

https与ssh克隆区别就是，在git克隆框内，URL地址的不同。

## 推送修改的文件及冲突解决

两个仓库（本地仓和github仓）都进行了文件修改，而没有及时的更新。

不是在最新的文件上进行的修改，报错

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。

**例如：**

1. 在本地仓中对应文件夹的hello.txt进行编辑

然后提交到本地仓库，再推送到github中。

2. 在本地克隆文件夹中拉取的文件中进行修改hello.txt

提交到本地仓库，再推送到github中。

此时相当于，本地仓原文件所做的修改已经成功提交到github，与此同时我们也将科龙和文件夹中对应文件修改了，当我们把克隆文件推送到github时候就会产生冲突。

**解决办法：**

我们拉取最新的对应文件，也会提示未能顺利结束。但是无伤大雅，点击关闭。

冲突文件有个黄色叹号，我们点开可以看到



我们需要手动去解决这个冲突。

改完之后，右击小乌龟，点击解决冲突，确定

接着，提交到本地仓库，再推送到github

## 搭建私有git服务器

### 服务器搭建

远程仓库实际上和本地仓库没啥不同，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。

搭建Git服务器需要准备一台运行Linux的机器，在此我们使用CentOS。以下为安装步骤：

## **1、安装git服务环境准备**

```
yum -y install curl curl-devel zlib-devel openssl-devel perl cpio expat-devel gettext-devel gcc cc
```

## **2、下载git-2.5.0.tar.gz**

- 1) 解压缩 tar zxf 文件名
- 2) cd git-2.5.0
- 3) autoconf
- 4) ./configure 执行完之后生成一个make file
- 5) make
- 6) make install
- 7) git --version 查看git版本

## **3、添加用户**

```
adduser -r -c 'git version control' -d /home/git -m git
```

此命令执行后会创建/home/git目录作为git用户的主目录。

## **5、设置密码**

```
passwd git
```

输入两次密码

## **6、切换到git用户**

```
su git
```

```
whoami 查看当前是在什么用户下
```

## **7、创建git仓库**

```
git --bare init /home/git/first
```

注意：如果不使用“--bare”参数，初始化仓库后，提交master分支时报错。这是由于git默认拒绝了push操作，需要.git/config添加如下代码：

```
[receive]
```

```
denyCurrentBranch = ignore
```

推荐使用：git --bare init初始化仓库。

```
git
[git@localhost ~]$ cd ~
[git@localhost ~]$ pwd
/home/git
[git@localhost ~]$ ll
total 0
[git@localhost ~]$ mkdir repo1
[git@localhost ~]$ ll
total 4
drwxrwxr-x. 2 git git 4096 May 15 01:27 repo1
[git@localhost ~]$ cd repo1/
[git@localhost repo1]$ git init --bare
Initialized empty Git repository in /home/git/repo1/
[git@localhost repo1]$ ll
total 32
drwxrwxr-x. 2 git git 4096 May 15 01:28 branches
-rw-rw-r--. 1 git git 66 May 15 01:28 config
-rw-rw-r--. 1 git git 73 May 15 01:28 description
-rw-rw-r--. 1 git git 23 May 15 01:28 HEAD
drwxrwxr-x. 2 git git 4096 May 15 01:28 hooks
drwxrwxr-x. 2 git git 4096 May 15 01:28 info
drwxrwxr-x. 4 git git 4096 May 15 01:28 objects
drwxrwxr-x. 4 git git 4096 May 15 01:28 refs
[git@localhost repo1]$
```

## 连接服务器

私有git服务器搭建完成后就可以向连接github一样连接使用了，但是我们的git服务器并没有配置密钥登录，所以每次连接时需要输入密码。

### 使用命令连接：

```
$ git remote add origin ssh://git@192.168.25.156/home/git/first
```

这种形式和刚才使用的形式好像不一样，前面有ssh://前缀，好吧你也可以这样写：

```
$ git remote add origin git@192.168.25.156:first
```

### 使用TortoiseGit同步

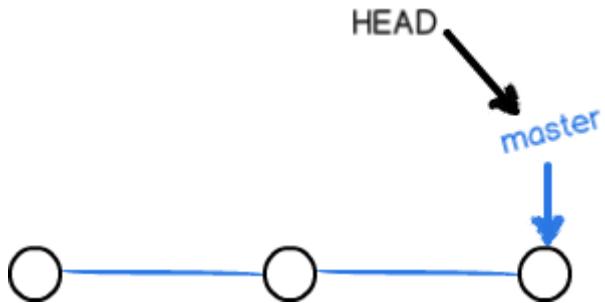
参考上面的使用方法。雷同。

## 分支管理（高级用法）

### 创建合并分支

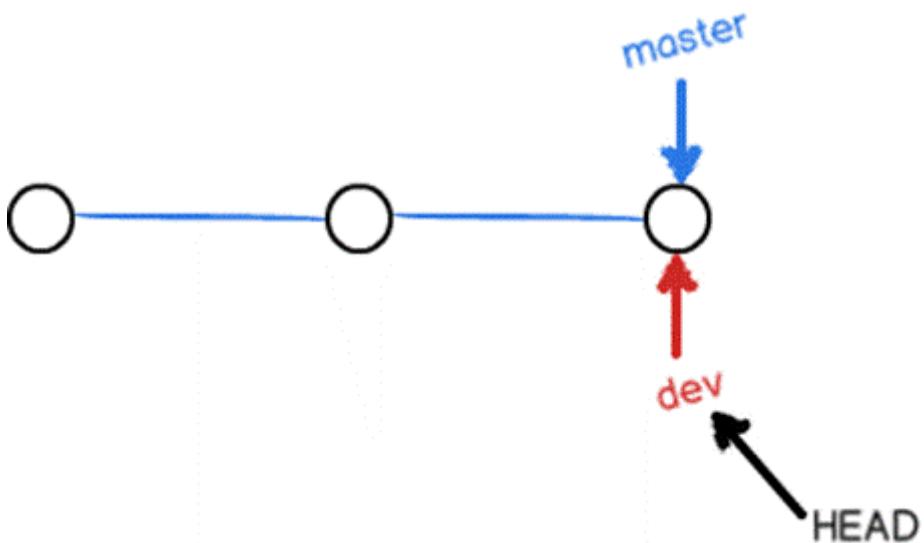
在我们每次的提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD指针严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。

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：



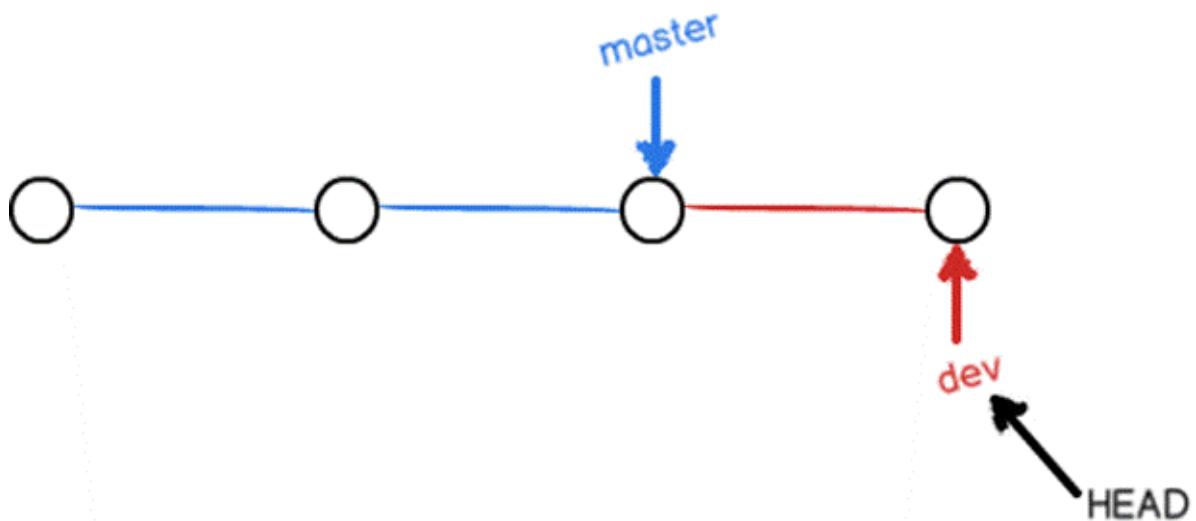
每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长。

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：

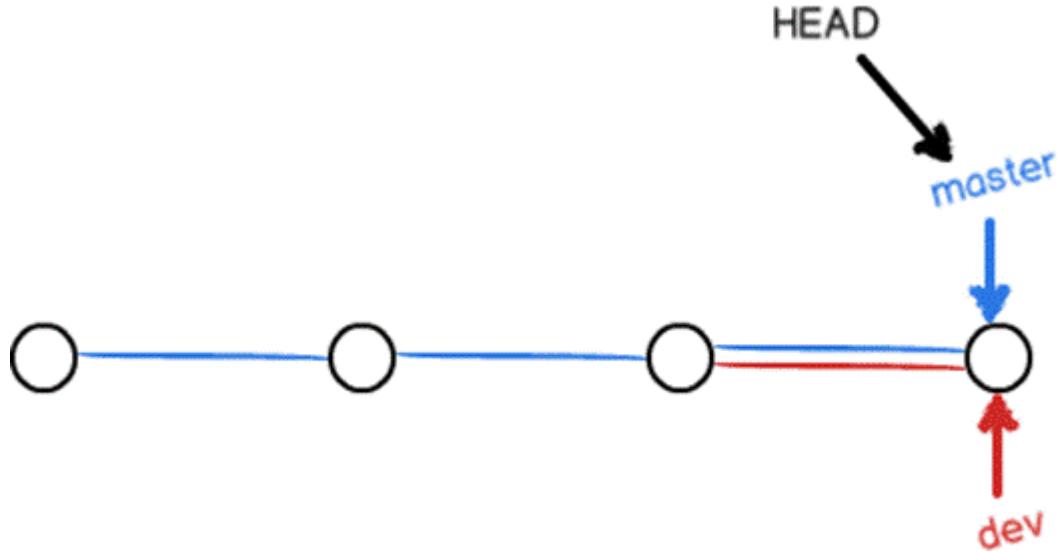


你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！

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：

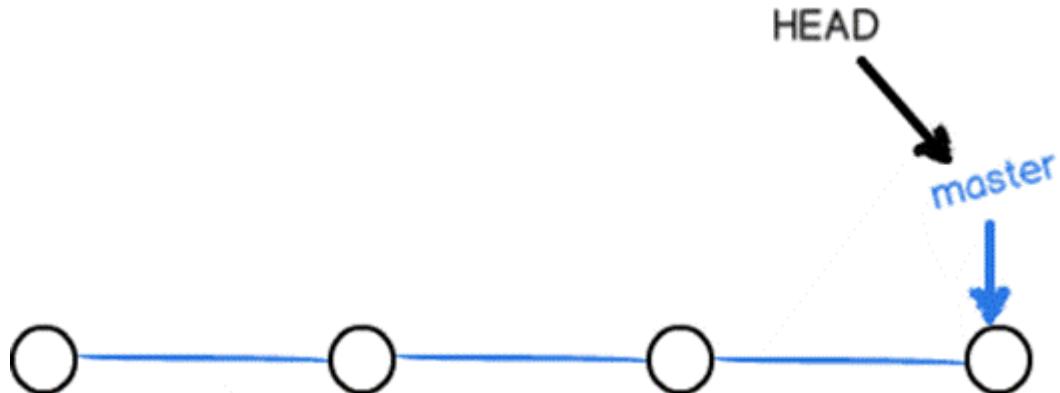


假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！

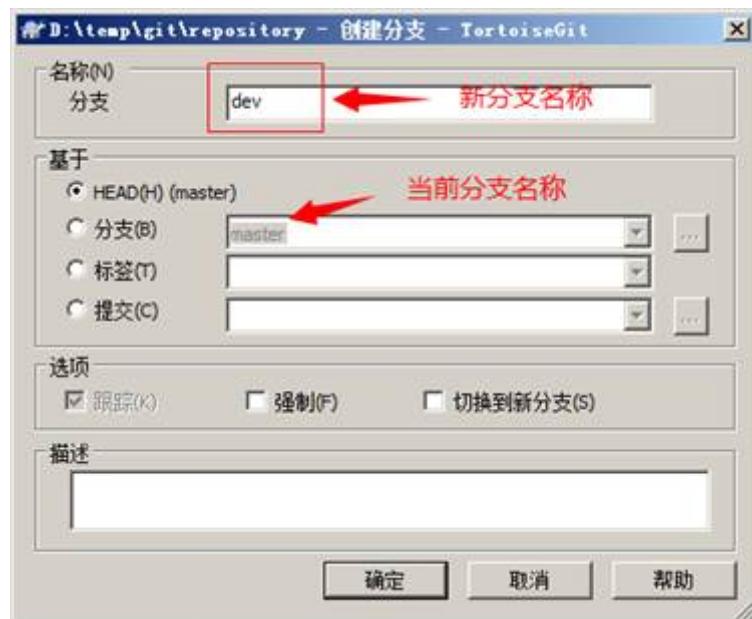
合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



## 使用TortoiseGit实现分支管理

### 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

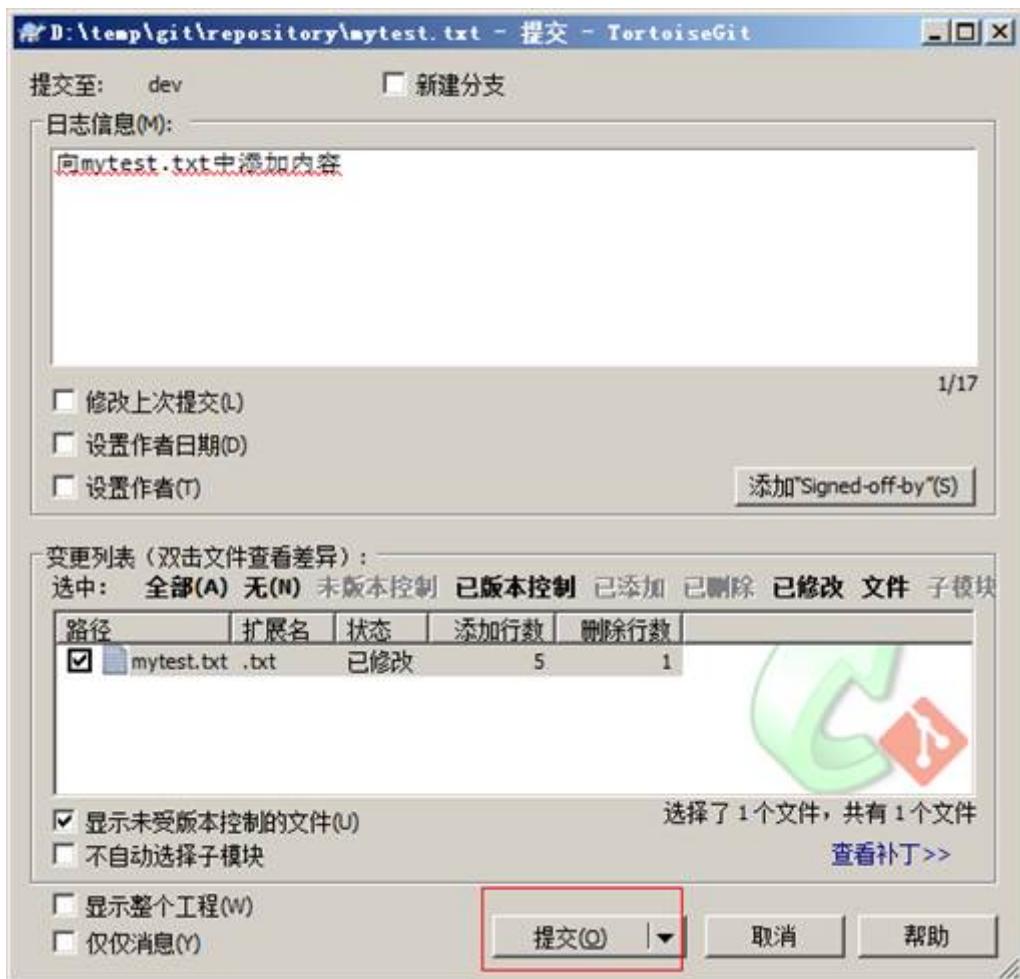
分支切换到dev后就可以对工作区的文件进行修改，然后提交到dev分支，原来的master分支不受影响。例如我们修改mytest.txt中的内容，然后提交到dev分支。

The screenshot shows a Windows file explorer window displaying a Git repository structure. The contents of the repository are:

名称	修改日期	类型	大小
.git	2017/5/19 17:14	文件夹	
Project02	2017/5/19 9:06	文件夹	
project-test	2017/5/19 9:09	文件夹	
mytest.txt	2017/5/19 17:15	文本文档	1 KB

Below the file explorer is a screenshot of a Notepad window titled 'mytest.txt - 记事本'. The content of the file is:

```
Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。  
-----  
这是dev分支新添加的内容
```



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”：

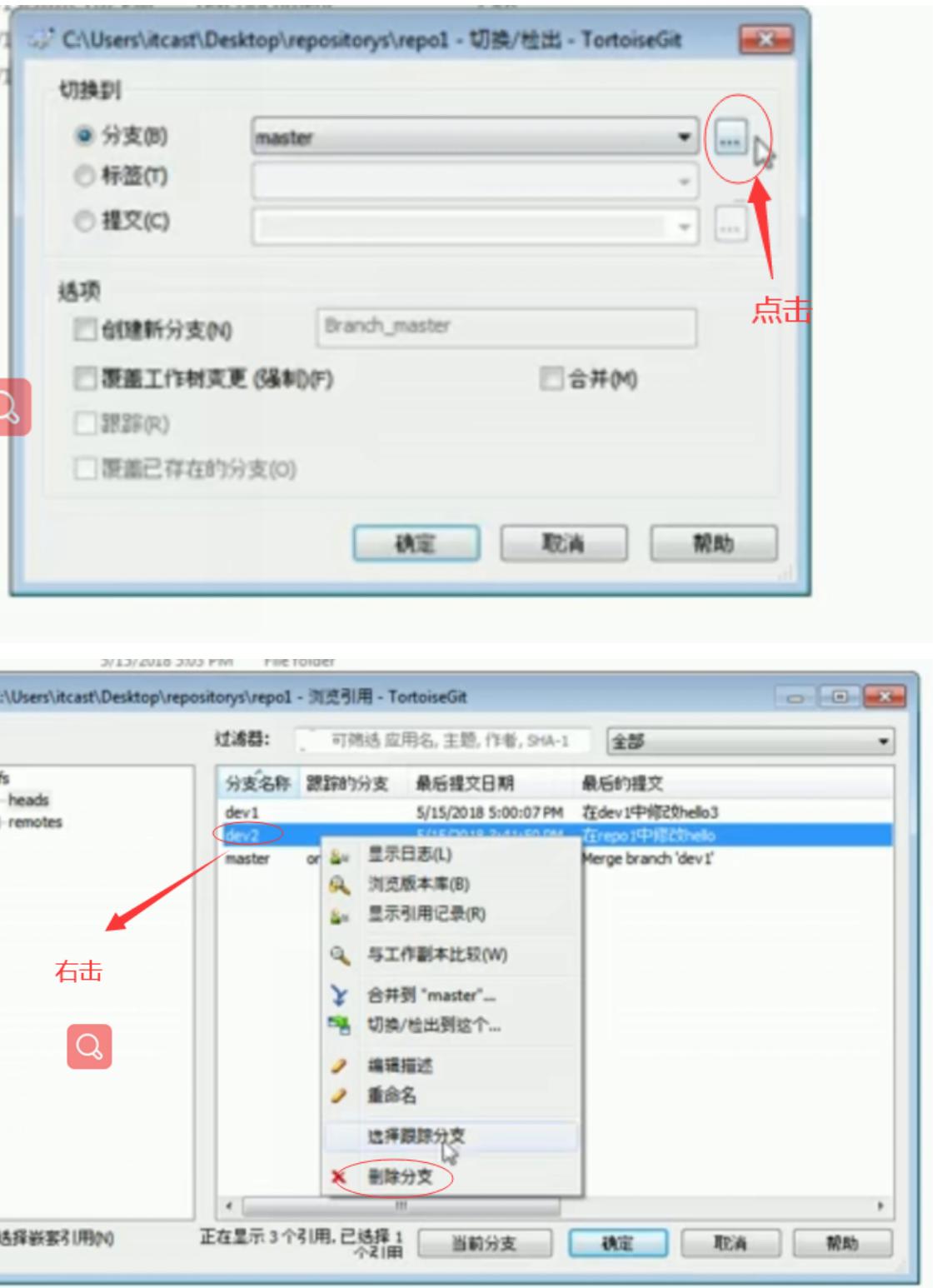


再查看mytest.txt的内容就已经更新了：



## 删除分支

右键菜单中选择“合并”：



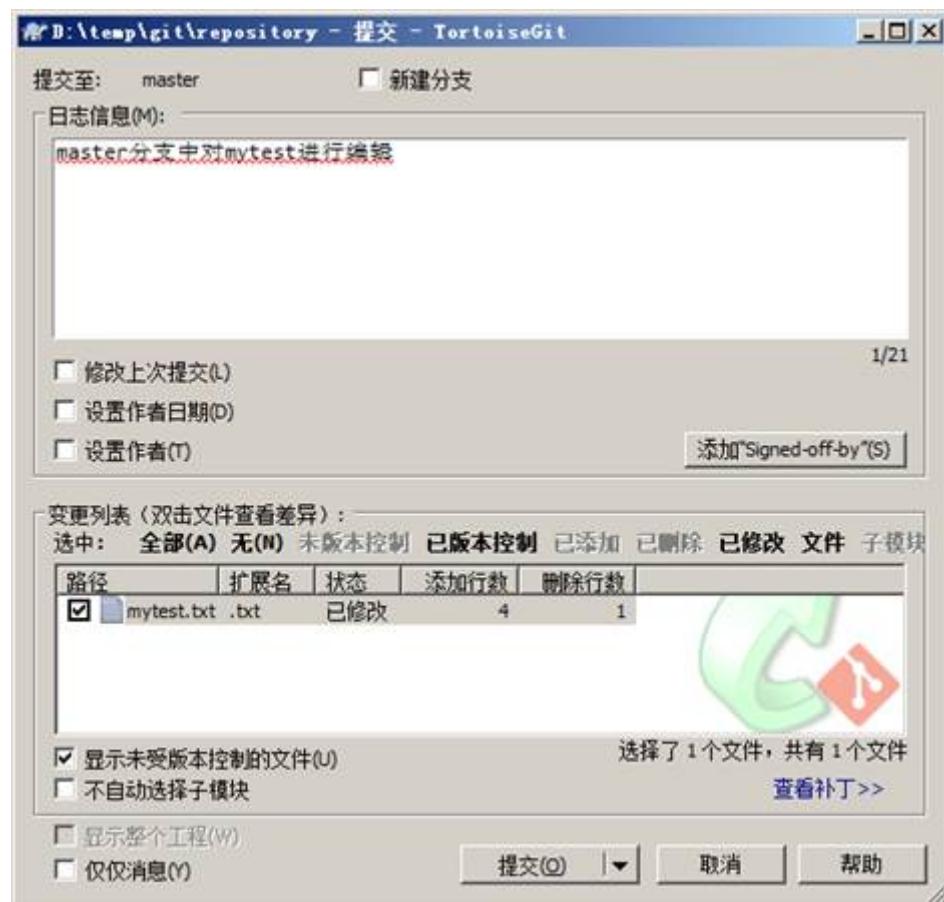
## 解决冲突

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。

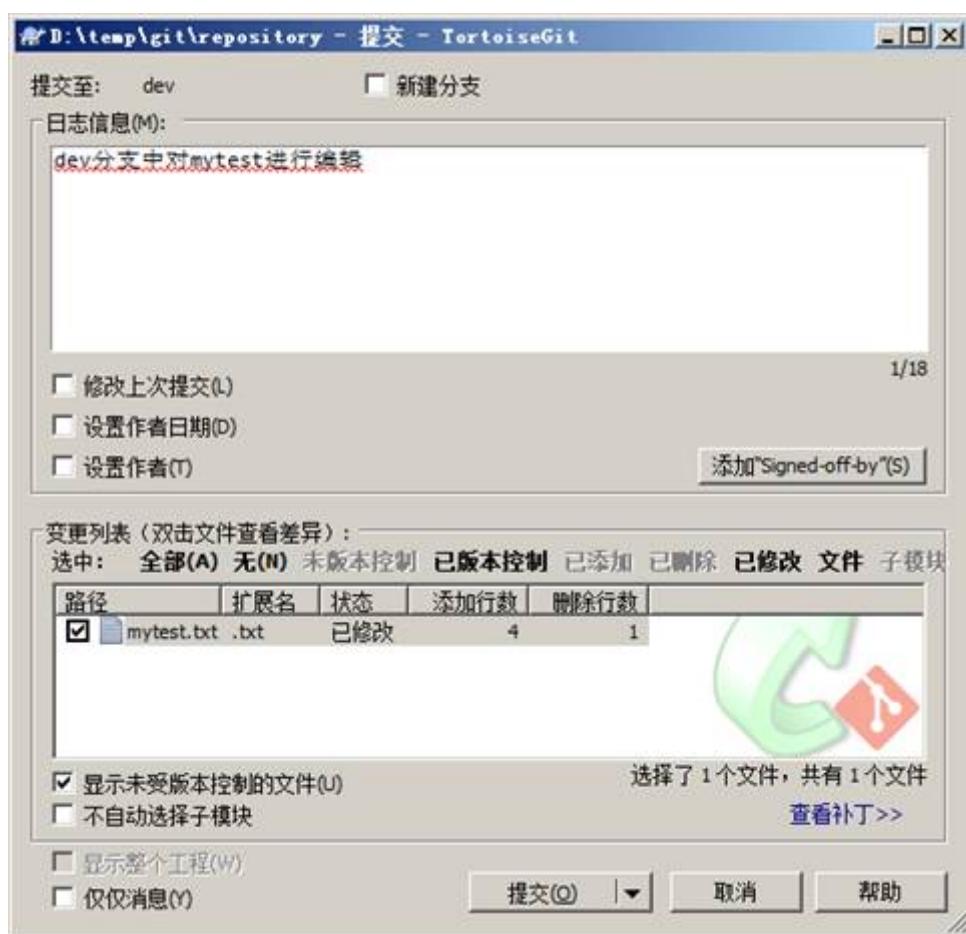
例如在master分支中对mytest.txt进行编辑：



然后提交到版本库。



切换到dev分支，对mytest.txt进行编辑：



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。

D:\temp\git\repository - Git命令进度 - TortoiseGit

```
git.exe merge dev

Auto-merging mytest.txt
CONFLICT (content): Merge conflict in mytest.txt
Automatic merge failed; fix conflicts and then commit the result.

git 未能顺利结束 (退出码 1) (218 ms @ 2017/5/19 17:35:01)
```

解决 | 关闭 | 中止

出现版本冲突。

本地磁盘 (D:) \ temp \ git \ repository

冲突的文件上有一个“!”

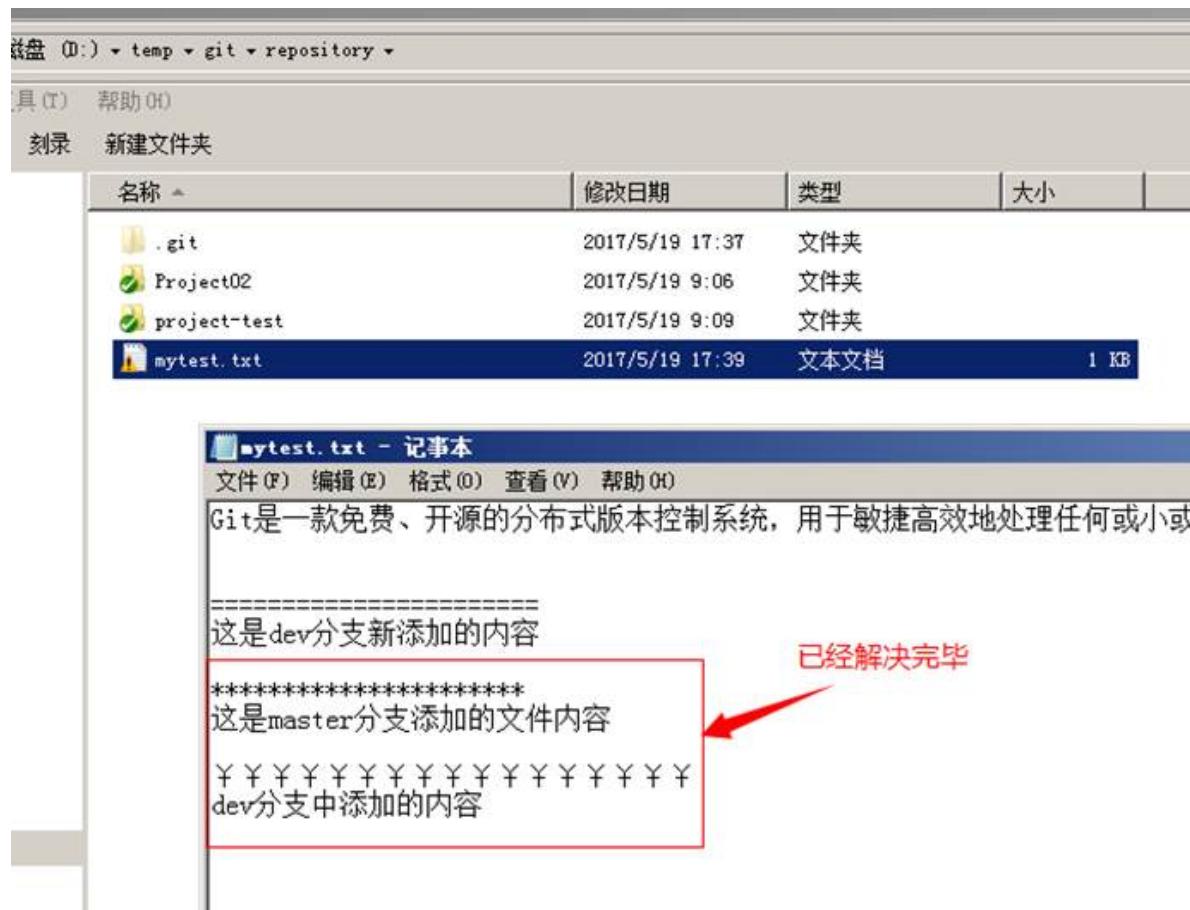
名称	修改日期	类型	大小
.git	2017/5/19 17:37	文件夹	
Project02	2017/5/19 9:06	文件夹	
project-test	2017/5/19 9:09	文件夹	
mytest.txt	2017/5/19 17:35	文本文档	1 KB

mytest.txt - 记事本

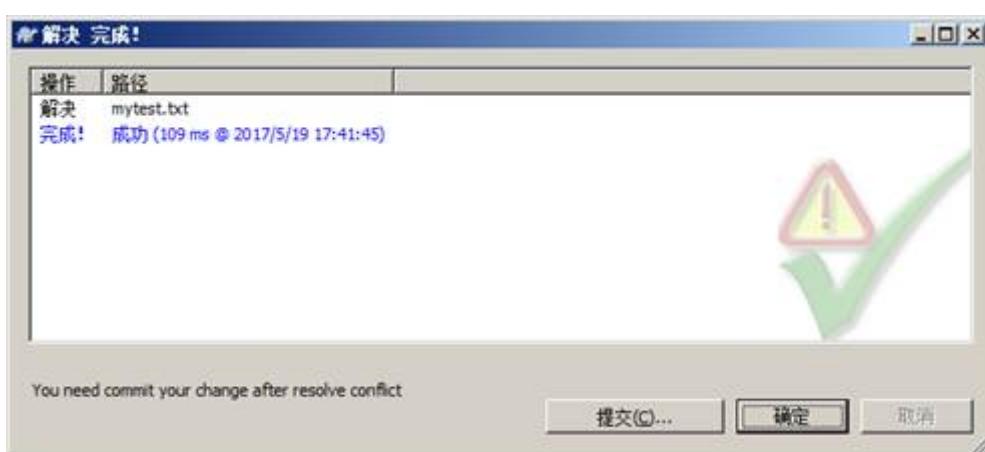
```
Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何大小的项目。

=====
这是dev分支新添加的内容
<<<<< HEAD
*****这是master分支添加的文件内容*****
=====
$ $ $ $ $ $ $ $ $ $ $ $ $ $ $ $ 
dev分支中添加的内容
>>>> dev
```

冲突需要手动解决。



在冲突文件上单机右键选择“解决冲突”菜单项：



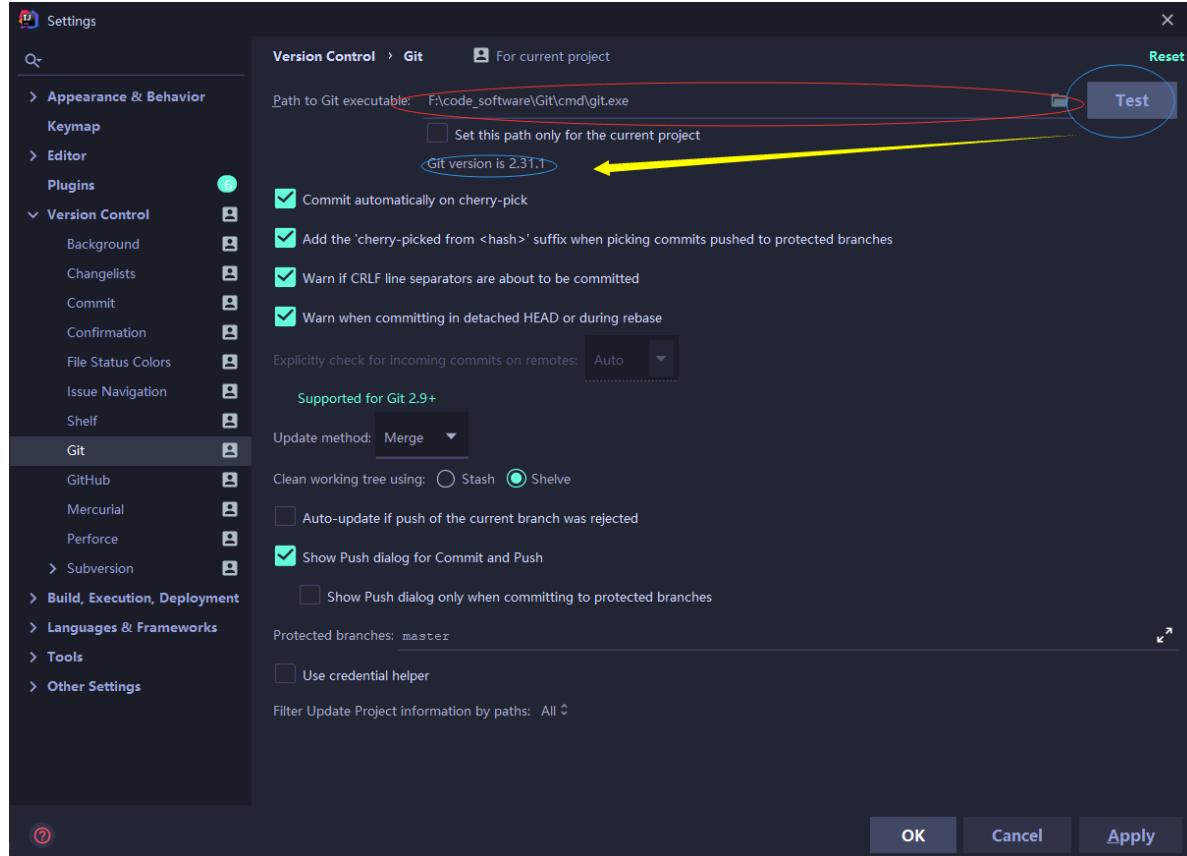
把冲突解决完毕的文件提交到版本库就可以了。

# 在IntelliJ IDEA中使用git

## 在idea中配置git

安装好IntelliJ IDEA后，如果Git安装在默认路径下，那么idea会自动找到git的位置，如果更改了Git的安装位置则需要手动配置下Git的路径。

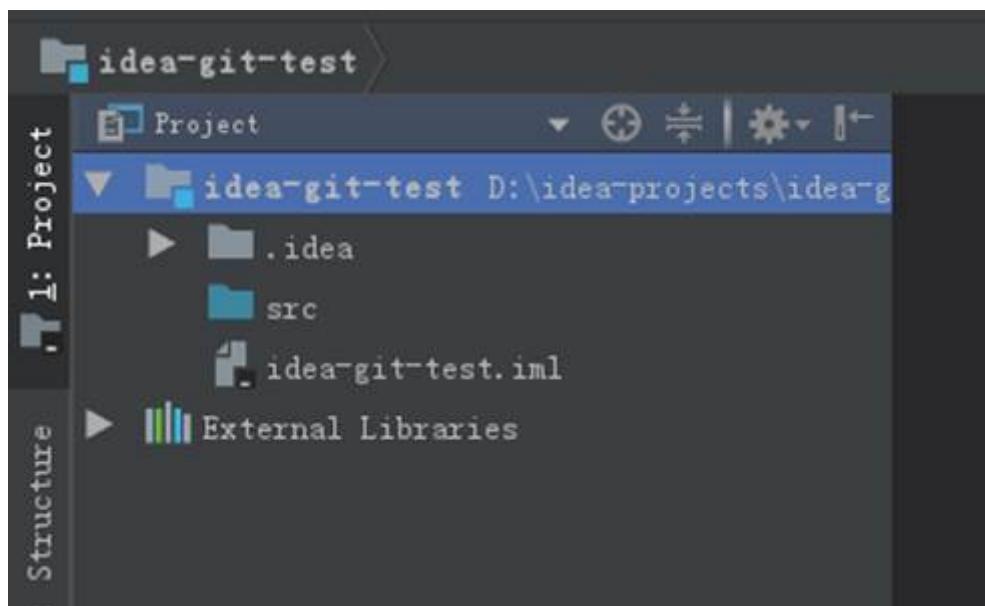
选择File→Settings打开设置窗口，找到Version Control下的git选项：



## 将工程添加到git

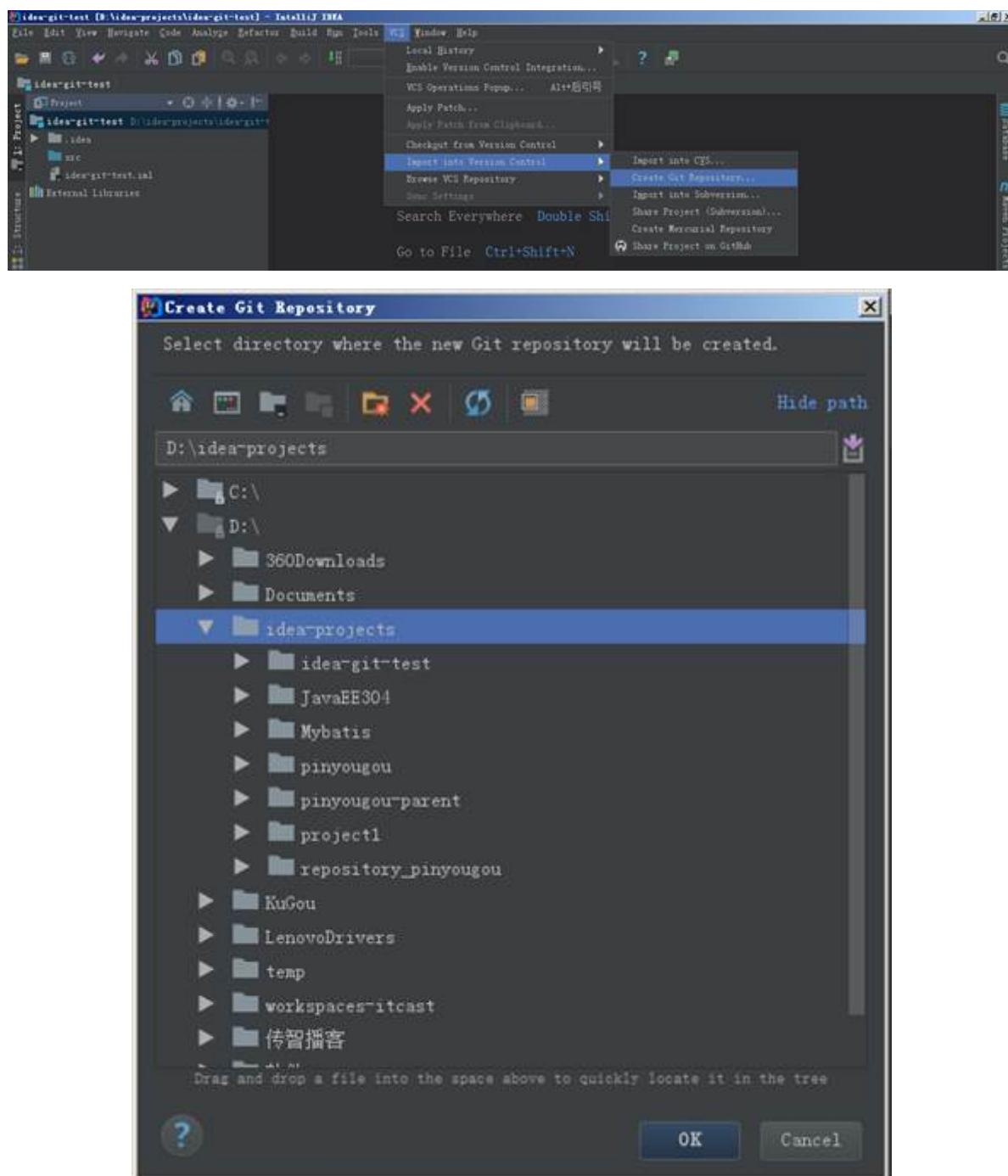
### 1) 在idea中创建一个工程

例如创建一个java工程，名称为idea-git-test，如下图所示：



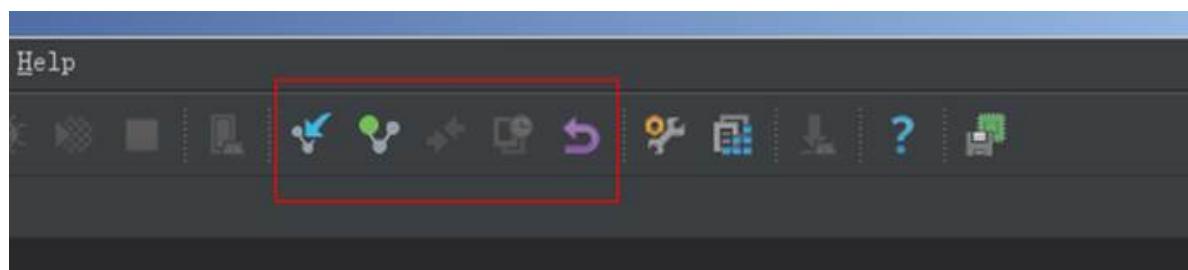
## 2) 创建本地仓库

在菜单中选择“vcs”→Import into Version Control→Create Git Repository...

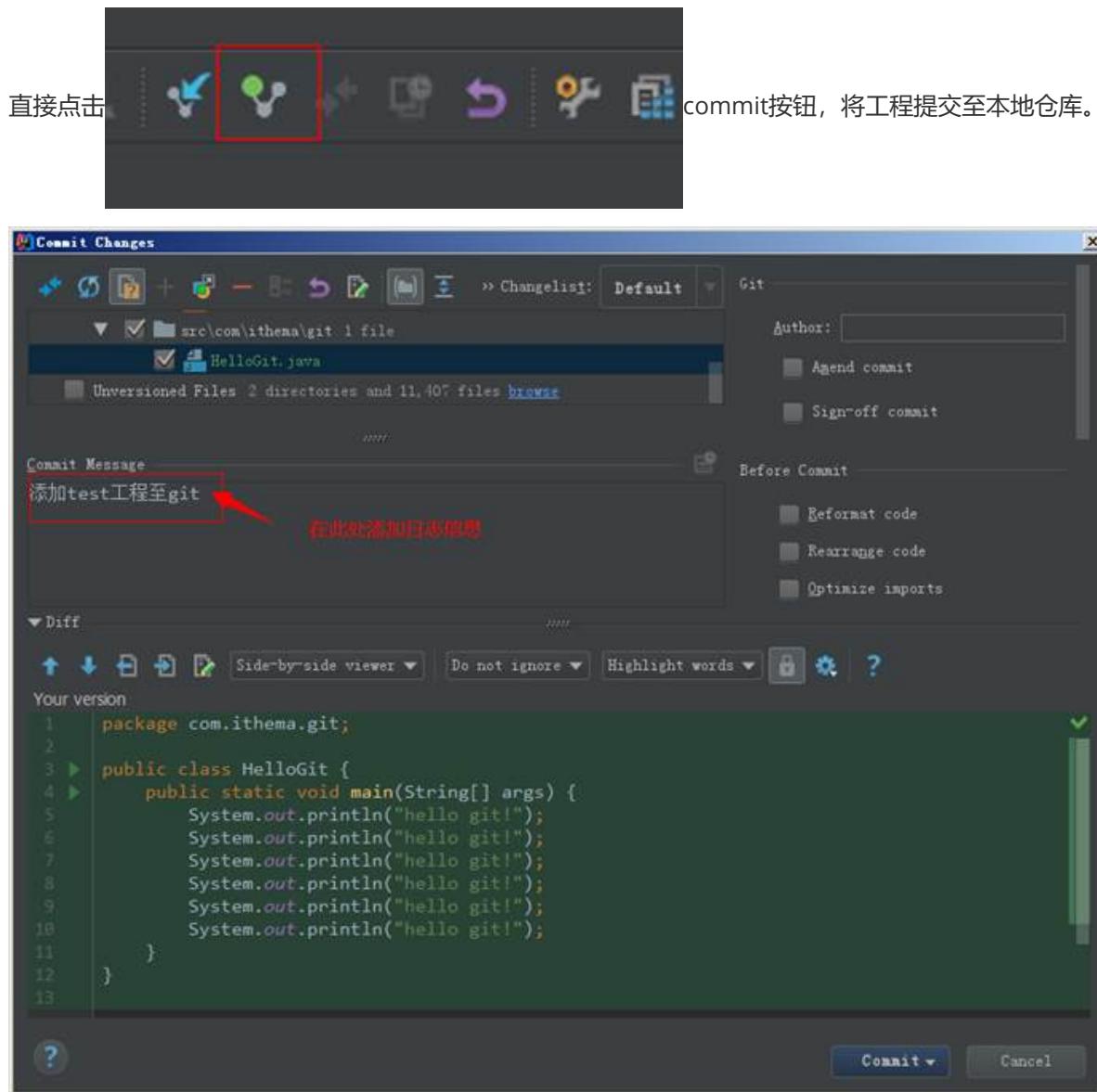


选择工程所在的上级目录。本例中应该选择idea-projects目录，然后点击“OK”按钮，在工程的上级目录创建本地仓库，那么idea-projects目录就是本地仓库的工作目录，此目录中的工程就可以添加到本地仓库中。也就是可以把idea-git-test工程添加到本地仓库中。

选择之后在工具栏上就多出了git相关工具按钮：



### 3) 将工程添加至本地仓库



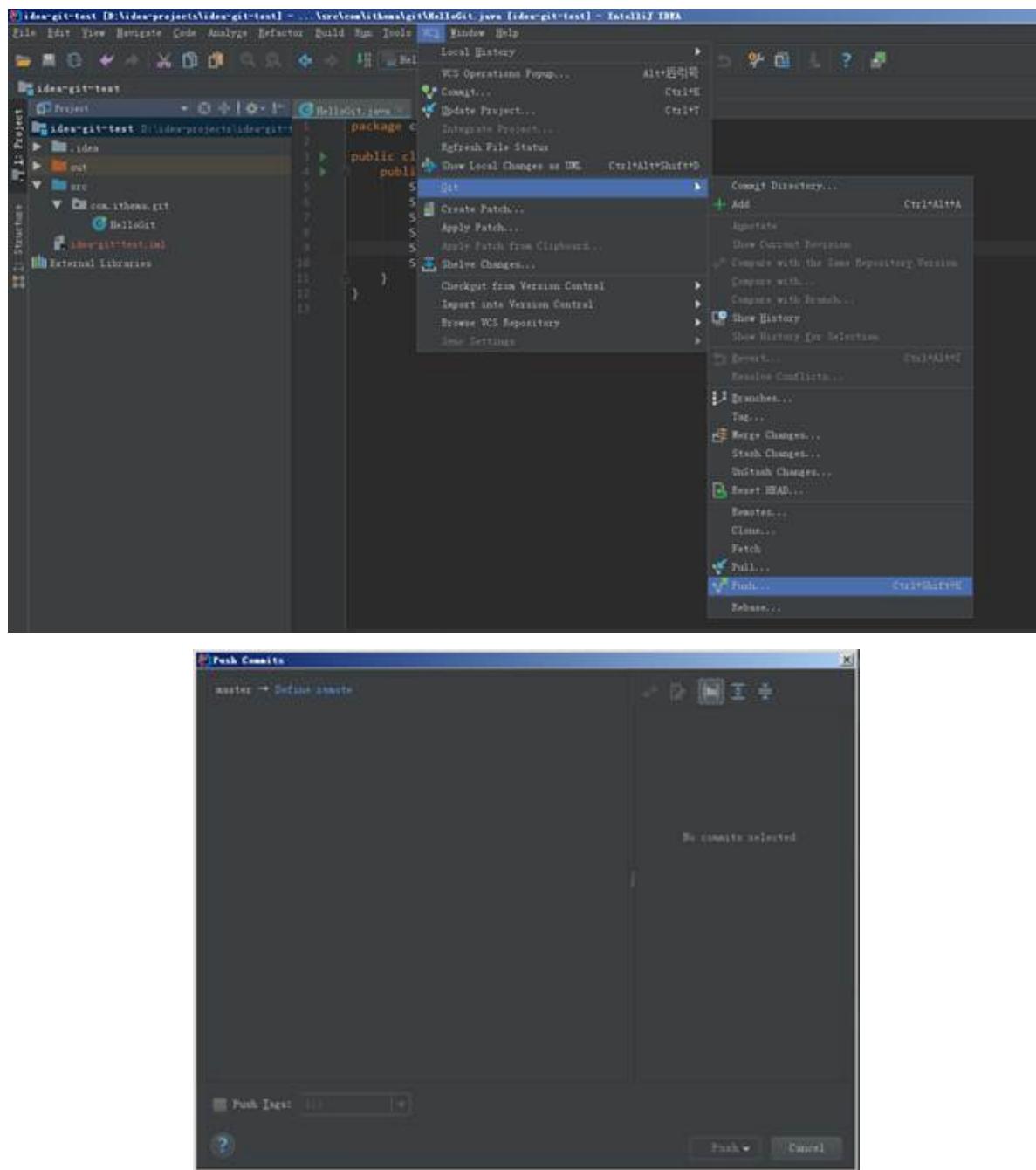
然后点击“commit”按钮，将工程添加至本地仓库。

### 4) 推送到远程

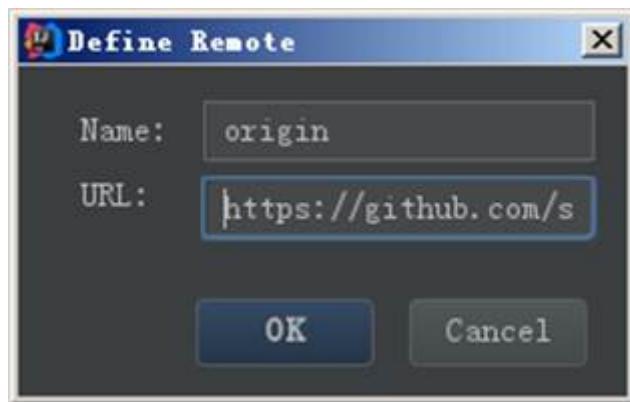
在github上创建一个仓库然后将本地仓库推送到远程。

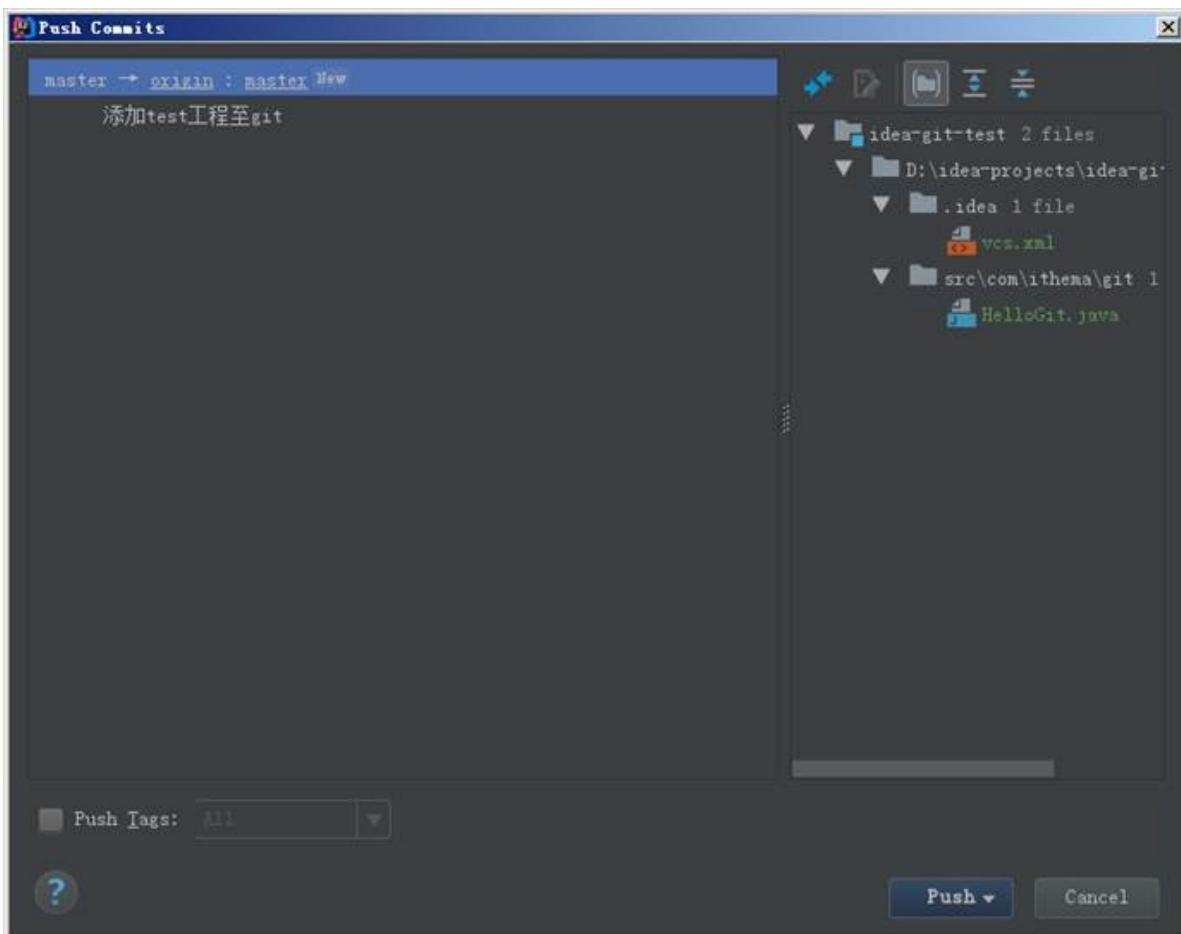
在工程上点击右键，选择git→Repository→push，

或者在菜单中选择vcs→git→push



点击“Define remote”链接，配置https形式的URL，SSH的无法通过。然后点击OK

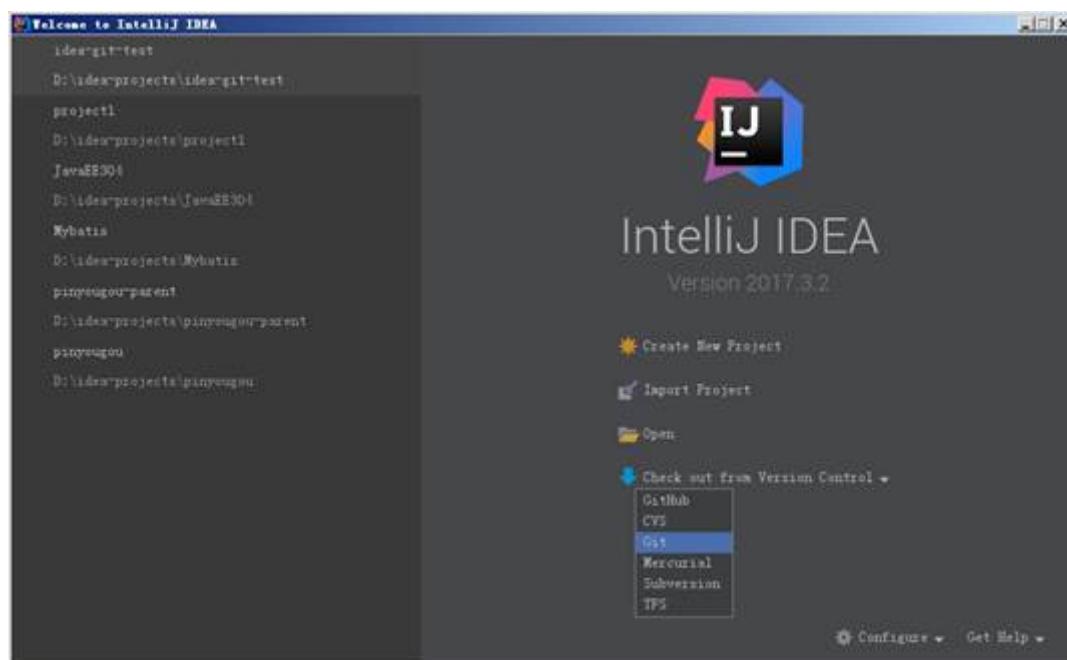


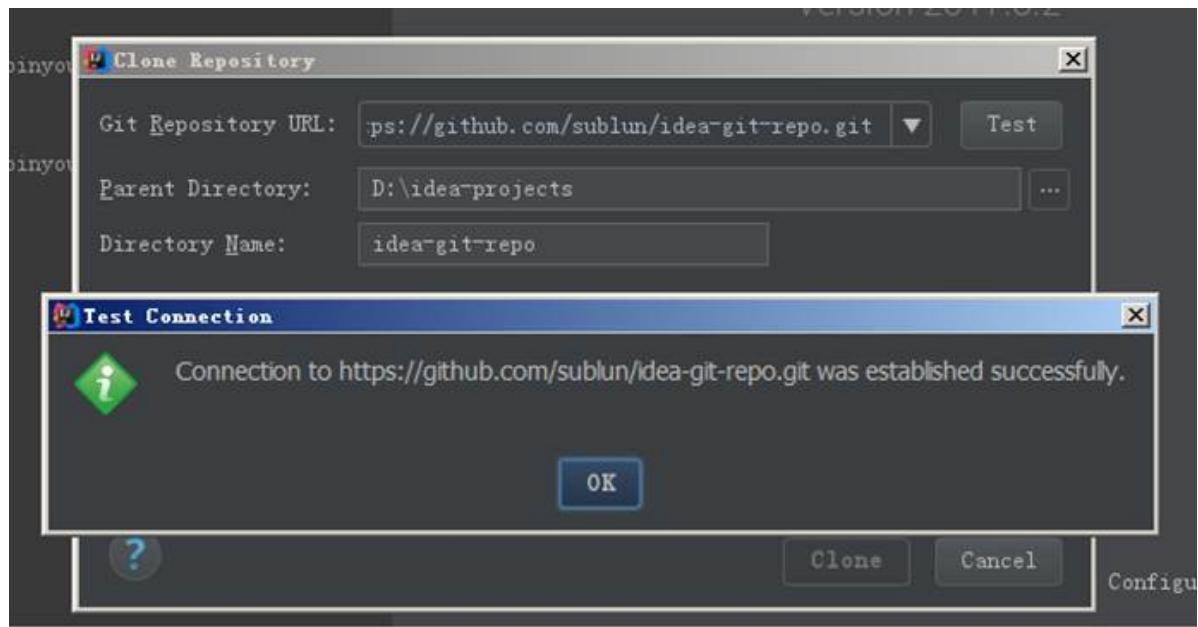


点击“push”按钮就将本地仓库推送到远程，如果是第一次配置推送需要输入github的用户名和密码。

## 从远程仓库克隆

关闭工程后，在idea的欢迎页上有“Check out from version control”下拉框，选择git



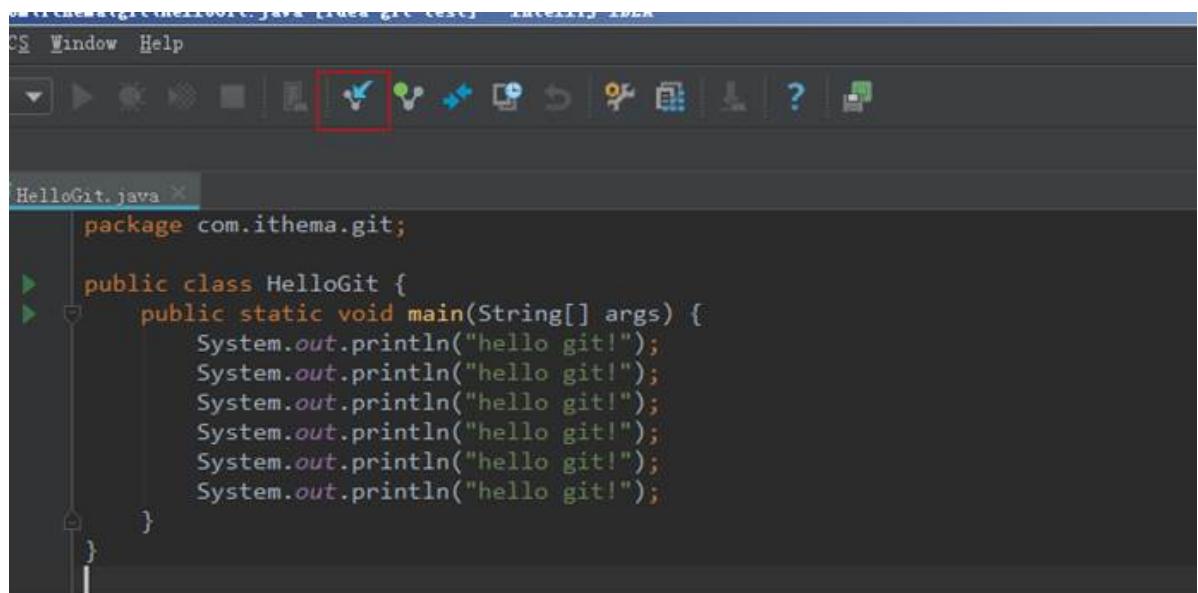


此处仍然推荐使用https形式的url，点击“test”按钮后显示连接成功。

点击OK按钮后根据提示将远程仓库克隆下来，然后倒入到idea中。

## 从服务端拉取代码

如果需要从服务端同步代码可以使用工具条中的“update”按钮



## 在idea中使用git分支

