

# Git

## 1. 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多的参与者。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）



1543301117357

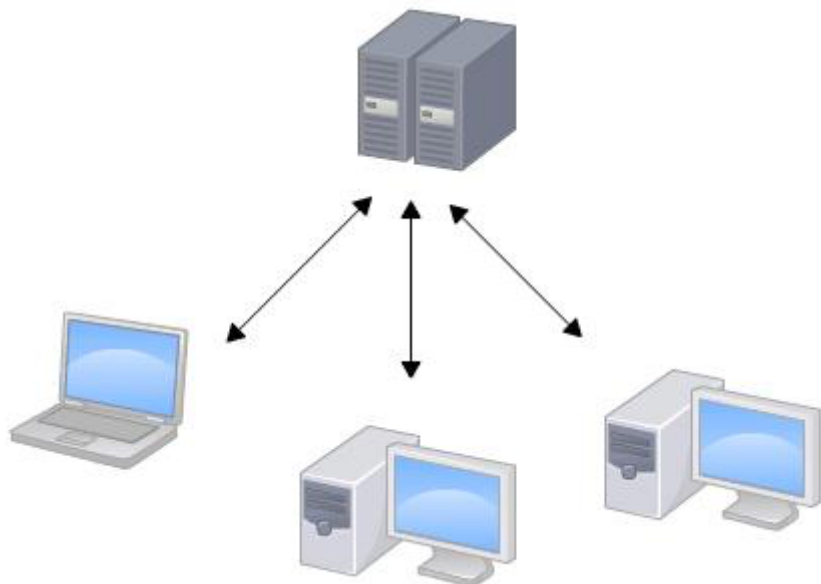
## 2. Git 与 SVN 对比

### 2.1 SVN概述

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。

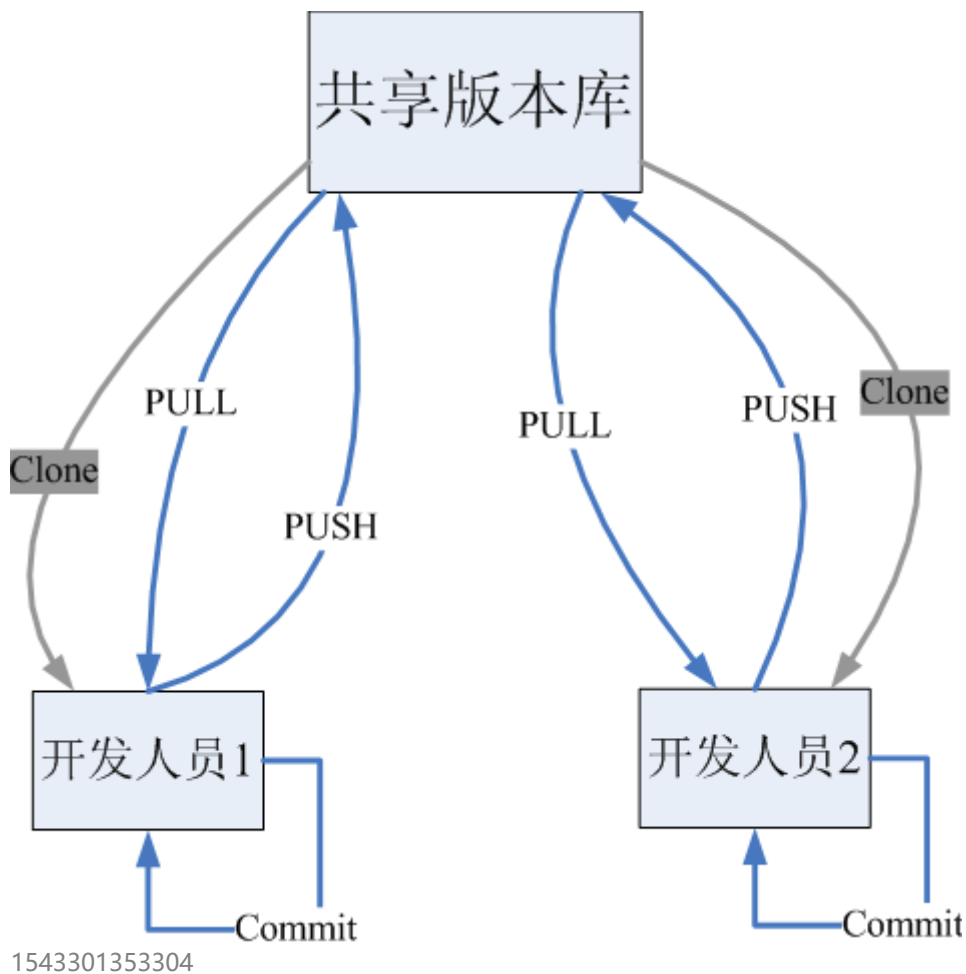
但是相较于其优点而言，集中式版本控制工具缺点很明显：

- 服务器单点故障
- 容错性差

## 2.2 Git的概述

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：



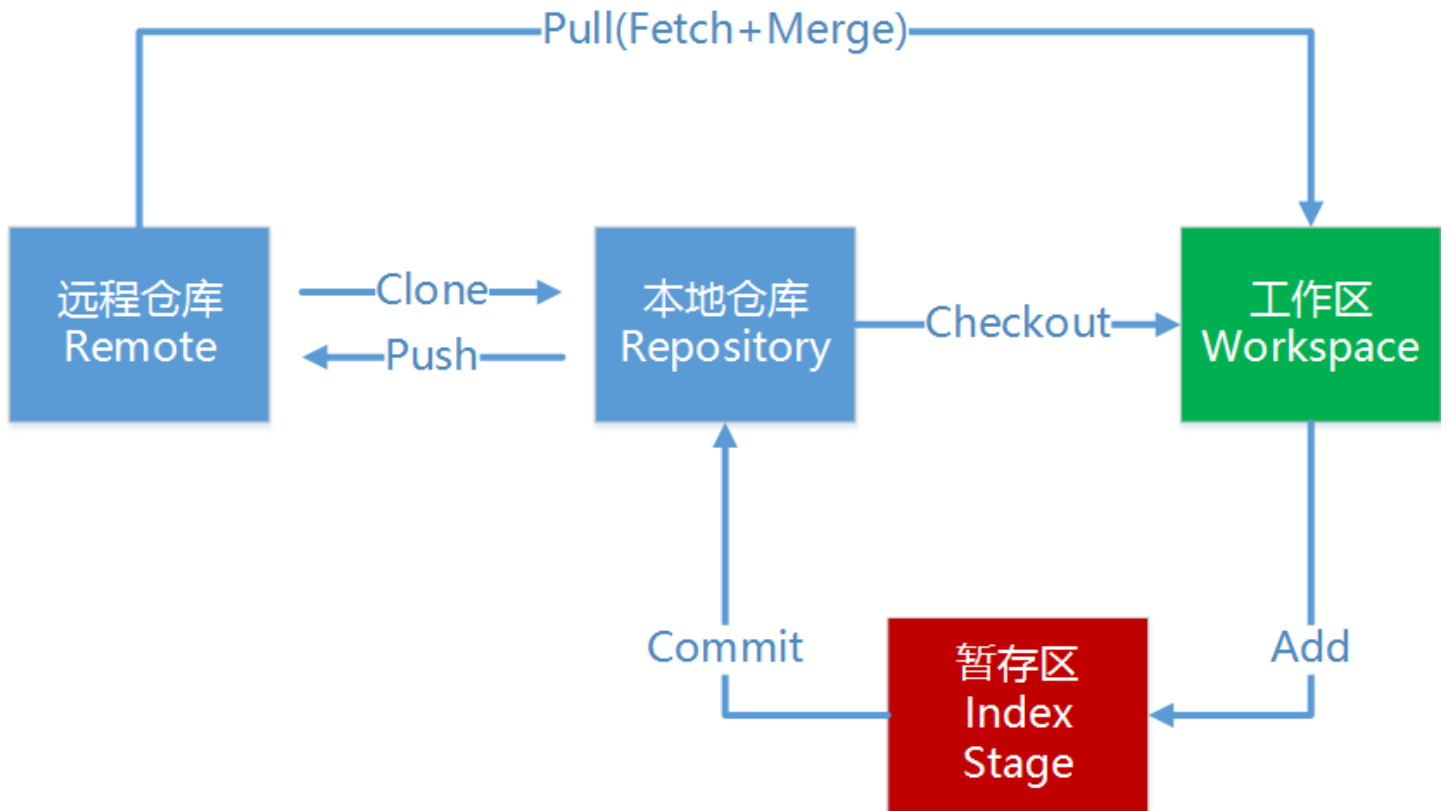
### 3. Git 工作流程

一般工作流程如下：

- 1) 从远程仓库中克隆 Git 资源作为本地仓库。
- 2) 从本地仓库中checkout代码然后进行代码修改
- 3) 在提交前先将代码提交到暂存区。
- 4) 提交修改。提交到本地仓库。本地仓库中保存修改的各个历史版本。
- 5) 在修改完成后，需要和团队成员共享代码时，可以将代码push到远程仓库。

下图展示了 Git 的工作流程：

# Git常用命令流程图



1543302039138

## 4. Git的安装

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，所以本教程只讲解windows下的git的安装及使用。

### 4.1 软件下载

下载地址：<https://git-scm.com/download>

# Downloads



Mac OS X



Windows



Linux/Unix

Older releases are available and the [Git source repository](#) is on GitHub.



1543302574360

此电脑 > 本地磁盘 (H:) > Git版本控制 > 01.参考资料 > 环境安装包 > 64位windows				
名称	修改日期	类型	大小	
Git-2.13.0-64-bit.exe	2018/11/25 0:02	应用程序	36,286 KB	
TortoiseGit-2.4.0.2-64bit.msi	2018/11/25 0:02	Windows Install...	19,520 KB	
TortoiseGit-LanguagePack-2.4.0.0-64...	2018/11/24 23:58	Windows Install...	2,996 KB	

1543302755251

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。

## 4.2 软件安装

### 4.2.1 安装 git for Windows

名称	修改日期	类型	大小
Git-2.13.0-64-bit.exe	2018/11/25 0:02	应用程序	36,286 KB
TortoiseGit-2.4.0.2-64bit.msi	2018/11/25 0:02	Windows Install...	19,520 KB
TortoiseGit-LanguagePack-2.4.0.0-64...	2018/11/24 23:58	Windows Install...	2,996 KB

1543303151256



1543303188322

- 一路下一步使用默认选项即可

## 4.2.2 安装TortoiseGit

<https://download.tortoisegit.org/tgit/>

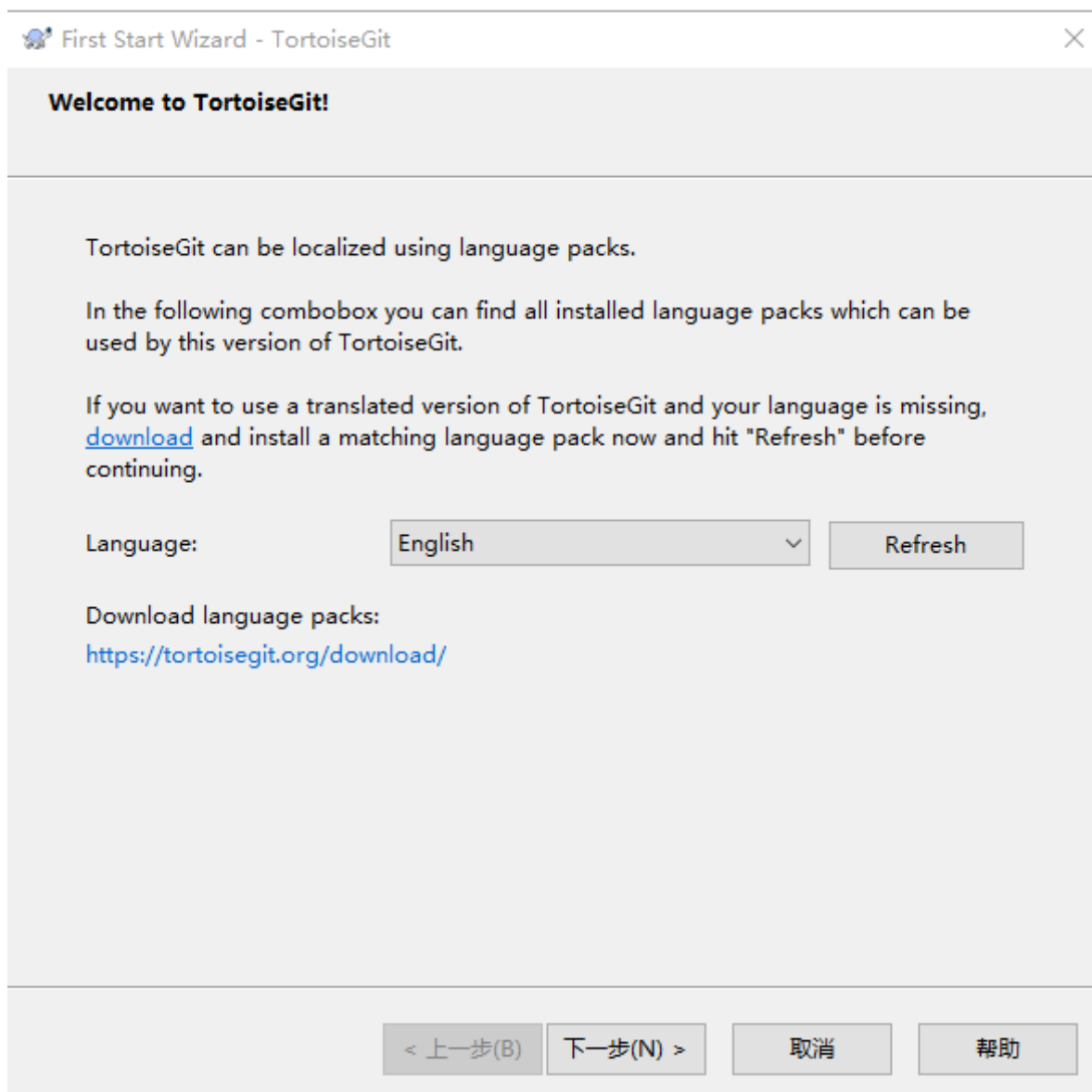
名称	修改日期	类型	大小
Git-2.13.0-64-bit.exe	2018/11/25 0:02	应用程序	36,286 KB
TortoiseGit-2.4.0.2-64bit.msi	2018/11/25 0:02	Windows Install...	19,520 KB
TortoiseGit-LanguagePack-2.4.0.0-64...	2018/11/24 23:58	Windows Install...	2,996 KB

1543309805552



1543309945750

- 一路下一步, 即可安装, 安装后会出现如下界面:



1543310005697

- 点击下一步:一直到如下这个界面
  - 如果在安装git时候,没有设置安装目录, 此处选择默认即可
- 点击下一步,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



First Start Wizard - TortoiseGit

×

### Configure user information

Git requires that you set up a user name and email address. Both are used as meta data for your commits (not for authentication).

Name:

用户名

Email:

邮箱

These settings will be stored to your global git configuration (%HOME%/.gitconfig) and will be used for all your git repositories as a default.

☐ Don't store these settings now.

< 上一步(B)

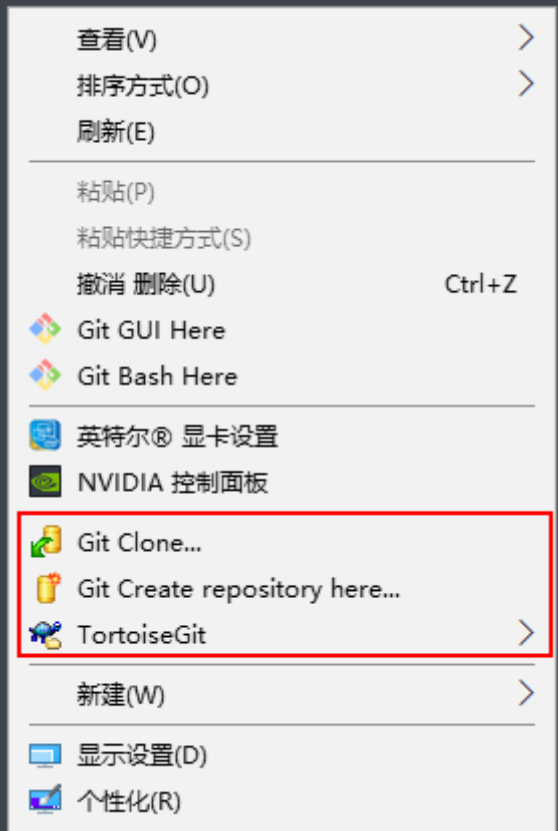
下一步(N) >

取消

帮助

1543310181198




- 完整安装后, 会添加右键快捷项,会出现如图的内容



1543310380905

### 4.2.3 安装中文语言包

说明中文语言包并不是必须选择，可以根据个人情况来选择安装

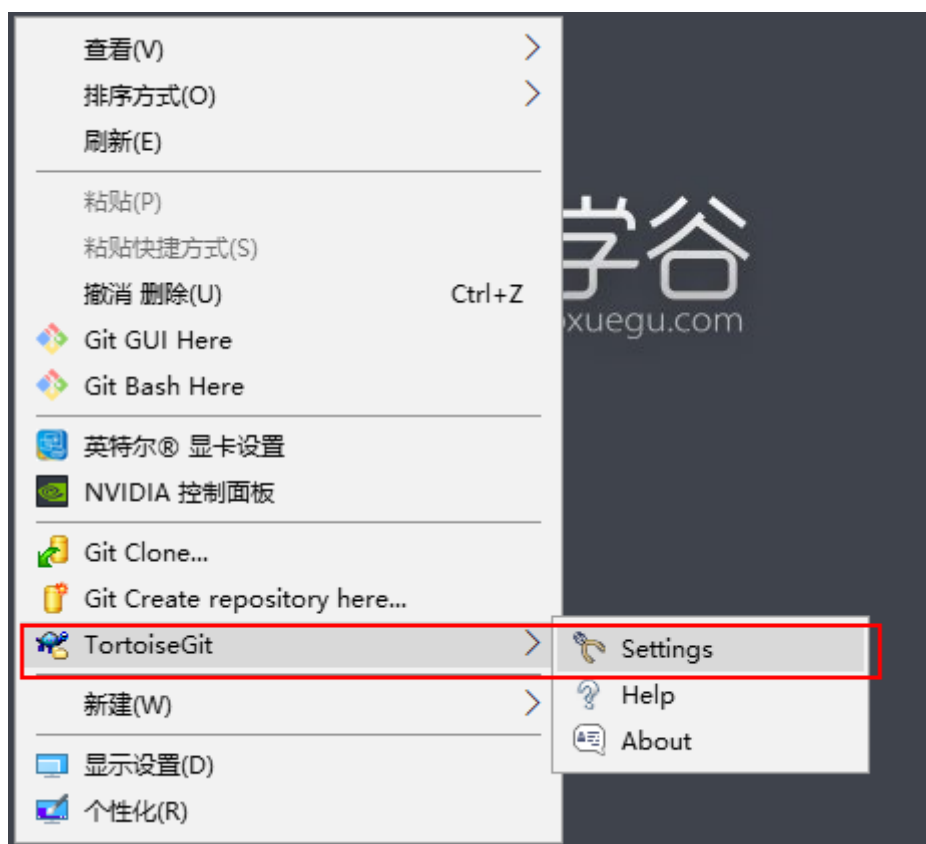
 Git-2.13.0-64-bit.exe	2018/11/25 0:02	应用程序	36,286 KB
 TortoiseGit-2.4.0.2-64bit.msi	2018/11/25 0:02	Windows Install...	19,520 KB
 TortoiseGit-LanguagePack-2.4.0.0-64...	2018/11/24 23:58	Windows Install...	2,996 KB

1543310476263

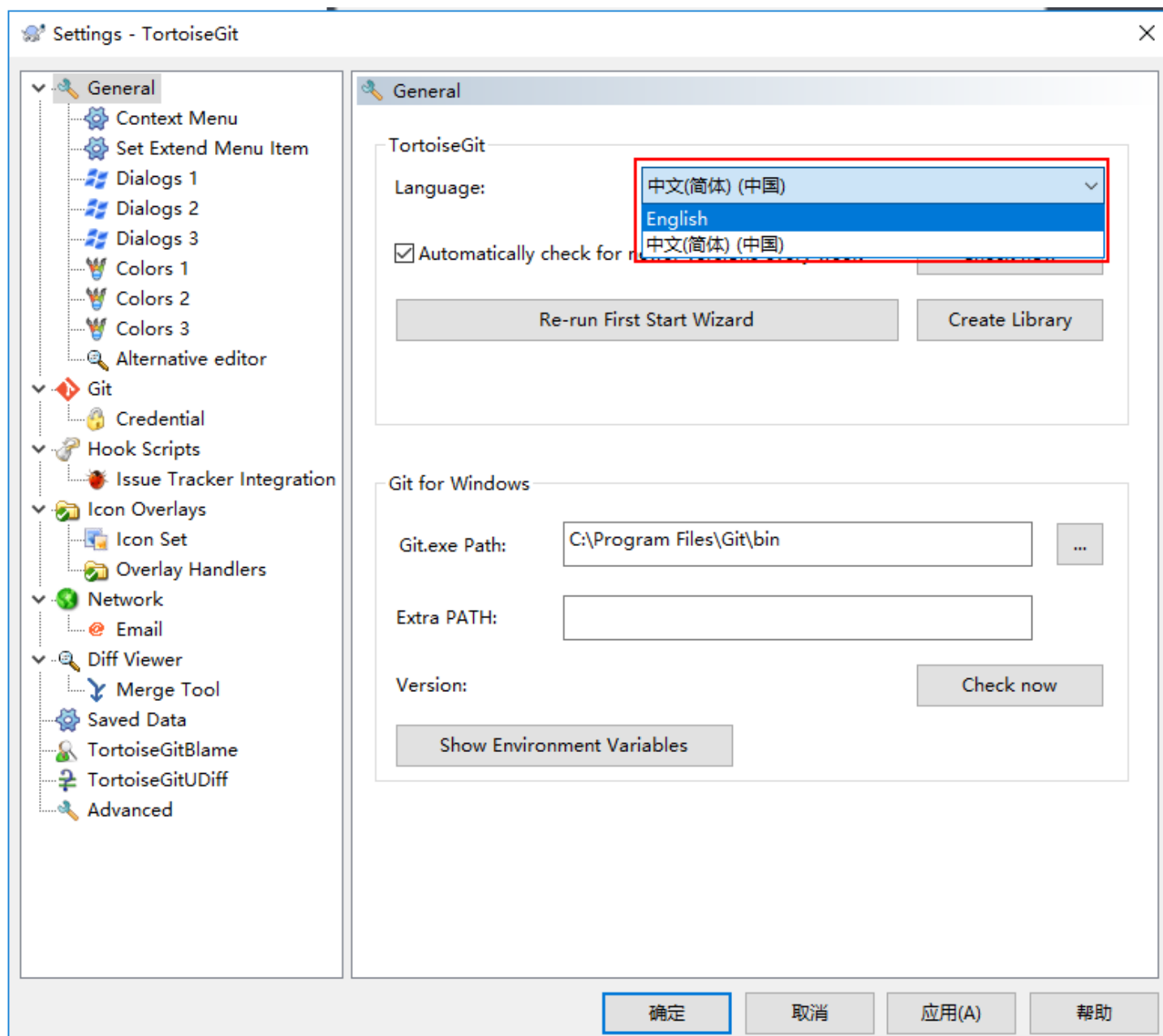


1543310501841

- 直接下一步, 即可安装, 安装完成后, 就可以修改为中文



1543310580094



1543310634219

## 5. 使用git管理文件版本

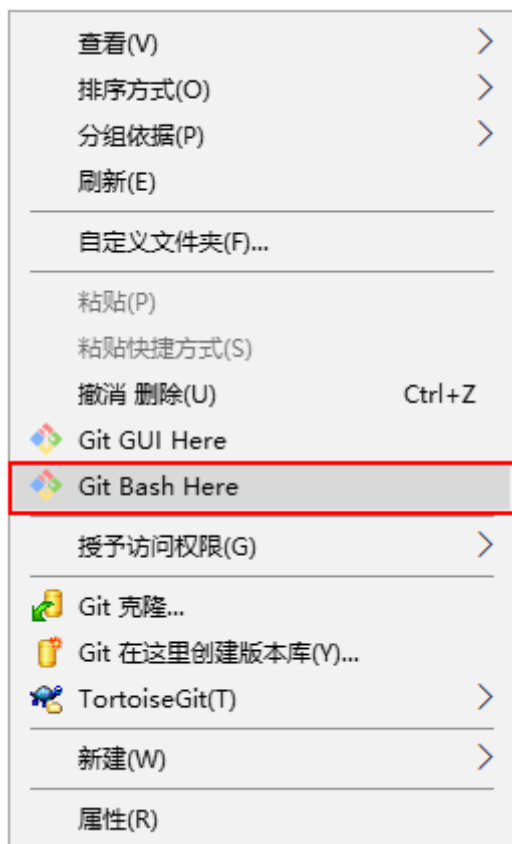
### 5.1 创建版本库

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，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）。空目录名称可以自定义

#### 5.1.1 使用Git bash 创建

- 在当前的空目录中右键选择 Git bash来启动



1543313474309



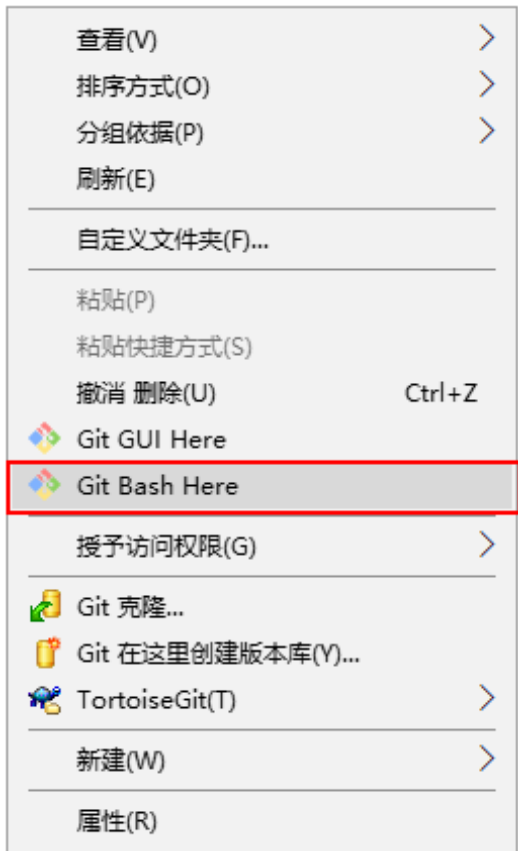
1543313512146

- 创建版本库的命令:

```
1 git init
```

### 5.1.2 使用 TortoiseGit

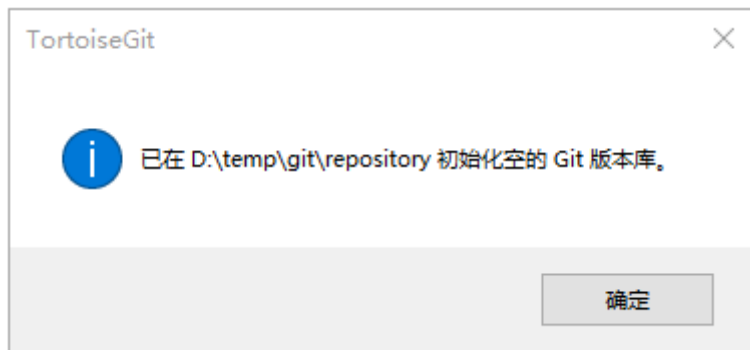
使用TortoiseGit时只需要在空目录中点击右键菜单选择“在这里创建版本库”



1543313716912



1543313759995



1543313776036

- 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



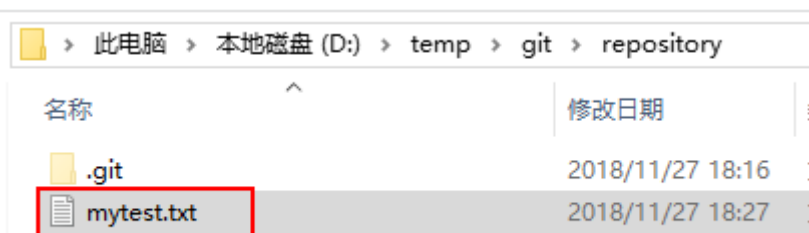
1543313825322

概念说明：版本库：“.git”目录就是版本库，将来文件都需要保存到版本库当中 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

## 5.2 添加文件

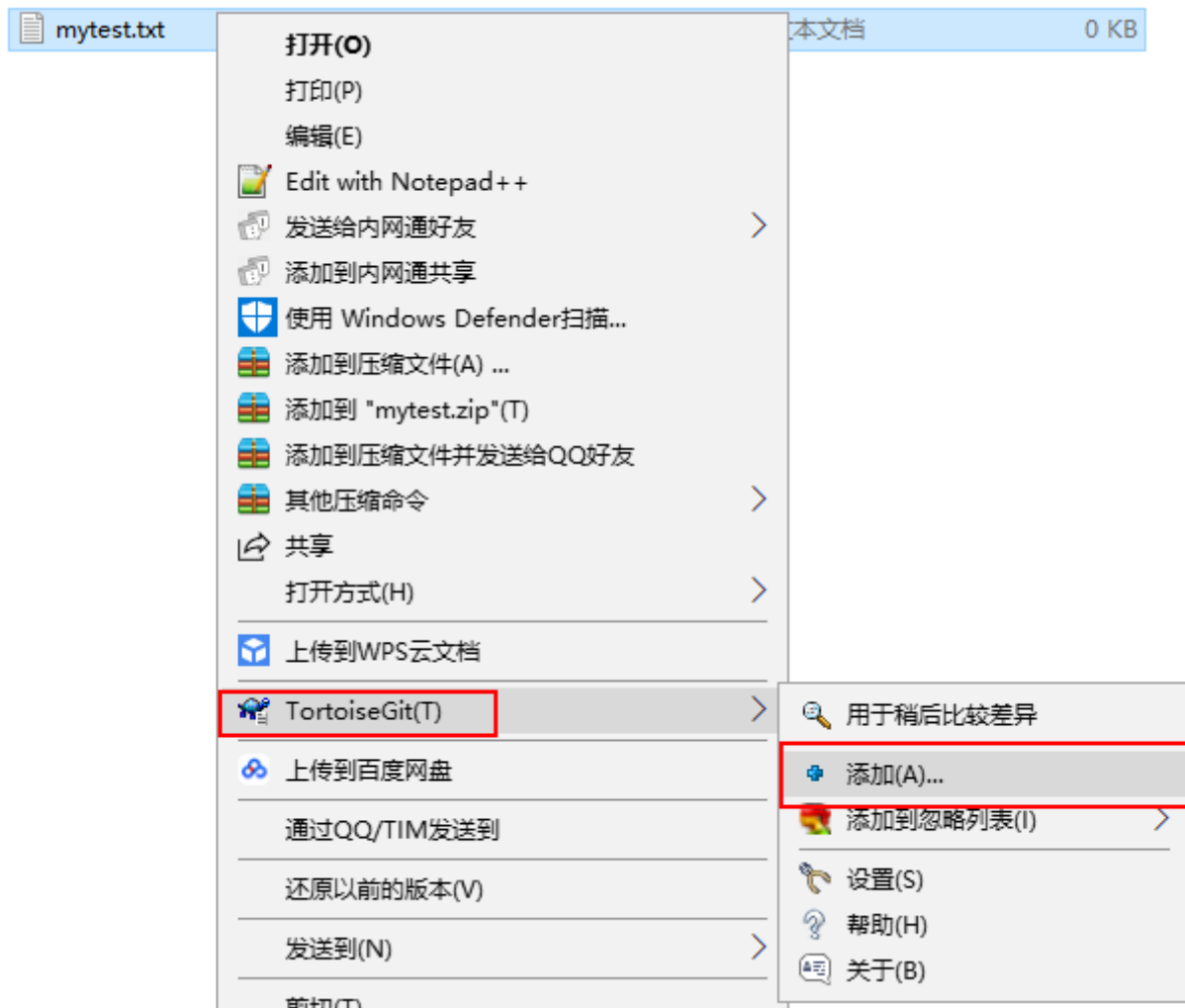
### 5.2.1 添加文件整个过程:

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

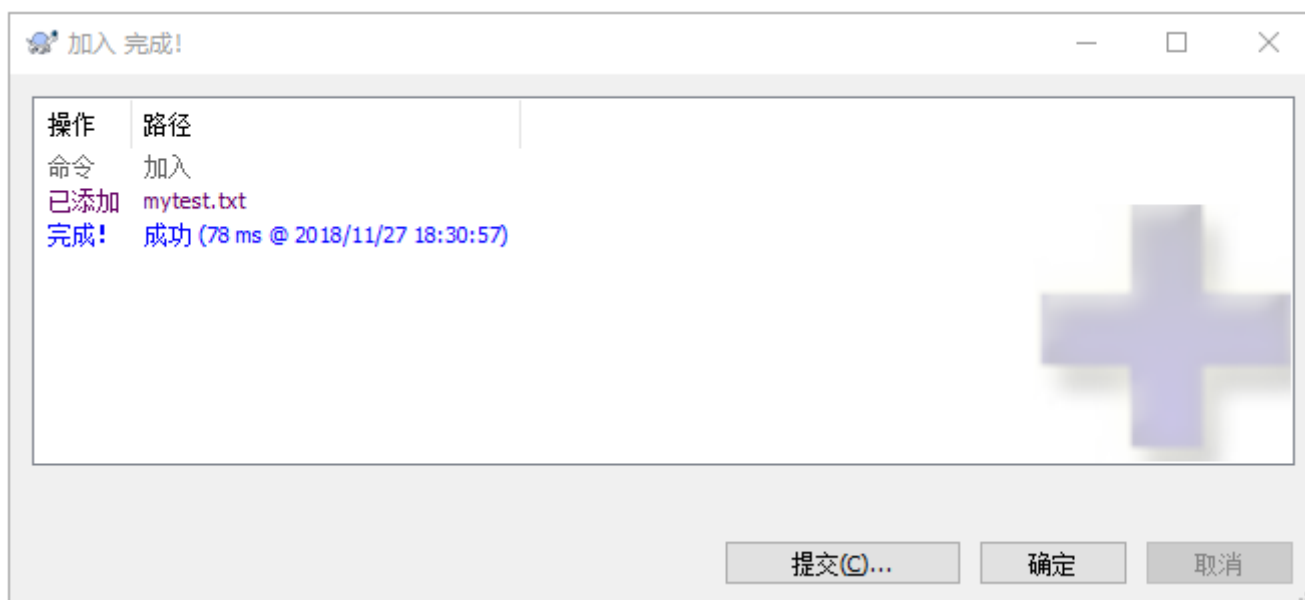


1543314532992

- 2) 选择文件, 右键




1543314632523



1543314669948

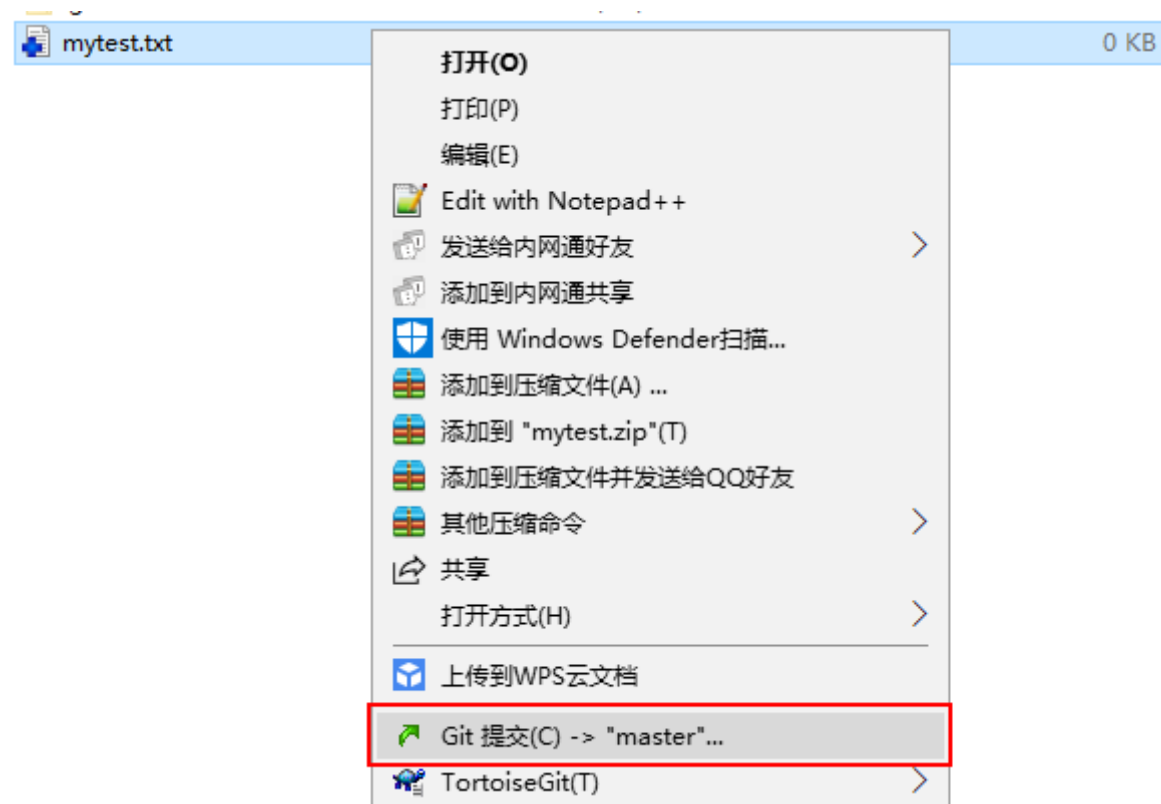
\*\*\*\*\* 此时文件变为带 '+' 号的图标



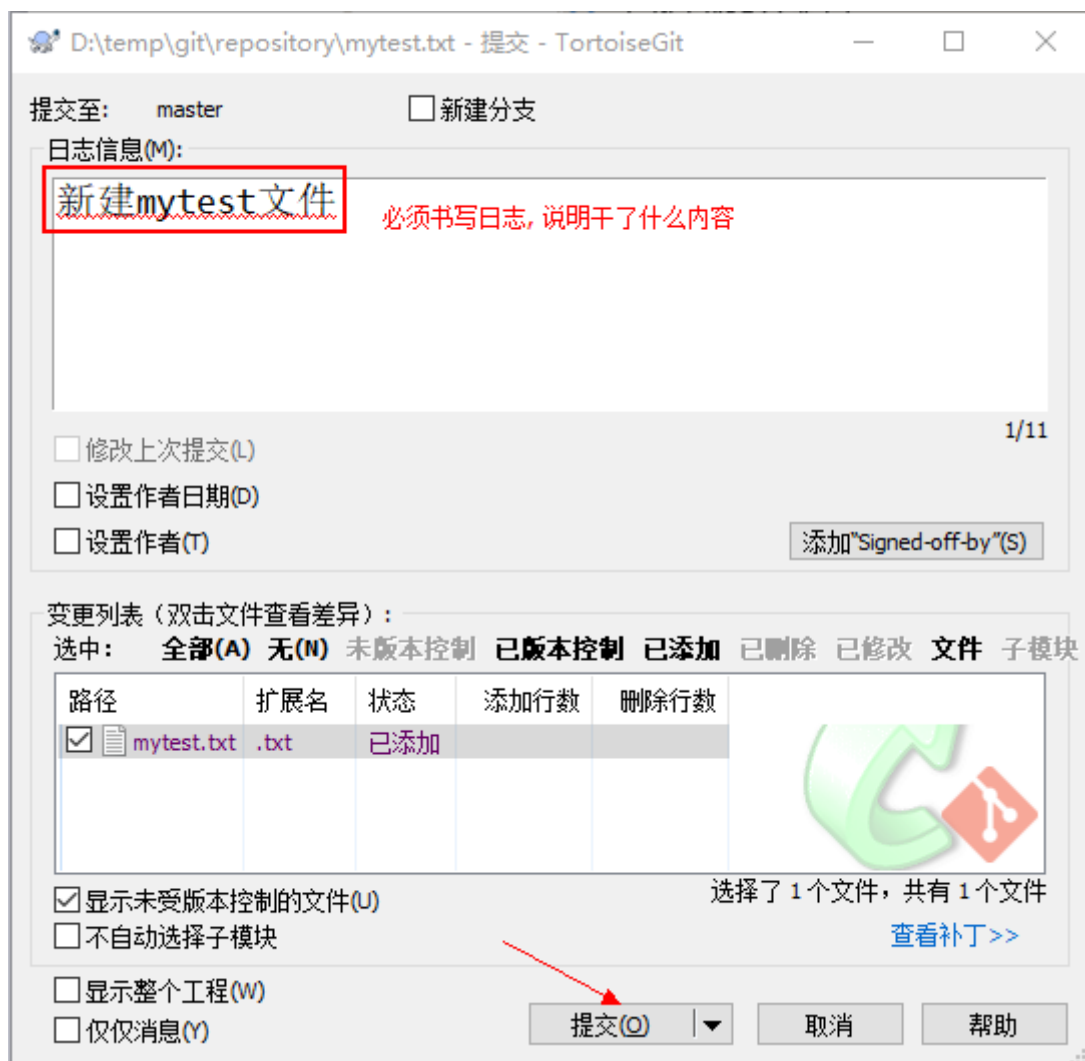
此电脑 > 本地磁盘 (D:) > temp > git > repository			
名称	修改日期	类型	大小
.git	2018/11/27 18:34	文件夹	
 mytest.txt	2018/11/27 18:27	文本文档	0 KB

1543314955755

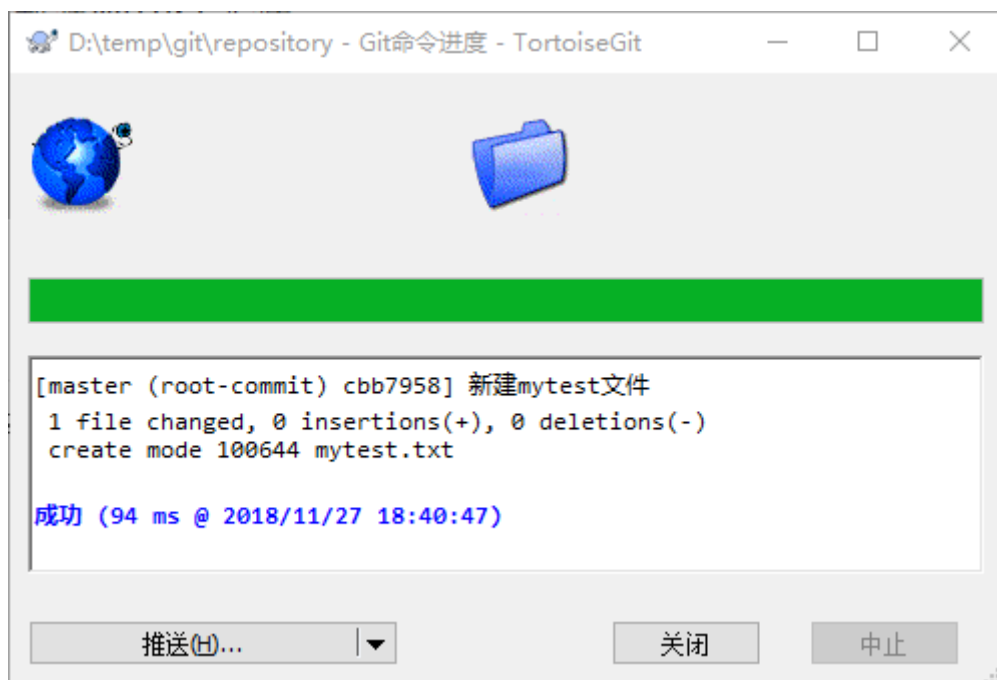
- 3) 提交文件: 在带有+号的文件上, 右键选择提交, 将其保存到版本库中



1543315127945



1543315239885

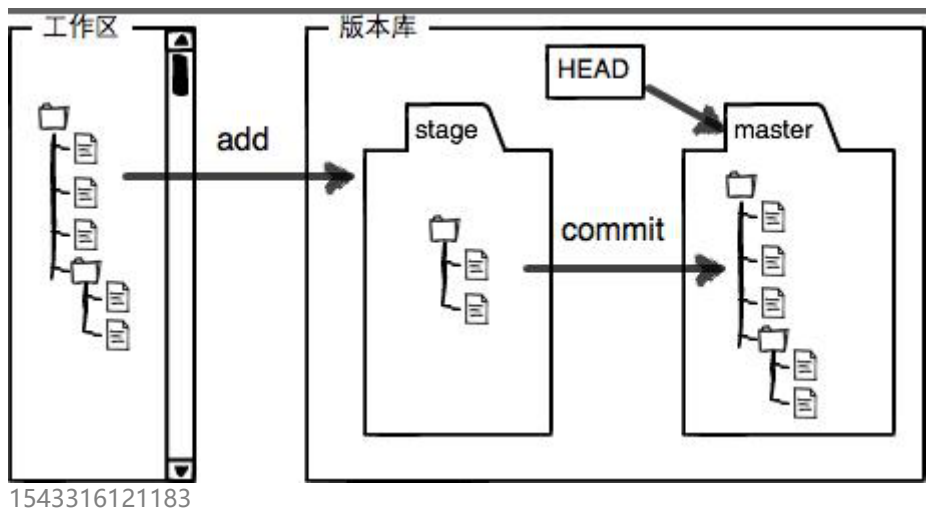


1543315263541

## 5.2.2 工作区 和 暂存区

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



1543316121183

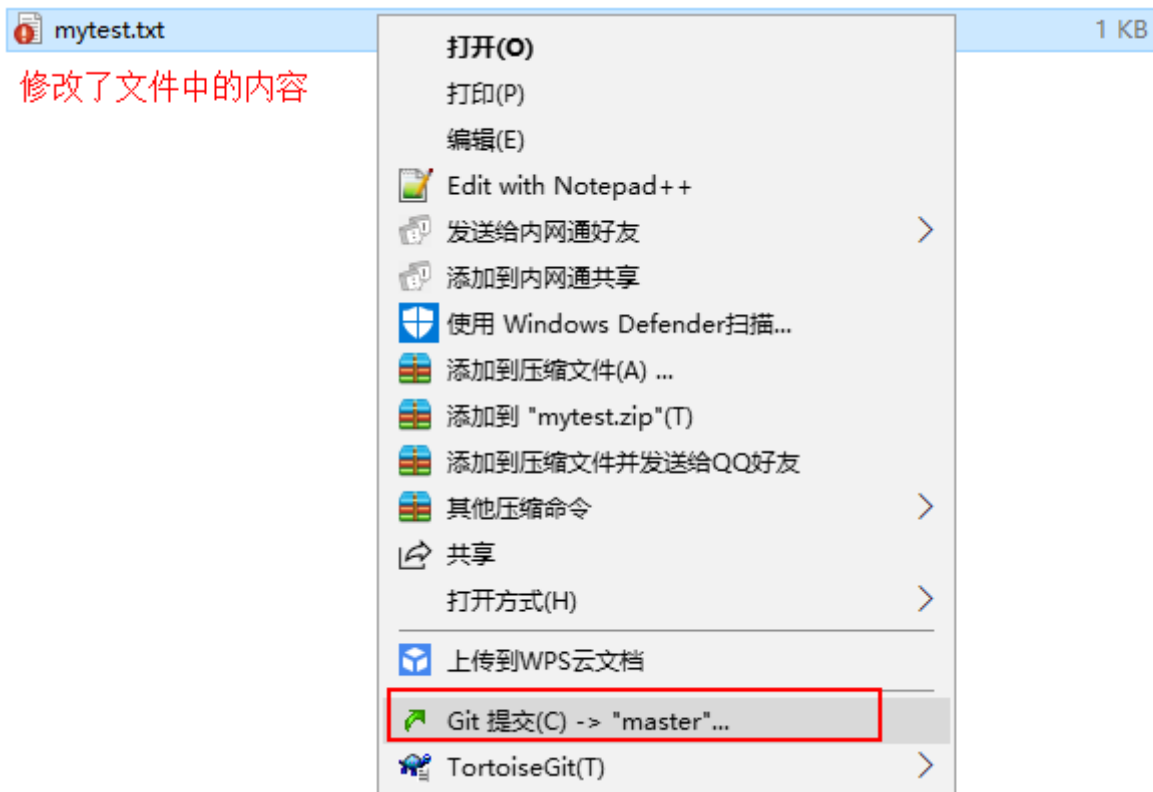
分支和HEAD的概念我们稍后再讲。前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

## 5.3 修改文件

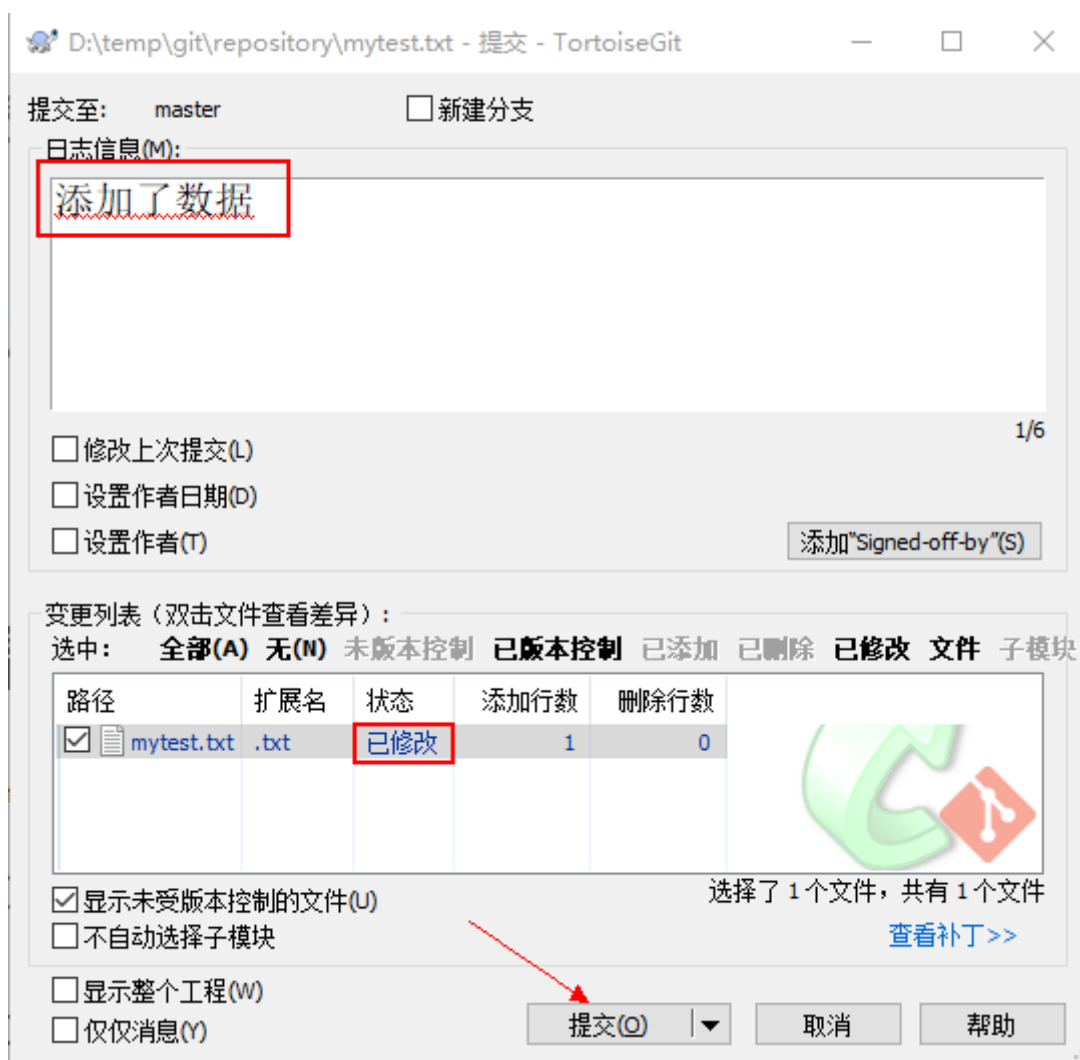
### 5.3.1 提交修改

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”



1543371880910



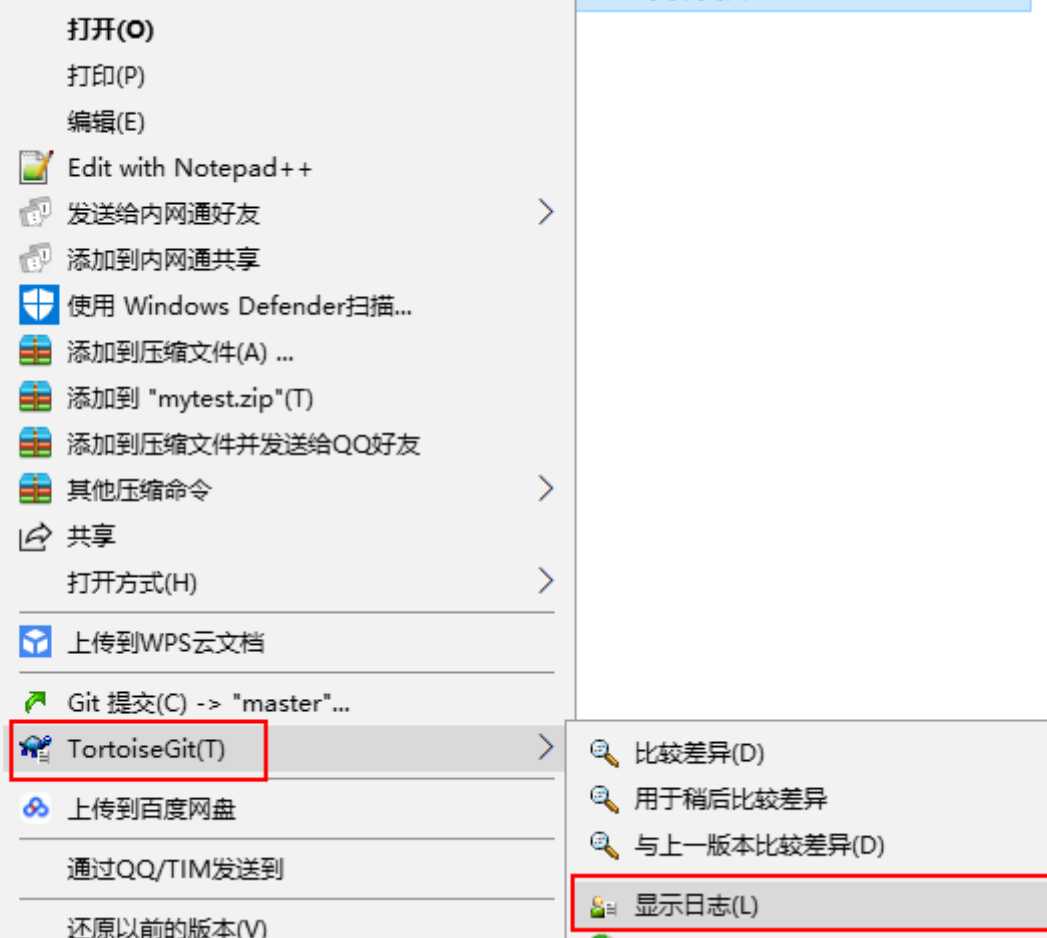
1543371822574



### 5.3.2 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。



1543371972604

D:\temp\git\repository\mytest.txt - 日志信息 - TortoiseGit

master 起始: 2018/11/27 至: 2018/11/28 信息, 路径, 作者, 邮件地址, SHA-1 作者邮件

版本树	操作	信息	作者	日期
		工作树变更		
		<b>master</b> 添加了数据	zhangsan	2018/11/28 10:23:46
		新建mytest文件	zhangsan	2018/11/27 18:40:47

SHA-1: 6d006bd98f6ffc9189c4af138c4695dac4c780a7

\* 添加了数据

路径	扩展名	状态	添加行数	删除行数
mytest.txt	.txt	已修改	1	0

正在显示 2 个修订版本, 从 cbb7958 到 6d006bd - 已选择 1 个版本, 已选择 0 个文件

☐ 显示整个工程(W) ☐ 所有分支(A) 路径过滤 帮助

刷新 统计(T) 遍历方法(H) 查看(V) 确定

1543372085185

### 5.3.3 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”

- 选择查看日志的窗口后, 选择要比较的版本, 直接进行差异化比较即可

D:\temp\git\repository\mytest.txt - 日志信息 - TortoiseGit

master 起始: 2018/11/27 至: 2018/11/28 信息, 路径, 作者, 邮件地址, SHA-1 作者邮件

版本树	操作	信息	作者	日期
		工作树变更		
		<b>master</b> 新添加了数据	zhangs an	2018/11/28 10:54:43
		添加了数据	zhangs an	2018/11/28 10:23:46
		新建mytest文件	zhangs an	2018/11/27 18:40:47

按住Ctrl键,选中要比较的文件右键

- 比较版本差异(C)
- 以标准差异格式显示变更(U)
- 显示 6d006bd..691f067 日志
- 显示 6d006bd...691f067 日志
- 还原这些版本的变更(R)
- 合并到一个提交
- 创建邮件补丁...
- 二分定位 - 开始
- 复制哈希值到剪贴板
- 复制到剪贴板
- 复制日志信息到剪贴板
- 查找日志信息(L)...

路径	扩展名	状态	添加行数
----	-----	----	------

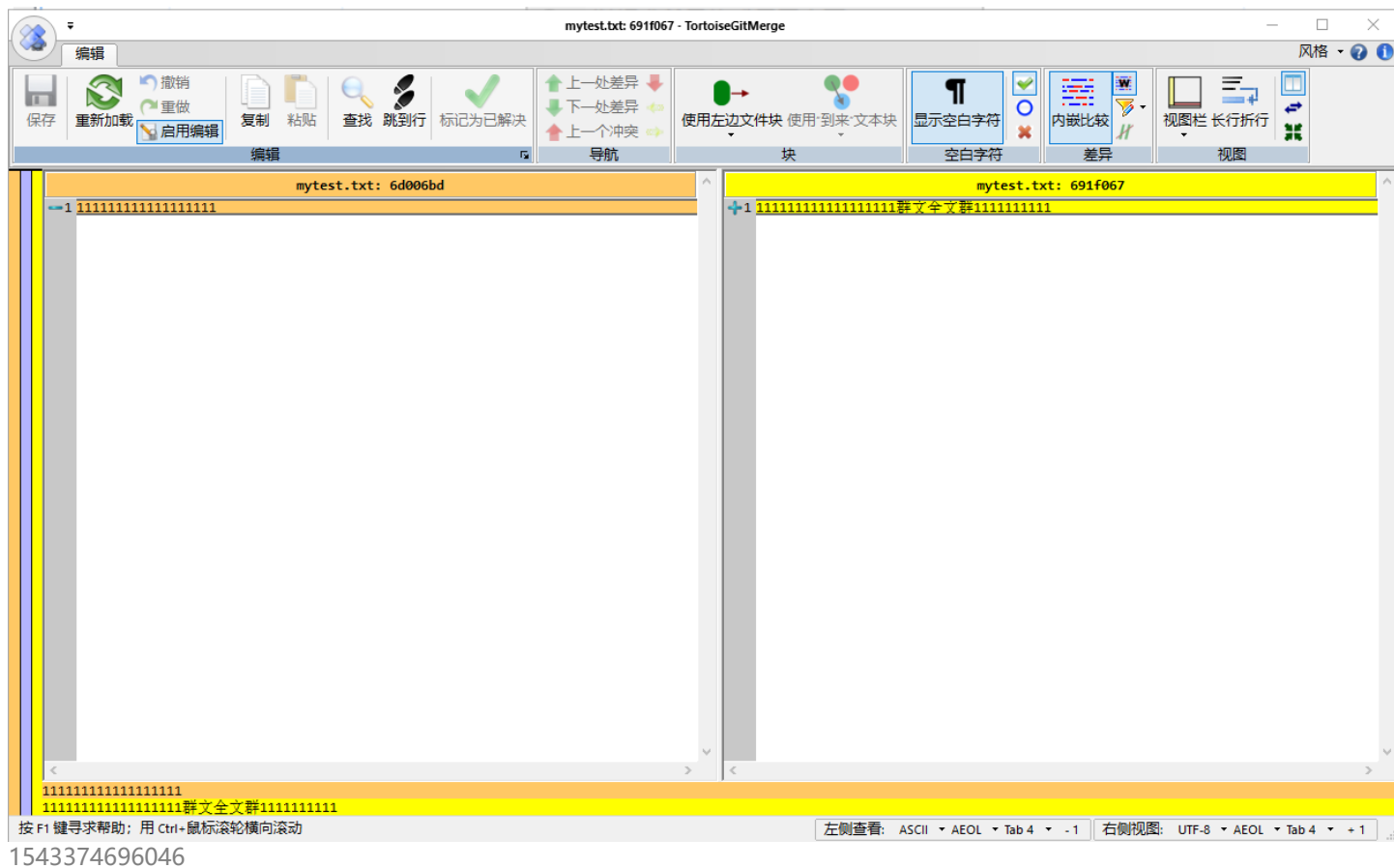
正在显示 3 个修订版本, 从 cbb7958 到 691f067 - 已选择 2 个版本, 已选择 0 个文件

☐ 显示整个工程(W) ☐ 所有分支(A) 路径过滤 帮助

刷新 统计(T) 遍历方法(H) 查看(V) 确定

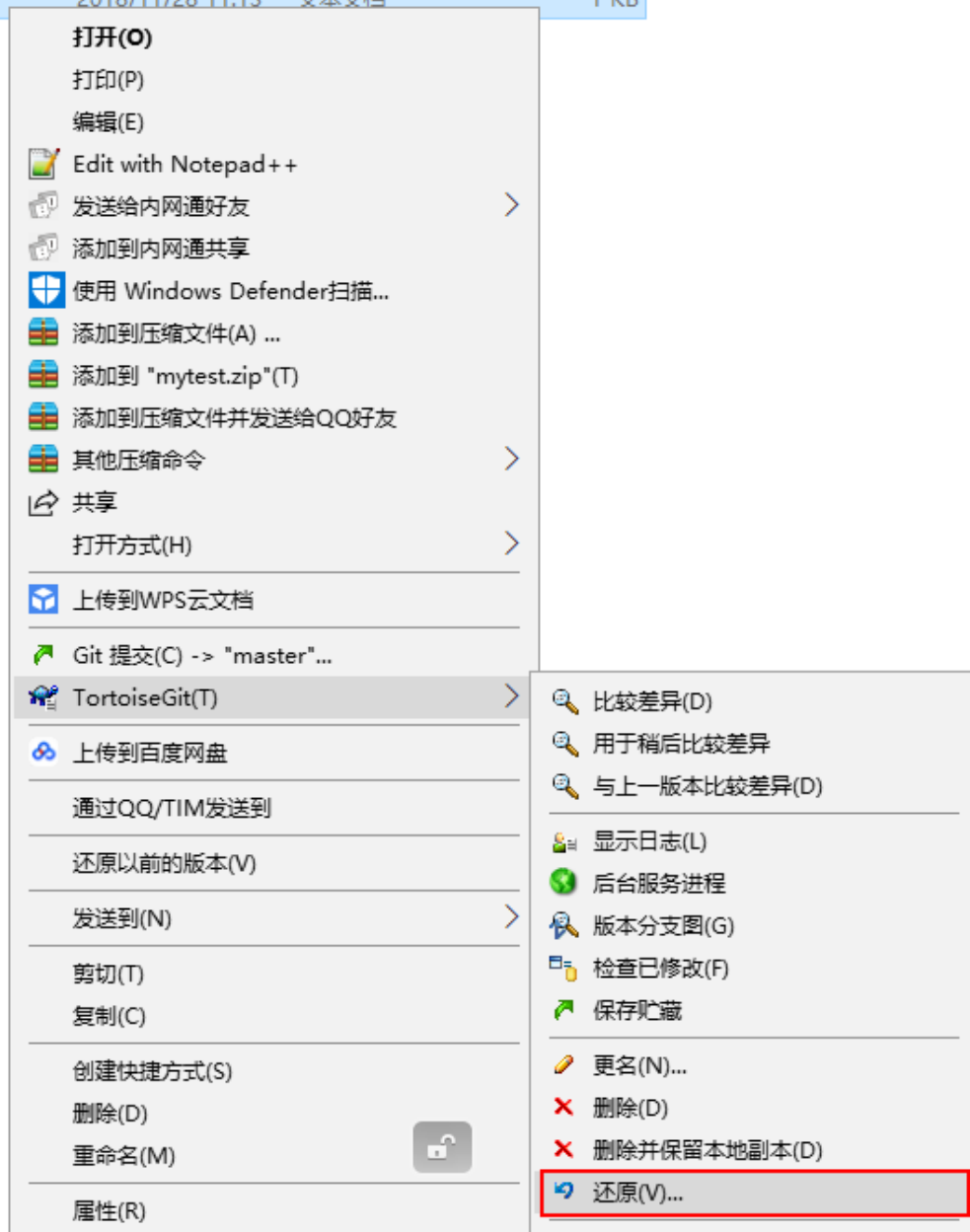
1543374650542



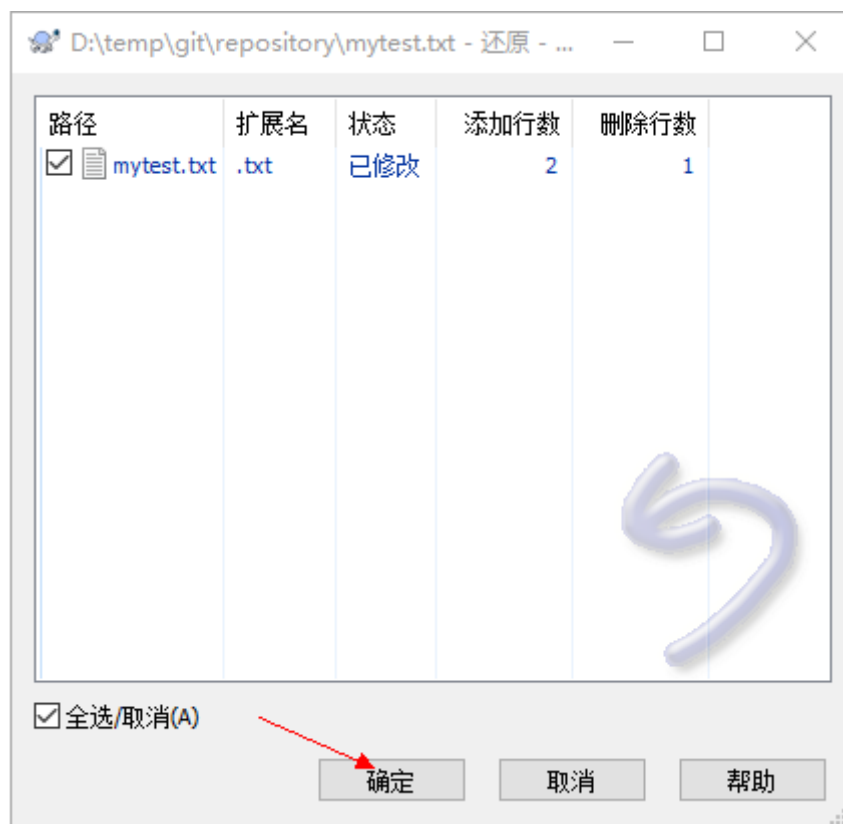


### 5.3.4 还原修改

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



1543374858466



1543374897148

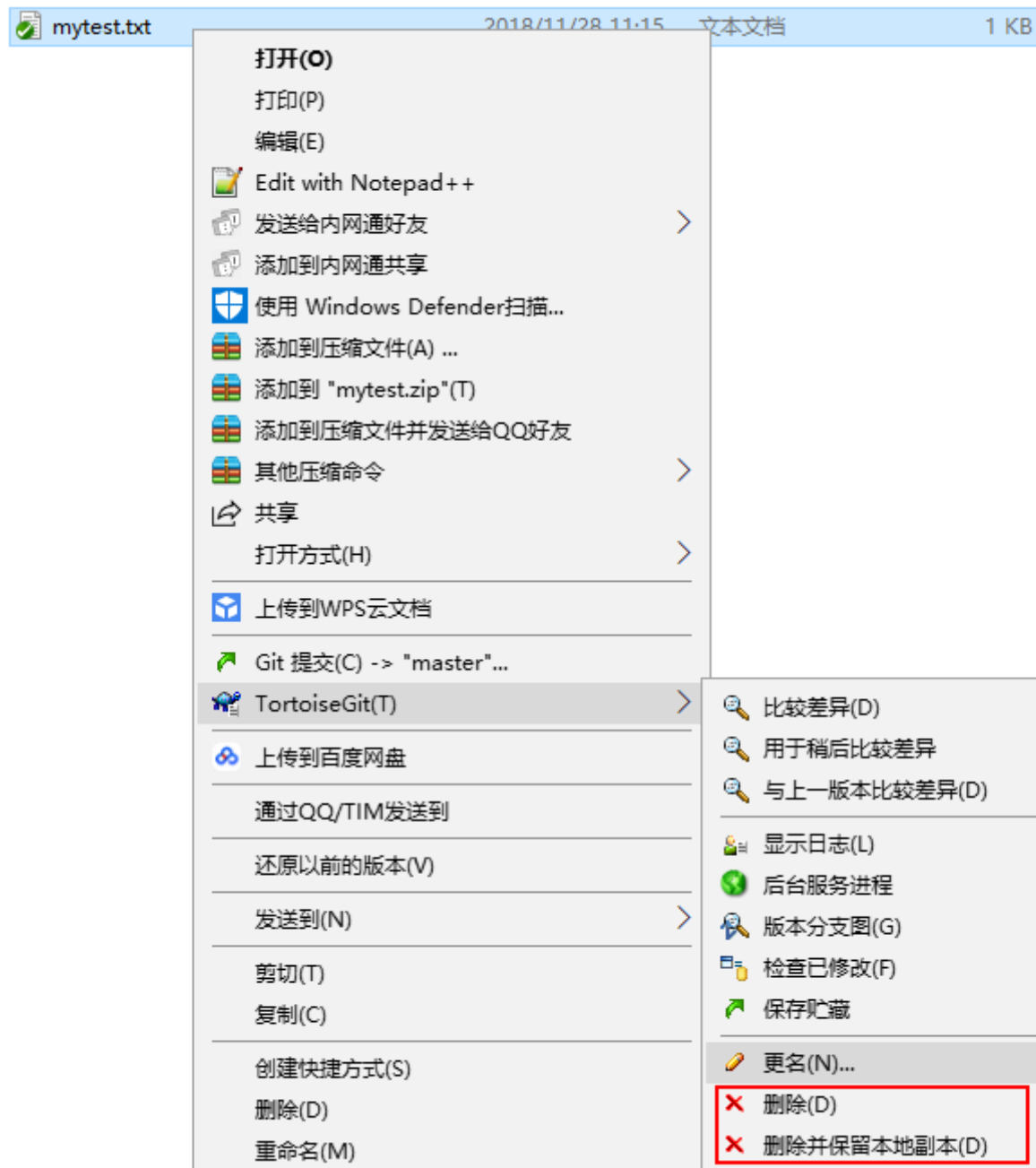


1543374927387

**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！！**

## 5.4 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



1543375306429

## 5.5 案例：将java工程提交到版本库

- 第一步：将参考资料中的java工程project-test复制到工作目录中

名称	修改日期	类型	大小
HelloProjet	2018/11/25 0:07	文件夹	
环境安装包	2018/11/25 0:01	文件夹	
Git Community Book 中文版.pdf	2018/11/24 23:56	WPS PDF 文档	1,118 KB

1543375695399

- 第二步：将工程添加到暂存区

.git

2018/11/28 11:15 文件夹

HelloProjet

mytest.txt

1 KB

### 打开(O)

在新窗口中打开(E)

固定到“快速访问”

添加到 VLC media player 播放列表

Git GUI Here

Git Bash Here

使用 VLC media player 播放

使用 Windows Defender扫描...

添加到压缩文件(A) ...

添加到 "HelloProjet.zip"(T)

添加到压缩文件并发送给QQ好友

其他压缩命令 >

上传到WPS云文档

授予访问权限(G) >

发送给内网通好友 >

添加到内网通共享

Git 同步...

Git 提交(C) -> "master"...

TortoiseGit(T) >

上传到百度网盘

还原以前的版本(V)

包含到库中(I) >

固定到“开始”屏幕(P)

发送到(N) >

剪切(T)

复制(C)

创建快捷方式(S)

删除(D)

重命名(M)

属性(R)

拉取(P)...

获取(E)...

推送(H)...

显示日志(L)

显示引用记录(R)

浏览引用

后台服务进程

版本分支图(G)

版本库浏览器(R)

检查已修改(F)

变基(rebase)...

二分定位 - 开始

解决冲突(O)...

还原(V)...

清理(C)...

切换/检出(W)...

合并(M)...

创建分支(B)...

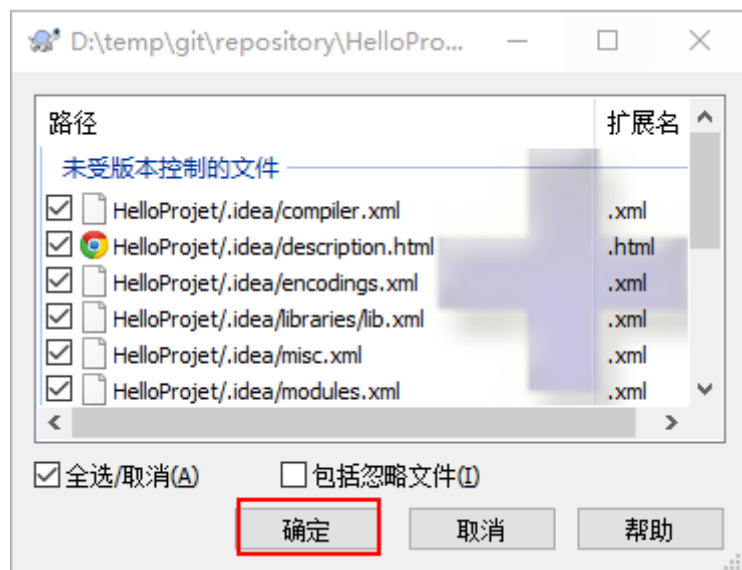
创建标签(T)...

导出(X)...

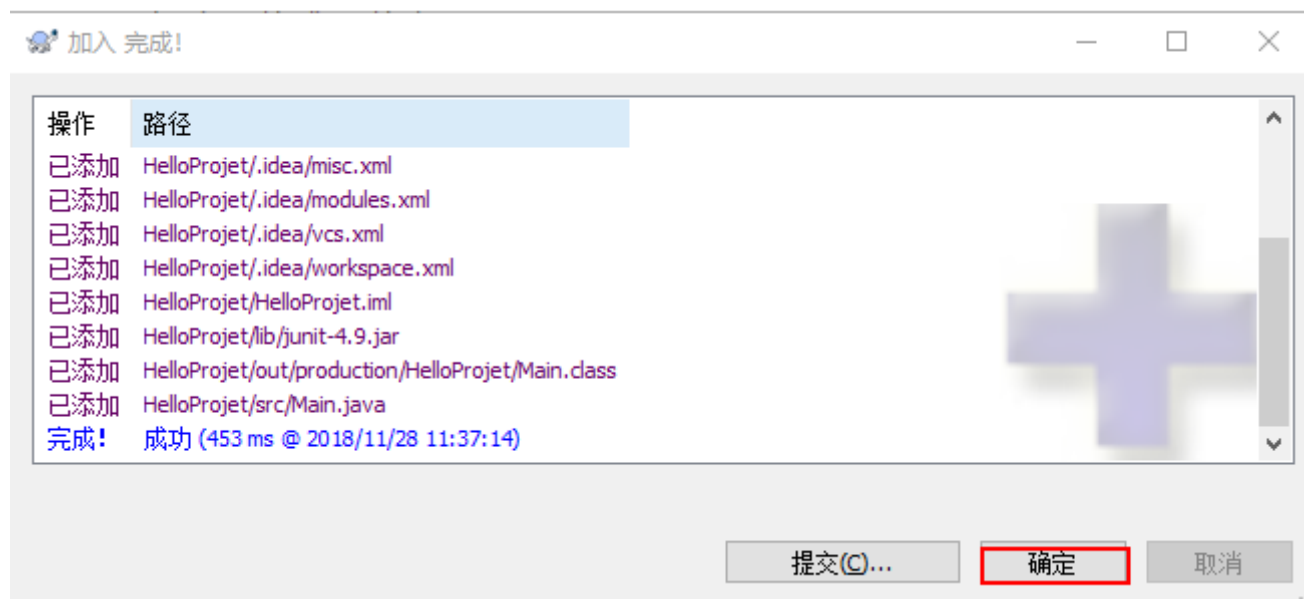
添加(A)...

添加到忽略列表(I) >

1543376150406



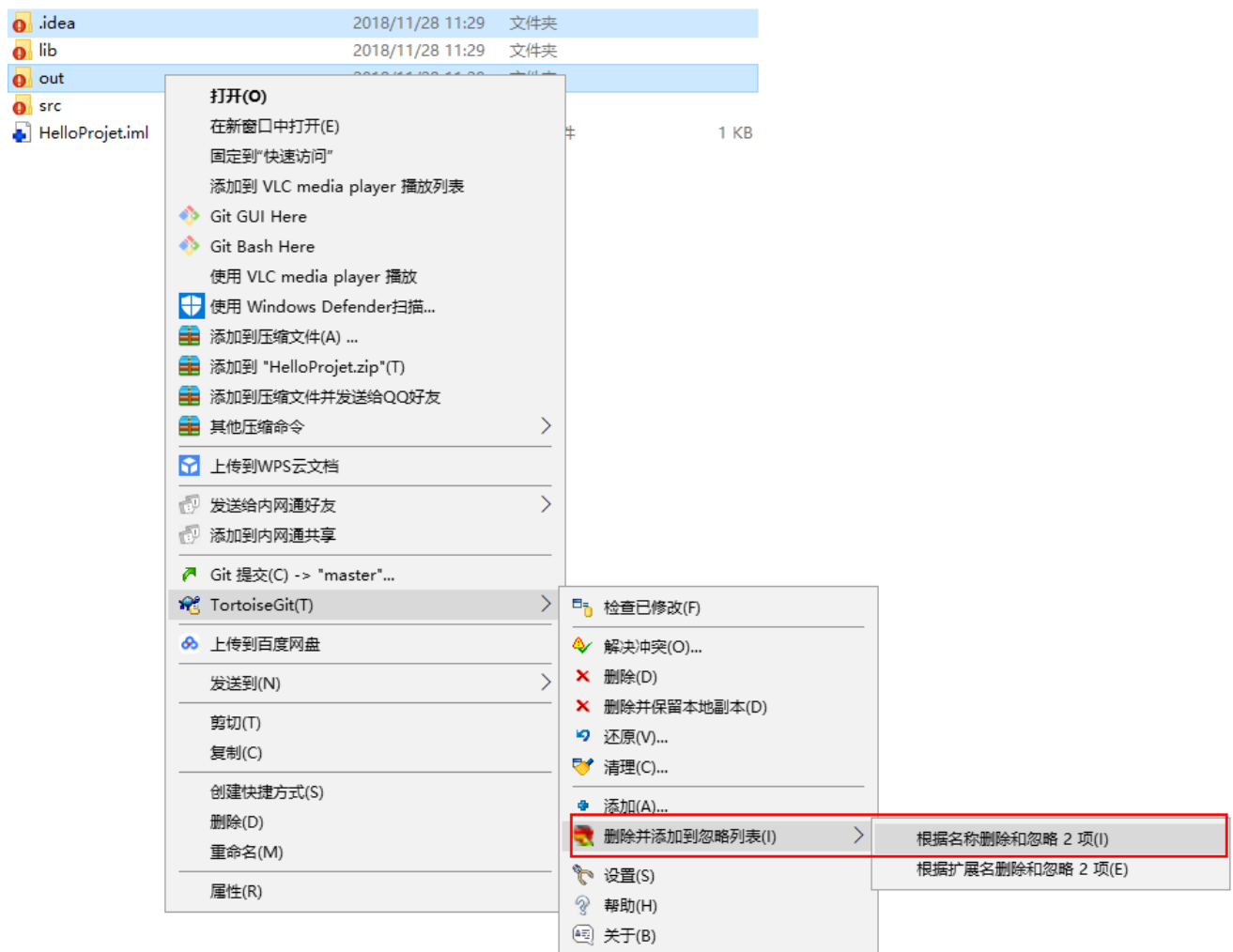
1543376232413



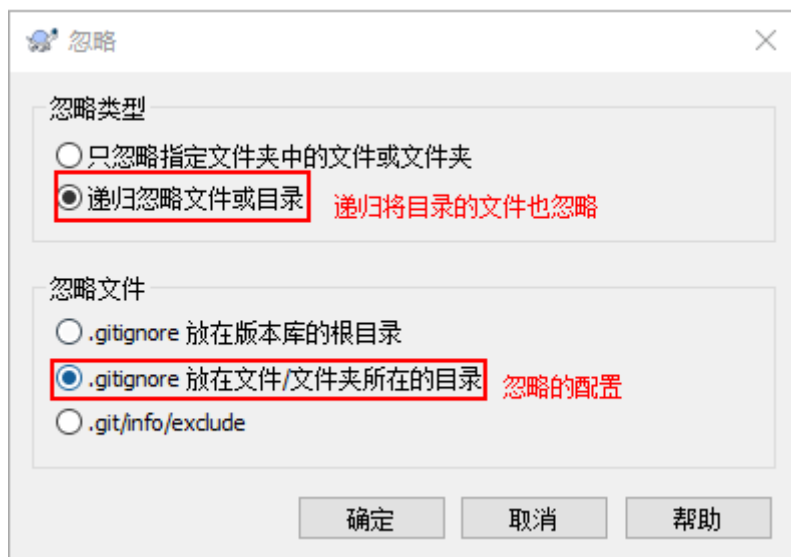
1543376267935

- 第三步: 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“bin”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。



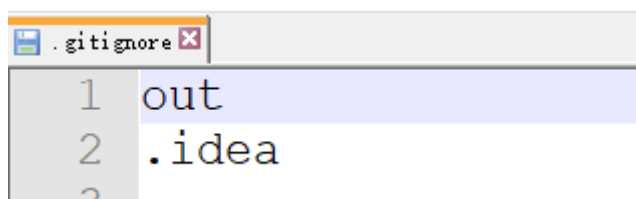
1543383442107





1543383532974

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。其中的内容就是把对应的目录忽略掉



1543383612257

- 第四步: 提交代码
  - 将代码添加到master分支上, 启动.gitignore文件也需要添加到暂存区, 然后提交代码

## 5.6 忽略文件语法规则

空行或是以 # 开头的行即注释行将被忽略。

可以在前面添加正斜杠 / 来避免递归, 下面的例子中可以很明白的看出来与下一条的区别。

可以在后面添加正斜杠 / 来忽略文件夹, 例如 build/ 即忽略build文件夹。

可以使用 ! 来否定忽略, 即比如在前面用了 \*.apk, 然后使用 !a.apk, 则这个a.apk不会被忽略。

\* 用来匹配零个或多个字符, 如 \*.oa] 忽略所有以".o"或".a"结尾, \*~ 忽略所有以 ~ 结尾的文件 (这种文件通常被许多编辑器标记为临时文件); [] 用来匹配括号内的任一字符, 如 [abc], 也可以在括号内加连接符, 如 [0-9] 匹配0至9的数; ? 用来匹配单个字符。

看了这么多, 还是应该来个例子:

1) 忽略 .a 文件

\*.a

2) 但否定忽略 lib.a, 尽管已经在前面忽略了 .a 文件

!lib.a

3) 仅在当前目录下忽略 TODO 文件, 但不包括子目录下的 subdir/TODO  
/TODO

4) 忽略 build/ 文件夹下的所有文件

build/

5) 忽略 doc/notes.txt, 不包括 doc/server/arch.txt  
doc/\*.txt



6) 忽略所有的 .pdf 文件 在 doc/ directory 下的  
doc/\*.pdf

## 6. 远程仓库

### 6.1 添加远程仓库

现在我们已经在本机创建了一个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:

github: <https://github.com/>

码云: <https://gitee.com/>

这两个都可以作为git的远程仓库, GitHub是国外的, 码云是国内的, 其使用上都是类似的, 这里演示将代码上传至码云上的方式:

#### 6.1.1 在 码云上创建仓库

首先你得先在码云上有一个账户，这里就不在演示，注册一个即可，然后就在码云上常见一个仓库

开源中国 2018 年度最后一场技术盛会邀你来约~错过



itcast

zjl0603



概览

动态

项目

所有

个人的

公开的

私有的

...

0

Followers

0

Stars

0

Following

0

Watches

itcast 很懒, 啥也没写

www.zhaojiale.cn

加入于 13分钟前

项目

组织

企业

Public

Forks

Private



创建项目

Search...



暂未加入任何公有项目

1543385244312

# 新建项目

项目名称

自定义一个项目即可(推荐使用英文)

归属

itcast

路径

https://gitee.com/zjl0603/ 此处会自动填充,也可进行修改,此路径将作为后期访问项目的路径

项目介绍 非必填

用简短的语言来描述一下这个项目吧

此处用来说明项目的基本概况

是否开源

☒ 公开 ☐ 私有

表示当前项目是公共的,还是私有的,码云支持创建私有的,而GitHub也支持但是收费

选择语言

请选择项目语言

添加 .gitignore

请选择 .gitignore 模板

添加开源许可证 

开源项目请选择许可证

☐ 使用Readme文件初始化这个项目

☐ 使用Issue模板文件初始化这个项目 

☐ 使用Pull Request模板文件初始化这个项目 

根据不同语言有不同的忽略方案

 导入已有项目

创建

点击创建即可

1543386428661

不管是GitHub 还是码云, 都支持两种同步方式“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githib的服务器上。这两种方式在实际开发中都应用, 所以我们都掌握。接下来我们先看ssh方式。

## 6.1.2 ssh协议

### 6.1.2.1 什么是ssh

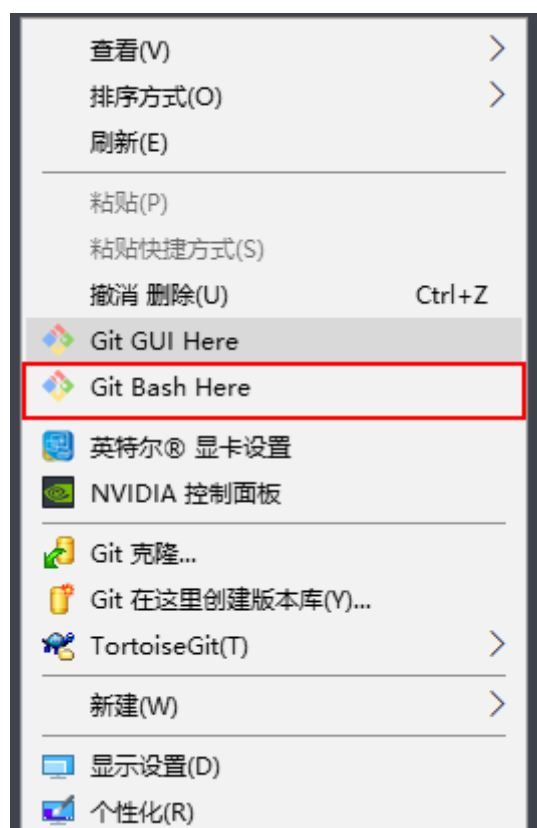
SSH 为 Secure Shell（安全外壳协议）的缩写，由 IETF 的网络小组（Network Working Group）所制定。SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

#### 6.1.2.2 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。

#### 6.1.2.3 ssh密钥的生成

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



1543387012192

git bash 执行命令,生成公钥和私钥

命令: `ssh-keygen -t rsa`

```

admin@jiale MINGW64 ~/Desktop
$ ssh-keygen -t rsa 一直回车即可
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/admin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/admin/.ssh/id_rsa.
Your public key has been saved in /c/Users/admin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:V+N7hjZnE9ISWZ2uyUUvaJDPVUiIbgZFEaFLqu09bN8 admin@jiale
The key's randomart image is:
+---[RSA 2048]---+
  o** oo+o
  ..+ .ooo.
  oo +=oo .
  o .+o=+.o.
  . So..= *.
  o . O .
  . . = *
  . .+ o * .
  ...o. E
+---[SHA256]---+

admin@jiale MINGW64 ~/Desktop
$

```

1543387101193

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:

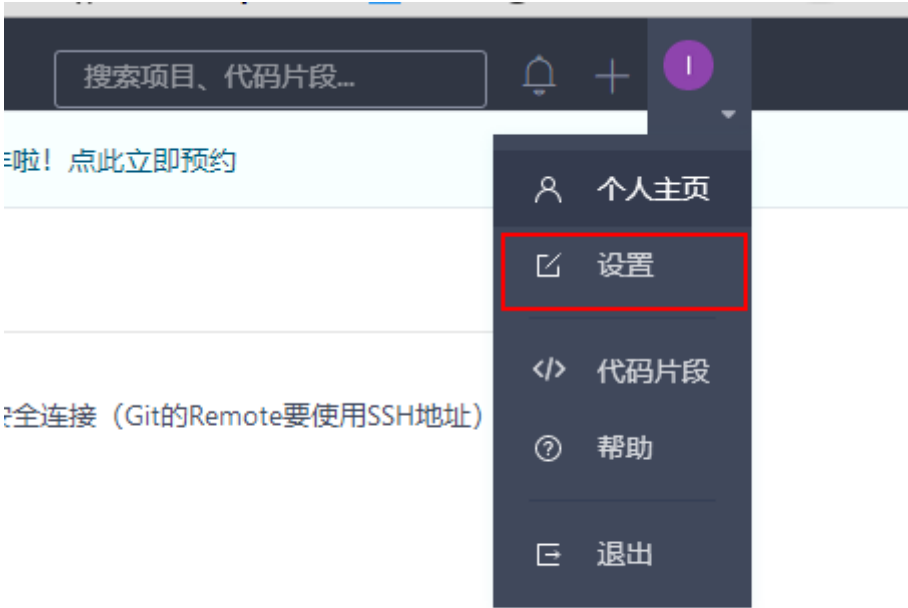
此电脑 > 本地磁盘 (C:) > 用户 > admin > .ssh

名称	修改日期	类型	大小
id_rsa 私钥	2018/11/28 14:37	文件	2 KB
id_rsa.pub 公钥	2018/11/28 14:37	Microsoft Publis...	1 KB
known_hosts	2018/11/7 15:21	文件	1 KB

1543387274255

#### 6.1.2.4 ssh 密钥配置

密钥生成后需要在码云上配置密钥本地才可以顺利访问



1543387484004



1543387612304

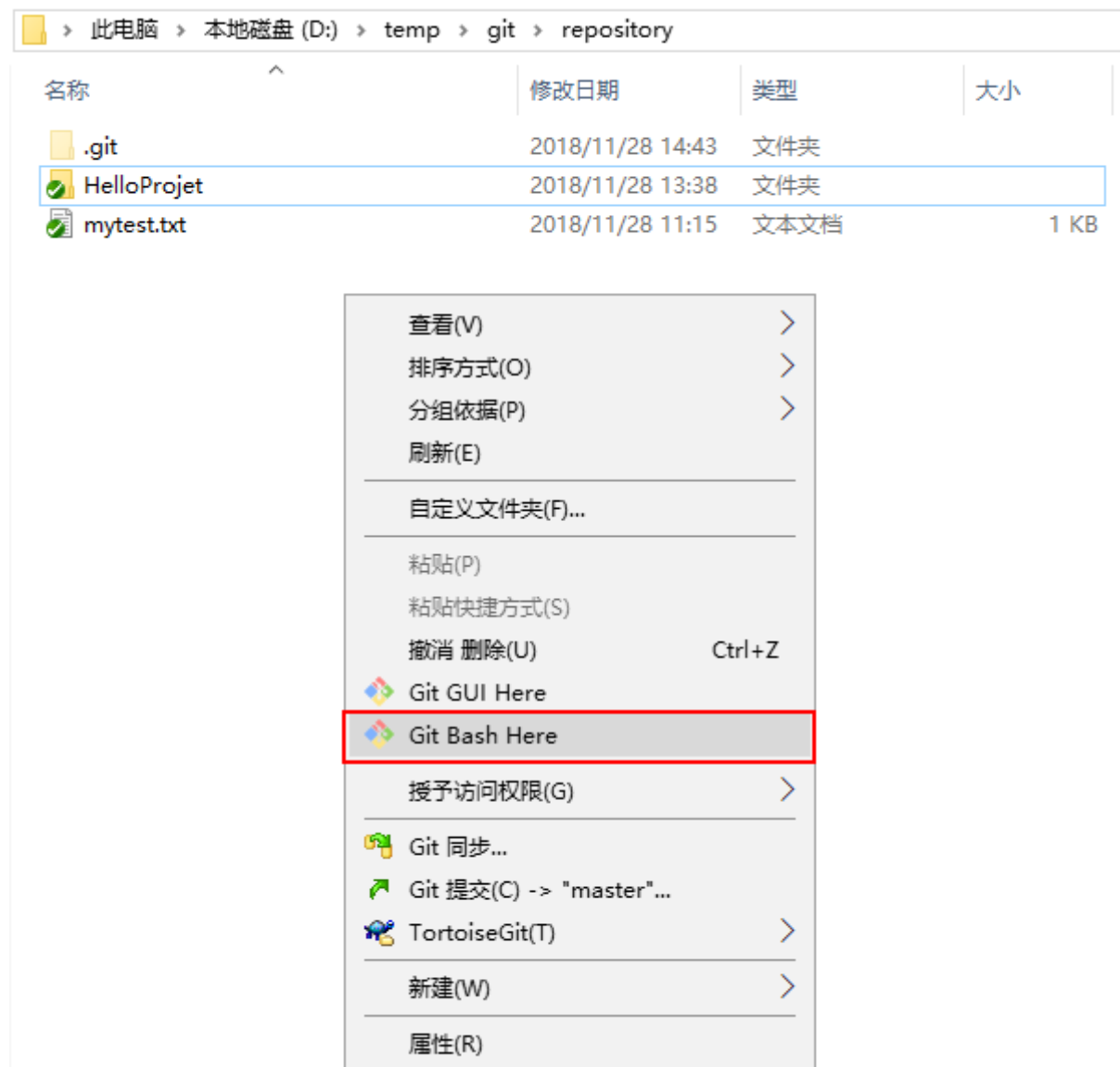
在key部分将id\_rsa.pub文件内容添加进去，然后点击“Add SSH key”按钮完成配置

### 6.1.3 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

### 6.1.3.1 使用 git bash

在仓库所在的目录（D:\temp\git\repository） 点击右键选择 “Git Bash Here” ， 启动git bash程序



1543388253241

然后在git bash中执行如下语句：

```
git remote add origin git@gitee.com:zjl0603/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。

```
MINGW64:/d/temp/git/repository

admin@jjale MINGW64 /d/temp/git/repository (master)
$ git remote add origin git@gitee.com:zjl0603/mytest.git

admin@jjale MINGW64 /d/temp/git/repository (master)
$ git push -u origin master
Counting objects: 17, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (17/17), 213.63 KiB | 0 bytes/s, done.
Total 17 (delta 0), reused 0 (delta 0)
remote: Powered by Gitee.com
To gitee.com:zjl0603/mytest.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

admin@jjale MINGW64 /d/temp/git/repository (master)
$ .....
```

1543390296115

如何出现如下错误:

```
MINGW64:/d/temp/git/repository

admin@jjale MINGW64 /d/temp/git/repository (master)
$ git remote add origin git@gitee.com:zjl0603/mytest.git
fatal: remote origin already exists.

admin@jjale MINGW64 /d/temp/git/repository (master)
$ .....
```

1543390073438

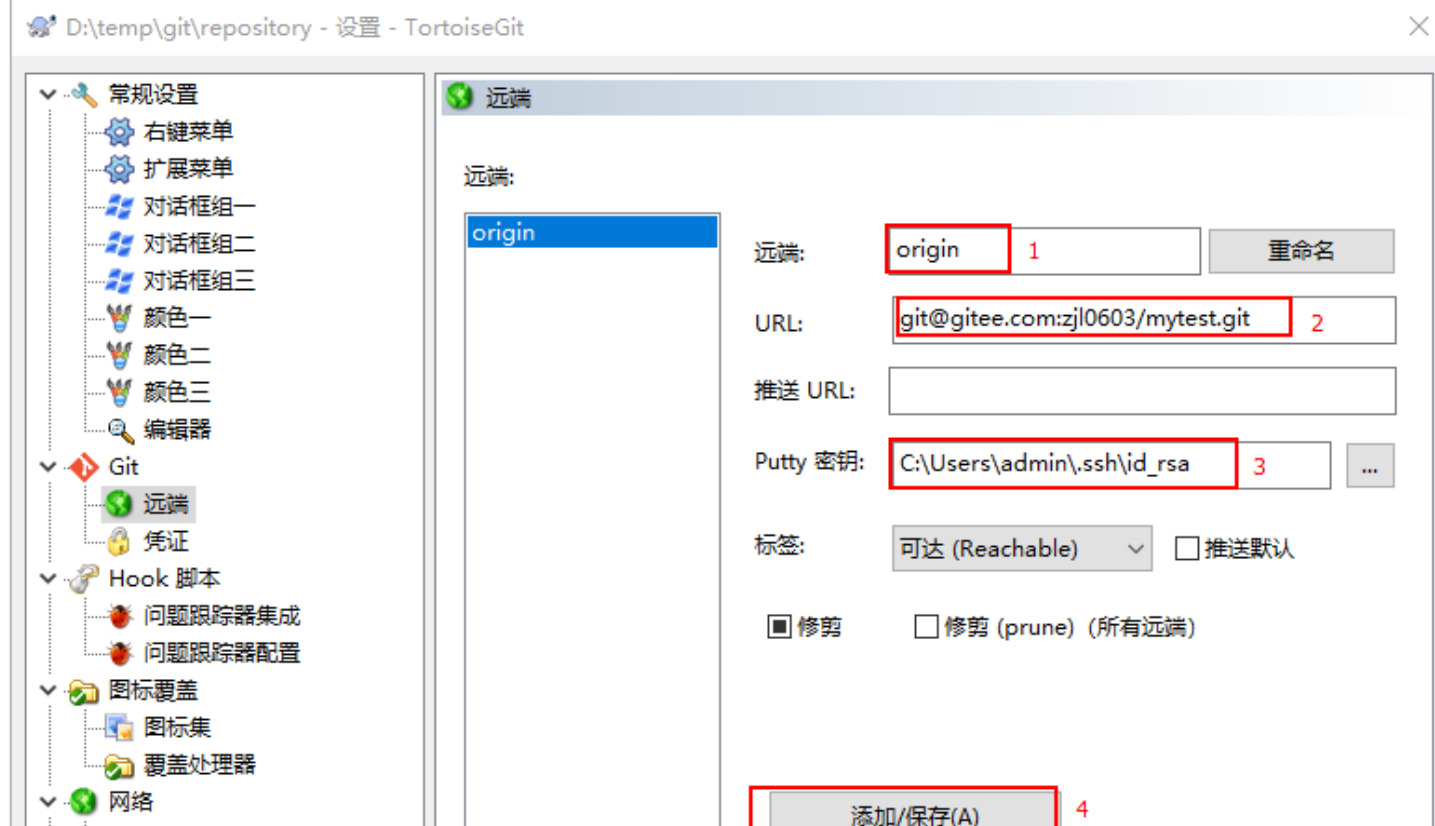
可以先执行如下命令, 然后再执行上面的命令

```
$ git remote rm origin
```

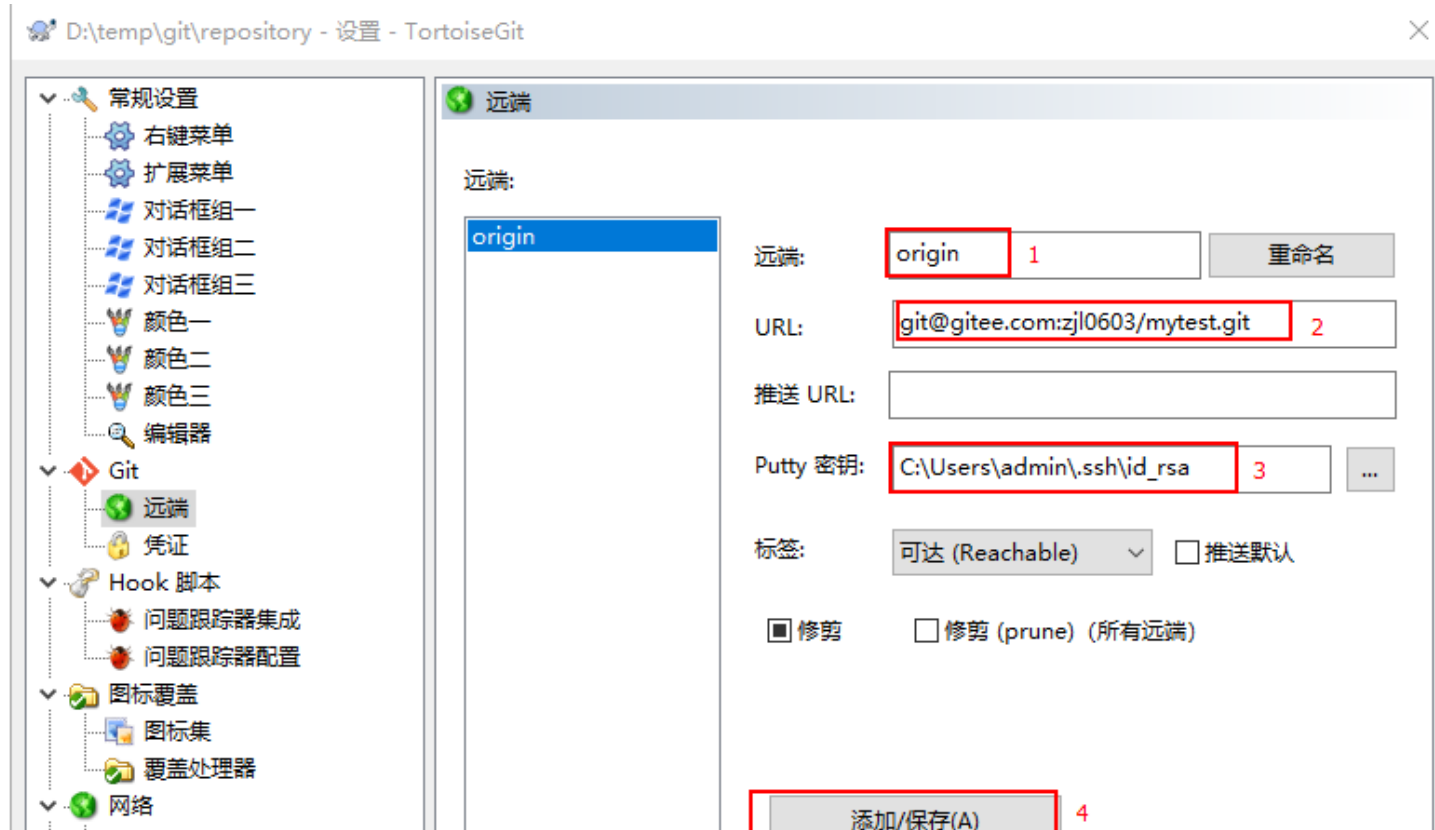
### 6.1.3.2 使用TortoiseGit同步

由于TortoiseGit使用的ssh工具是“PuTTY” git Bash使用的ssh工具是“openSSH”, 如果想让TortoiseGit也使用刚才生成的密钥可以做如下配置:





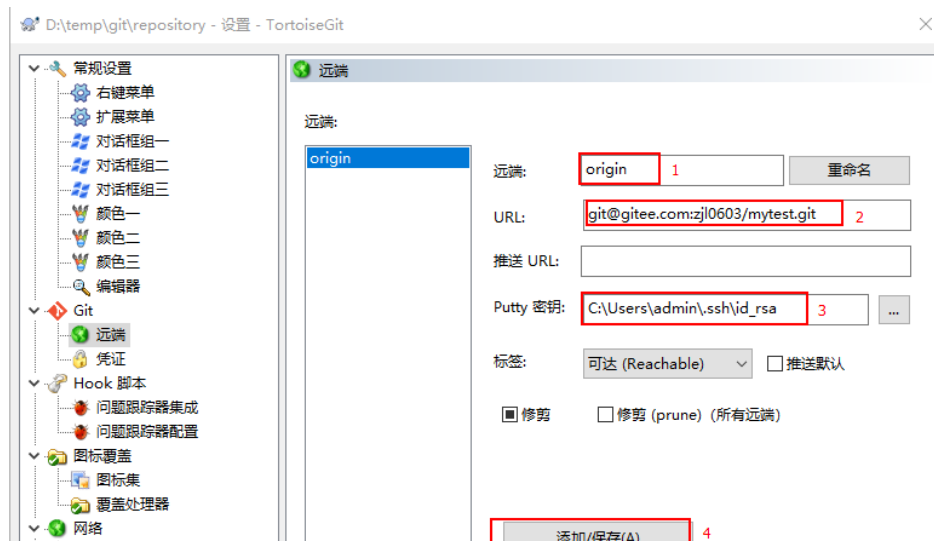
1543391258984



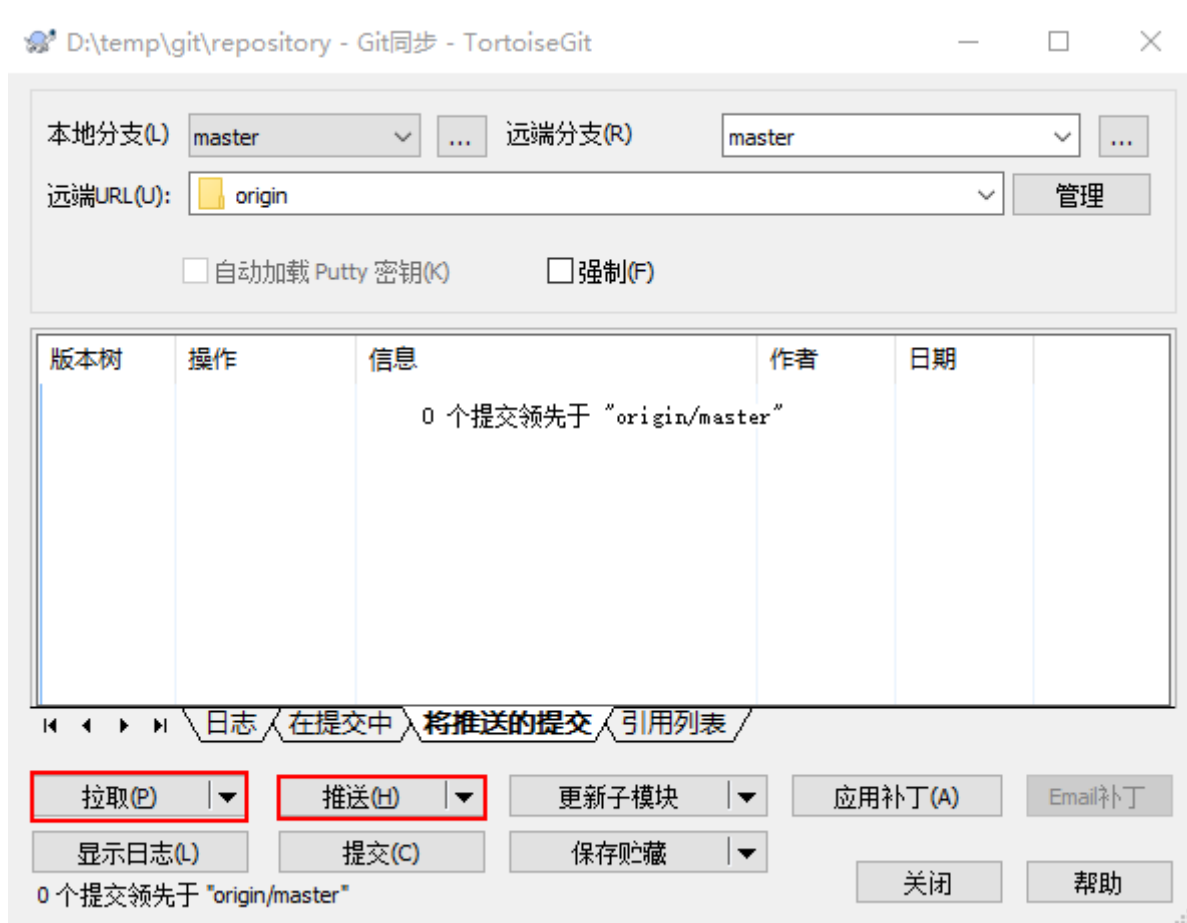
1543391332917

Url: 远程仓库的地址推送URL: 也是相同的Putty密钥: 选择刚才生成的密钥中的私钥

- 在本地仓库的文件夹中单击右键, 选择“Git同步”



1543391407578



1543391488324

## 6.2 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。

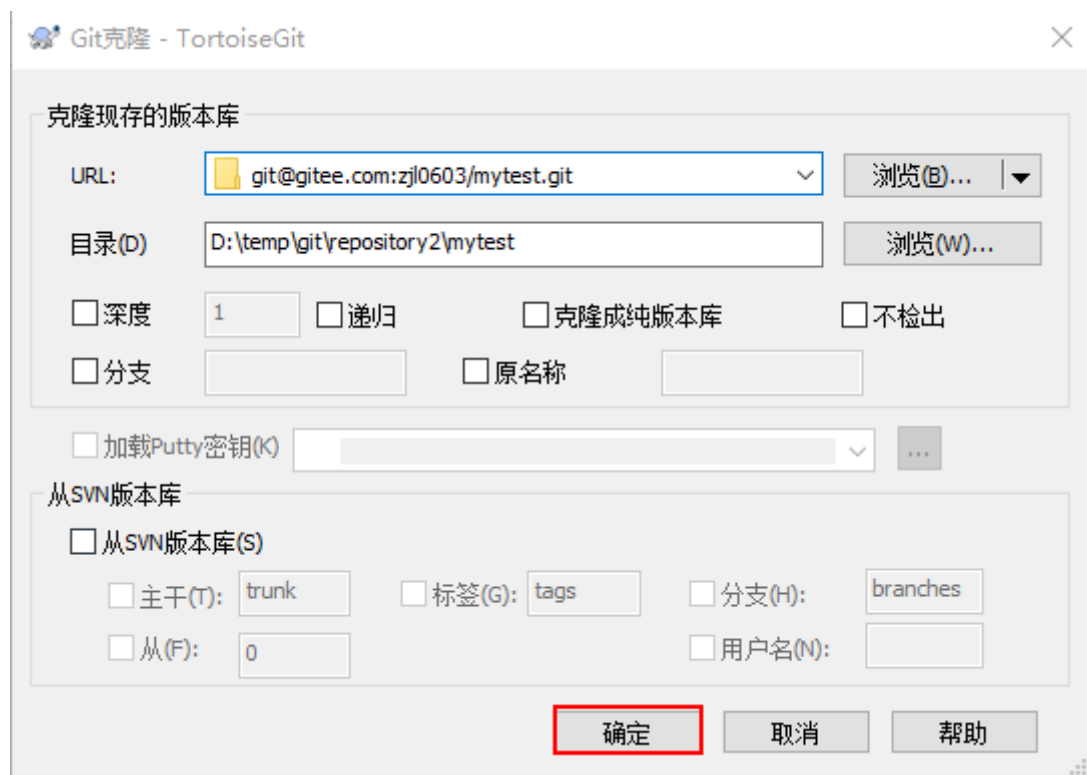
## 6.2.1 使用 git bash

```
1 $ git clone git@gitee.com:zjl0603/mytest.git
```

## 6.2.2 使用 TortoiseGit

在任意目录点击右键：





1543391915674

## 6.3 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

因为在merge前，我们可以查看更新情况，然后再决定是否合并

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）

## 6.4 搭建私有的Git服务器

### 6.4.1 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(资料中)

```
1 yum -y install lrzsz
```

1) 解压缩(/export/servers)

2) cd git-2.5.0

3) autoconf

4) ./configure

5) make

6) make install

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

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。

## 6.4.2 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@192.168.72.141/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@192.168.72.144:first

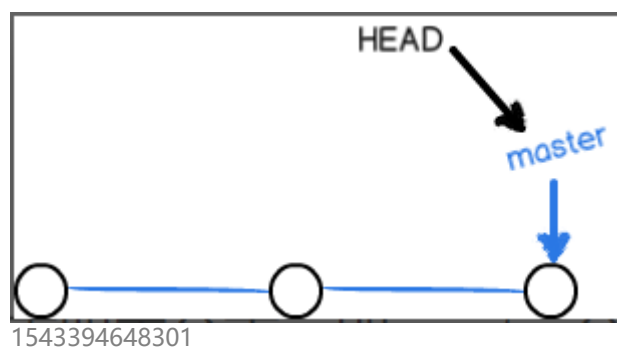
使用TortoiseGit同步的话参考上面的使用方法。

## 7. 分支管理

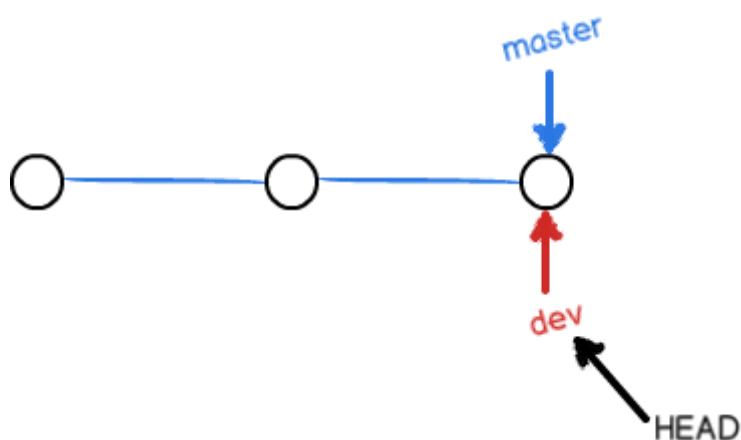
### 7.1 创建合并分支

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：

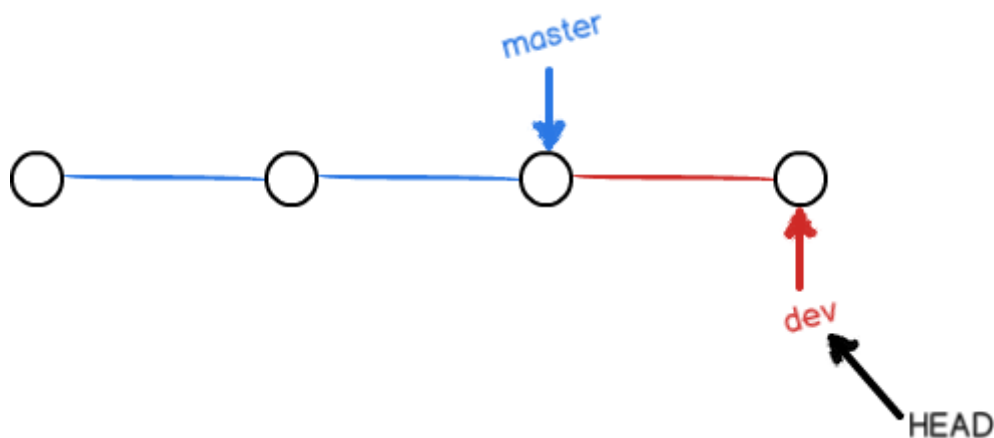


每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长。当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



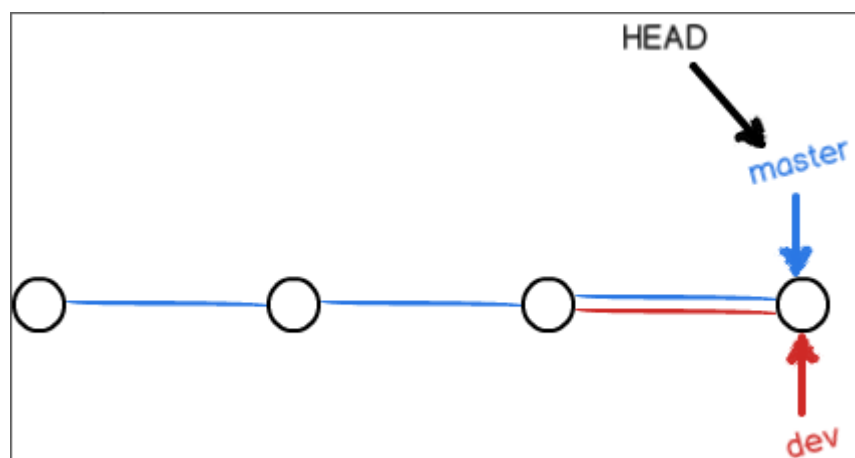
你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！

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：



1543394743673

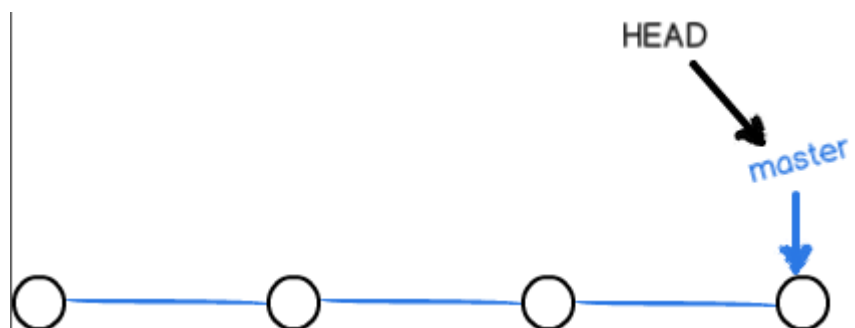
假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



1543394801359

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：



1543394841235

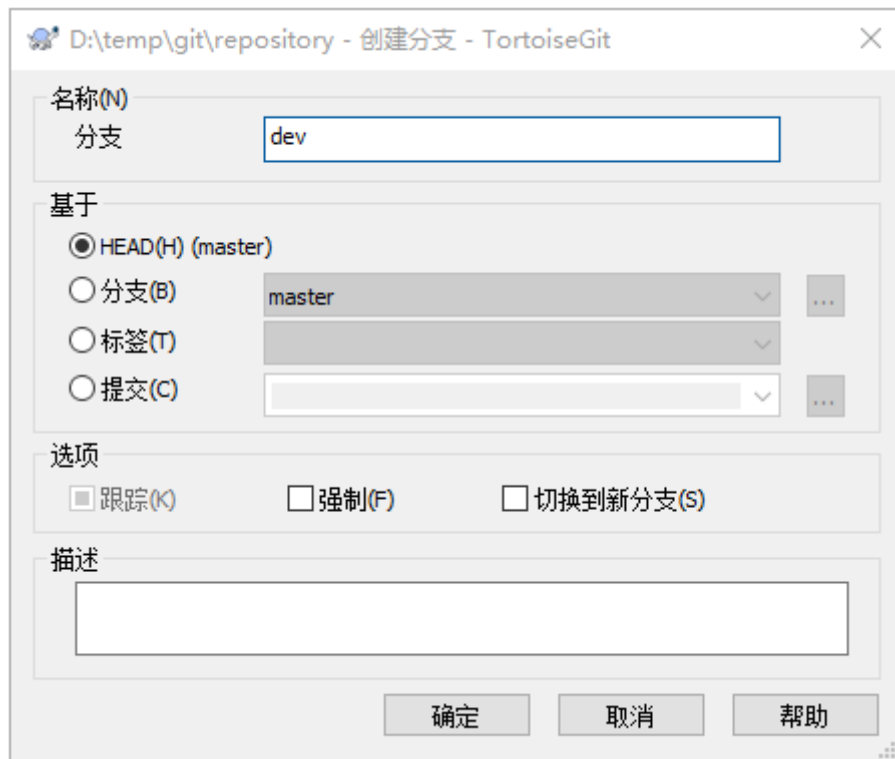
## 7.2 使用TortoiseGit实现分支管理

使用TortoiseGit管理分支就很简单了

### 7.2.1 创建分支

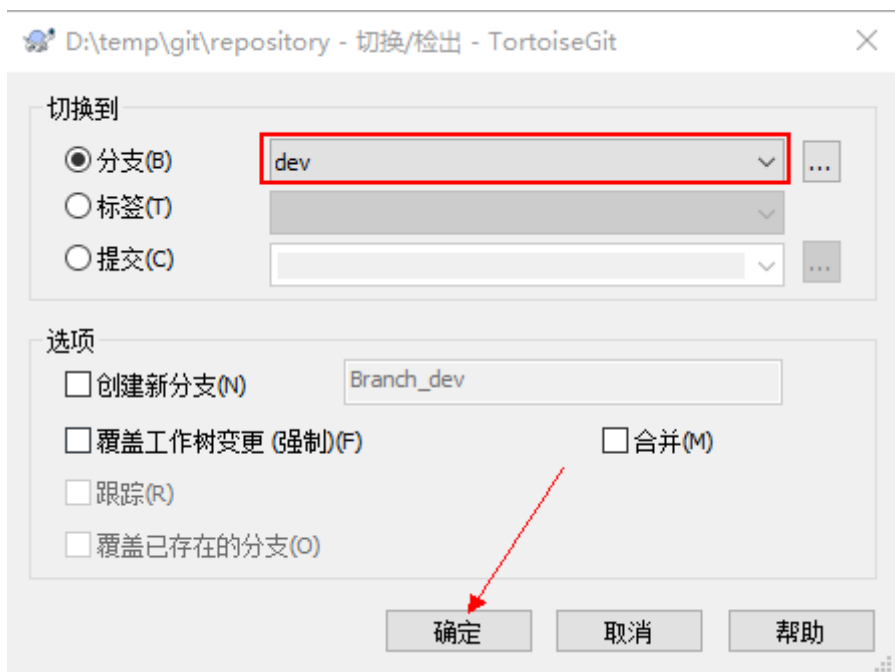


- 在本地仓库文件夹中点击右键，然后从菜单中选择“创建分支”：



1543396732586

- 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：



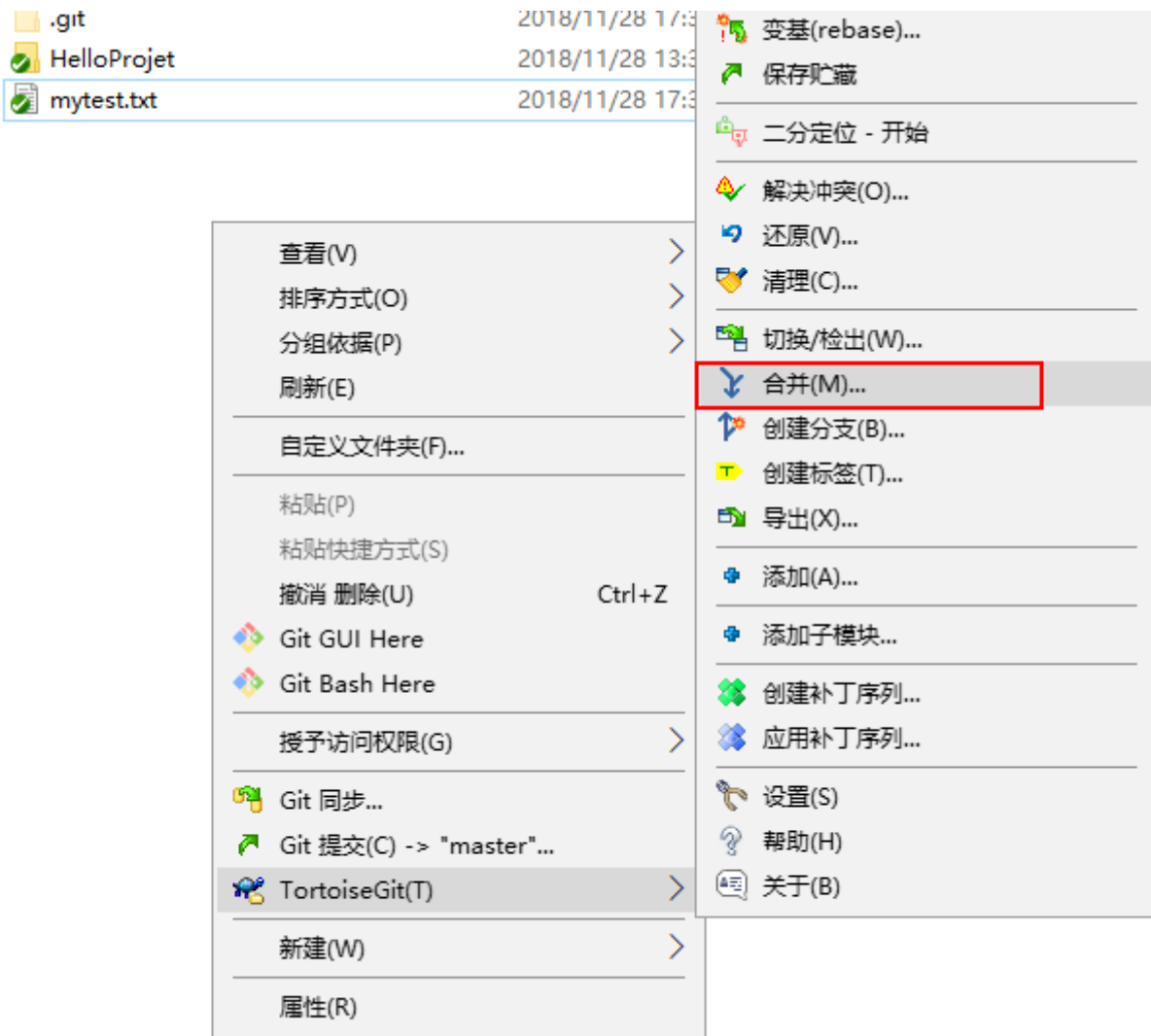
1543396889011

## 7.2.2 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。







1543397849676





D:\temp\git\repository\mytest.txt - 提交 - TortoiseGit

提交至: **master** ☐ 新建分支

日志信息(M):

master节点添加文件内容

1/9

☐ 修改上次提交(L) ☐ 设置作者日期(D) ☐ 设置作者(T) 添加"Signed-off-by"(S)

变更列表 (双击文件查看差异):

选中: **全部(A)** 无(N) 未版本控制 已版本控制 已添加 已删除 已修改 文件 子模块

路径	扩展名	状态	添加行数	删除行数
<input checked="" type="checkbox"/> mytest.txt	.txt	已修改	3	0

☒ 显示未受版本控制的文件(U) 选择了 1 个文件, 共有 1 个文件  
☐ 不自动选择子模块 [查看补丁>>](#)

☐ 显示整个工程(W) ☐ 仅仅消息(M)

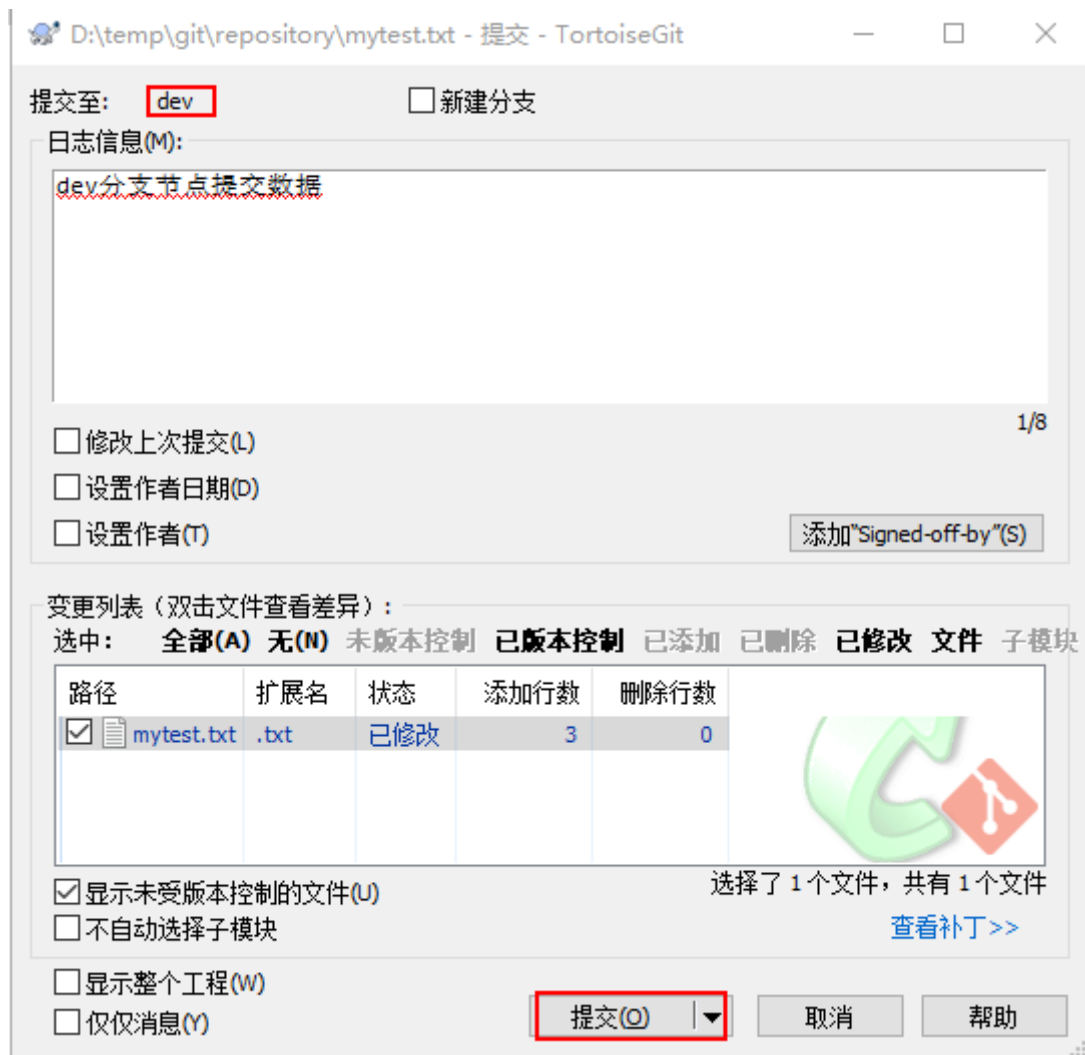
**提交(O)** |

1543398600322

将分支切换到dev上, 然后修改文件内容, 并提交







1543398716474

切换到master分支上, 将dev分支的内容合并过来

D:\temp\git\repository\mytest.txt - 提交 - TortoiseGit

提交至: 

dev

新建分支

日志信息(M):

dev分支节点提交数据

☐ 修改上次提交(L)

☐ 设置作者日期(D)

☐ 设置作者(T)

添加"Signed-off-by"(S)

1/8

变更列表 (双击文件查看差异):

选中: 

全部(A)

无(N)

未版本控制

已版本控制


已添加

已删除

已修改

文件

子模块

路径	扩展名	状态	添加行数	删除行数
<input checked="" type="checkbox"/>  mytest.txt	.txt	已修改	3	0
<div><div><input checked="" type="checkbox"/> 显示未受版本控制的文件(U)</div><div><input type="checkbox"/> 不自动选择子模块</div></div> <div>选择了 1 个文件，共有 1 个文件</div> <div><a href="#">查看补丁&gt;&gt;</a></div>				

☐ 显示整个工程(W)

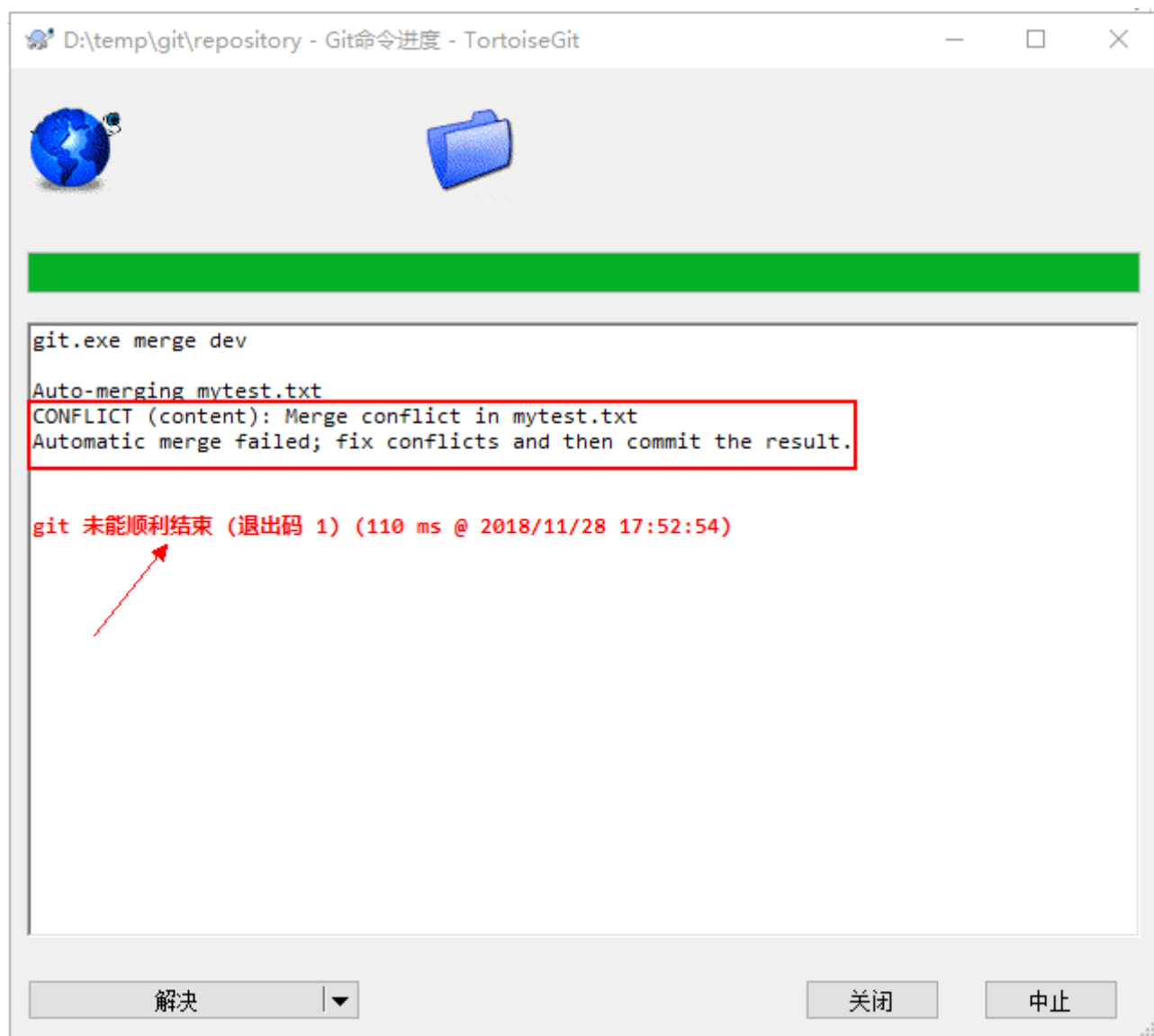
☐ 仅仅消息(Y)

提交(O)

取消

帮助

1543398770004



1543398829815

提示不是最新的版本，需要先解决冲突，在进行合并  
解决冲突:

名称	修改日期	类型	大小
 .git	2018/11/28 17:52	文件夹	
 HelloProjet	2018/11/28 13:38	文件夹	
 mytest.txt	2018/11/28 17:52	文本文档	1 KB

```

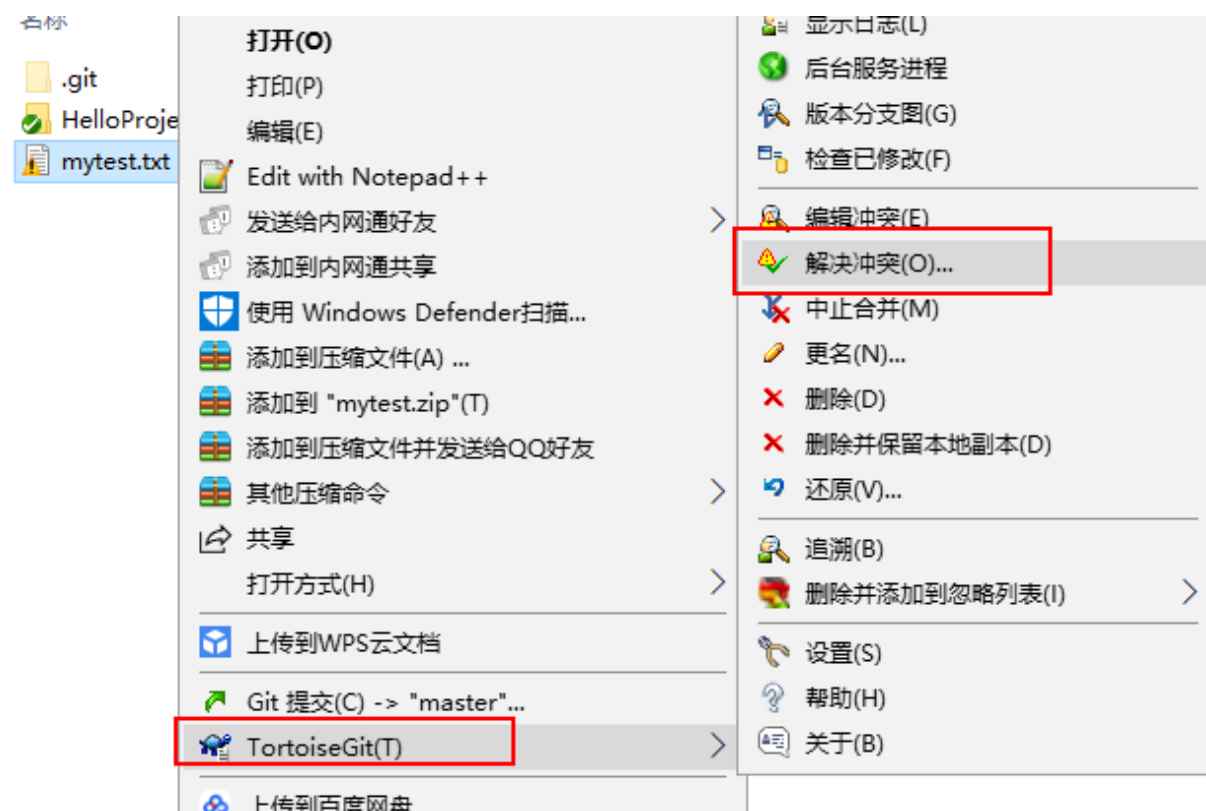
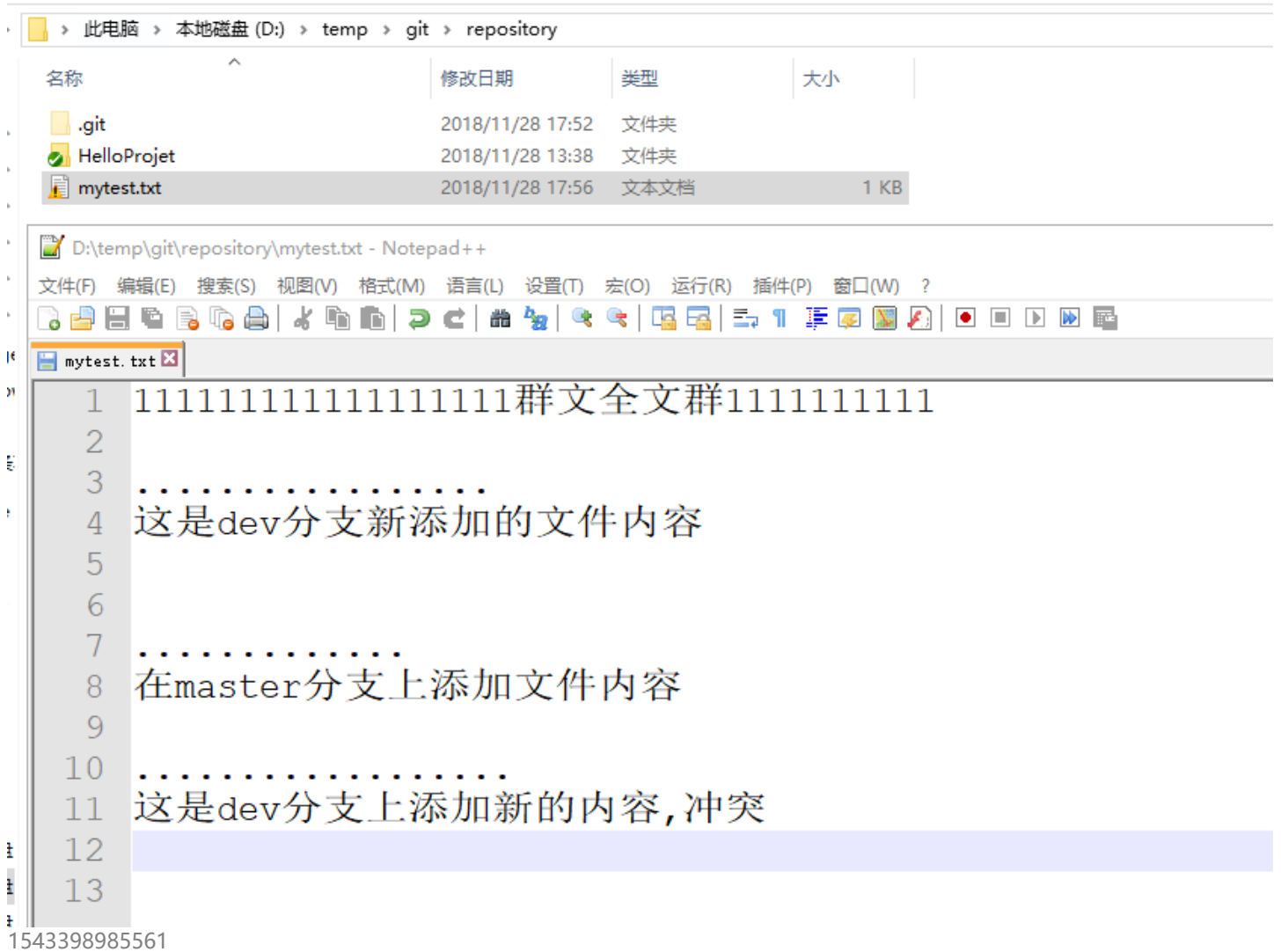
1 11111111111111111111群文全文群1111111111
2
3 .....
4 这是dev分支新添加的文件内容
5
6 <<<<<< HEAD
7 .....
8 在master分支上添加文件内容
9 =====
10 .....
11 这是dev分支上添加新的内容,冲突
12 >>>>>> dev
13

```

### 当前master节点的数据

## dev节点的数据

解决冲突必须手动解决，因为程序并不知道，对应的内容应该放置在什么位置，需要手动处理



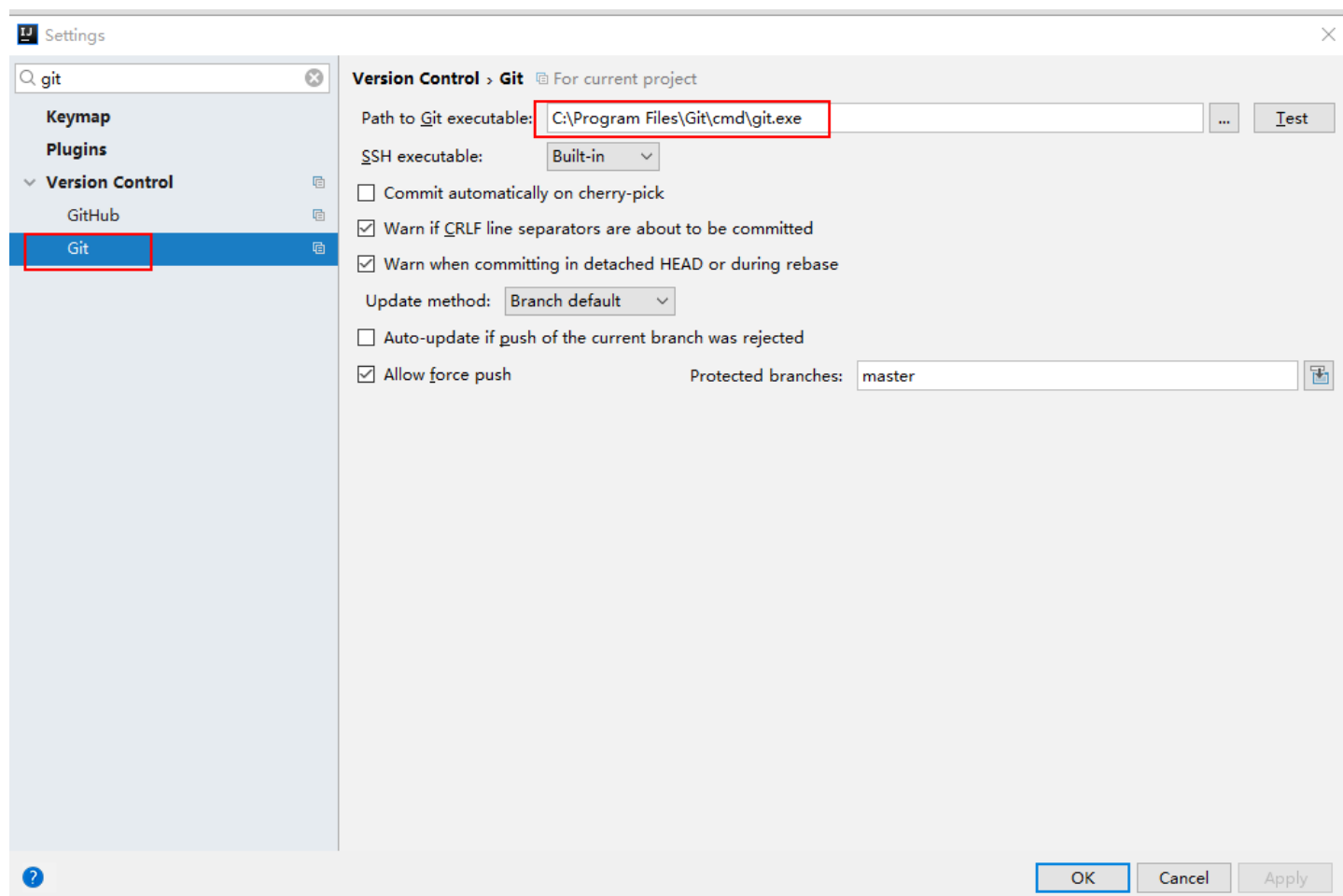
解决冲突完成后, 在重新提交即可合并

## 8. 在IDEA中使用git

### 8.1 在 idea中配置git

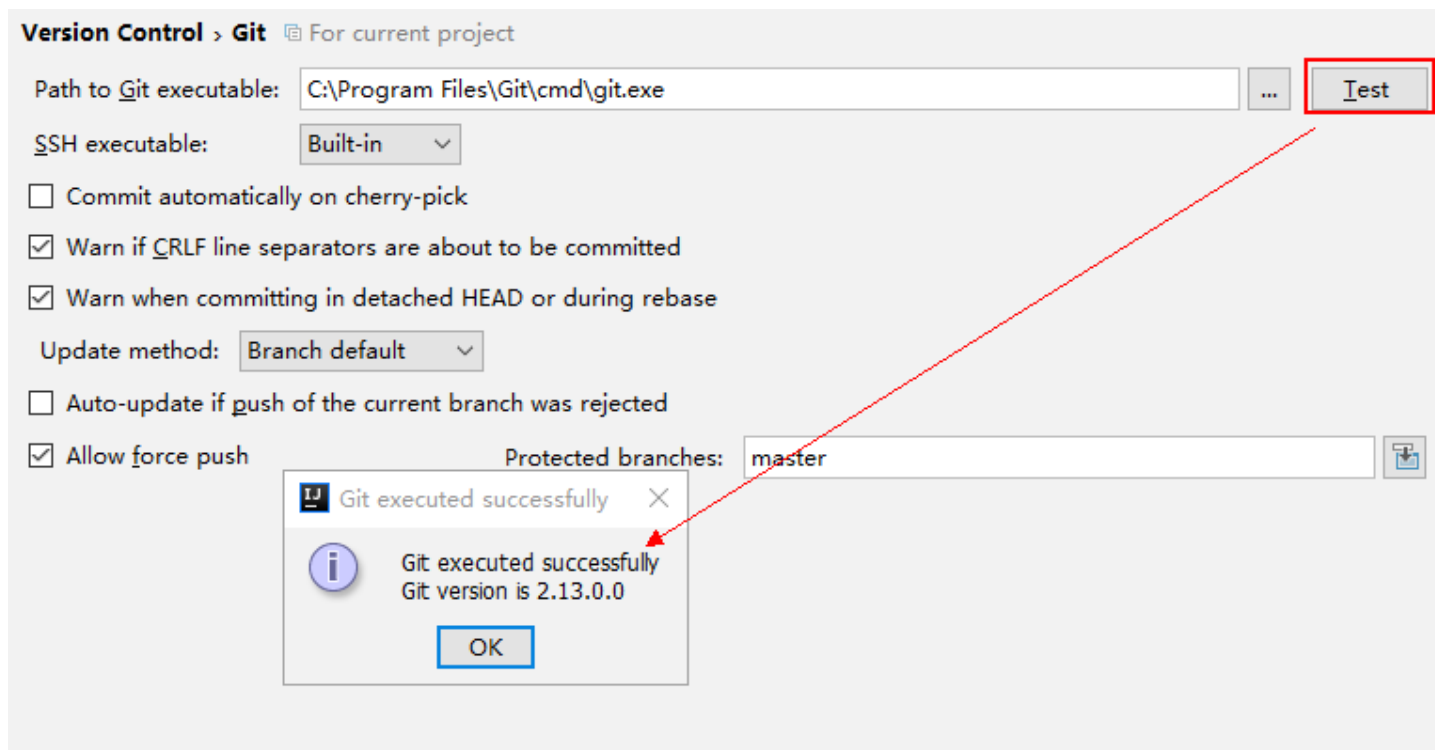
安装好IntelliJ IDEA后, 如果Git安装在默认路径下, 那么idea会自动找到git的位置, 如果更改了Git的安装位置则需要手动配置下Git的路径。

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



1543399359546

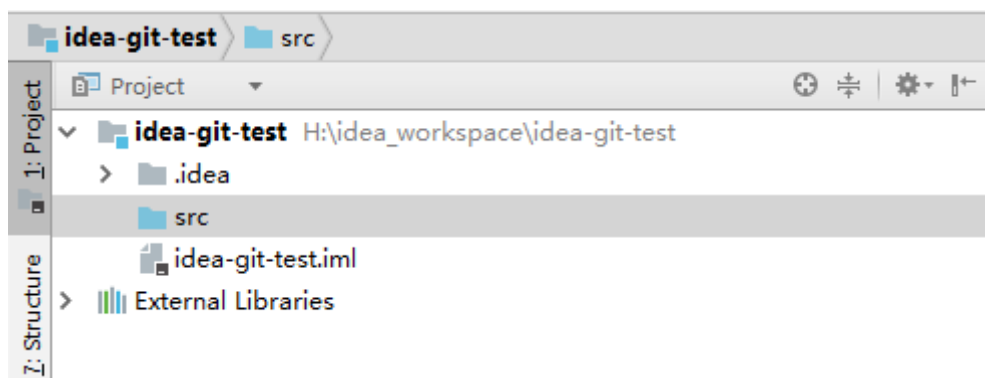
选择git的安装目录后可以点击 “Test” 按钮测试是否正确配置。



1543399391526

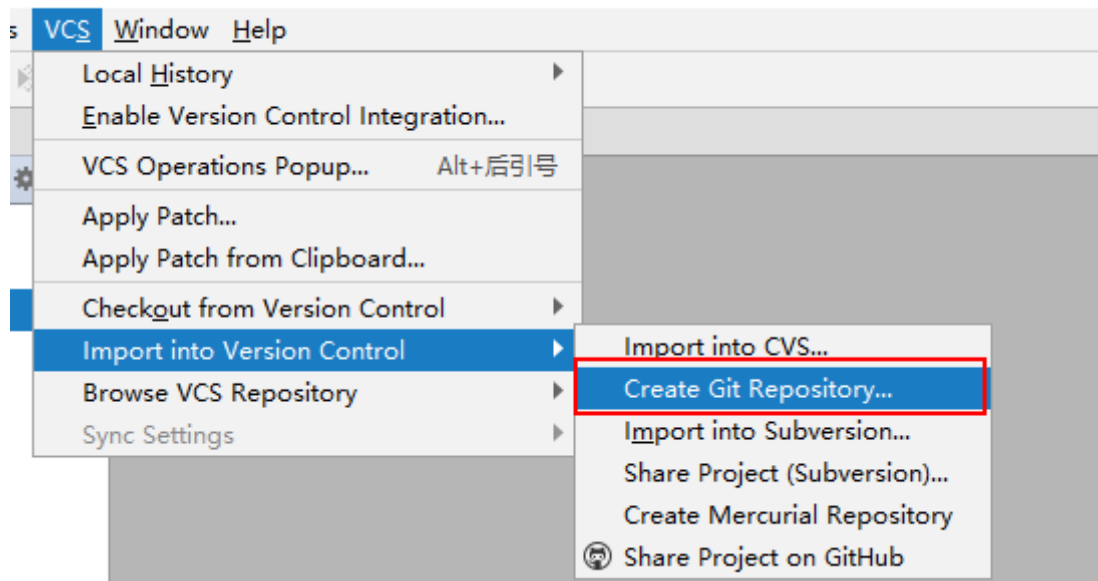
## 8.2 将工程添加到git

- 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

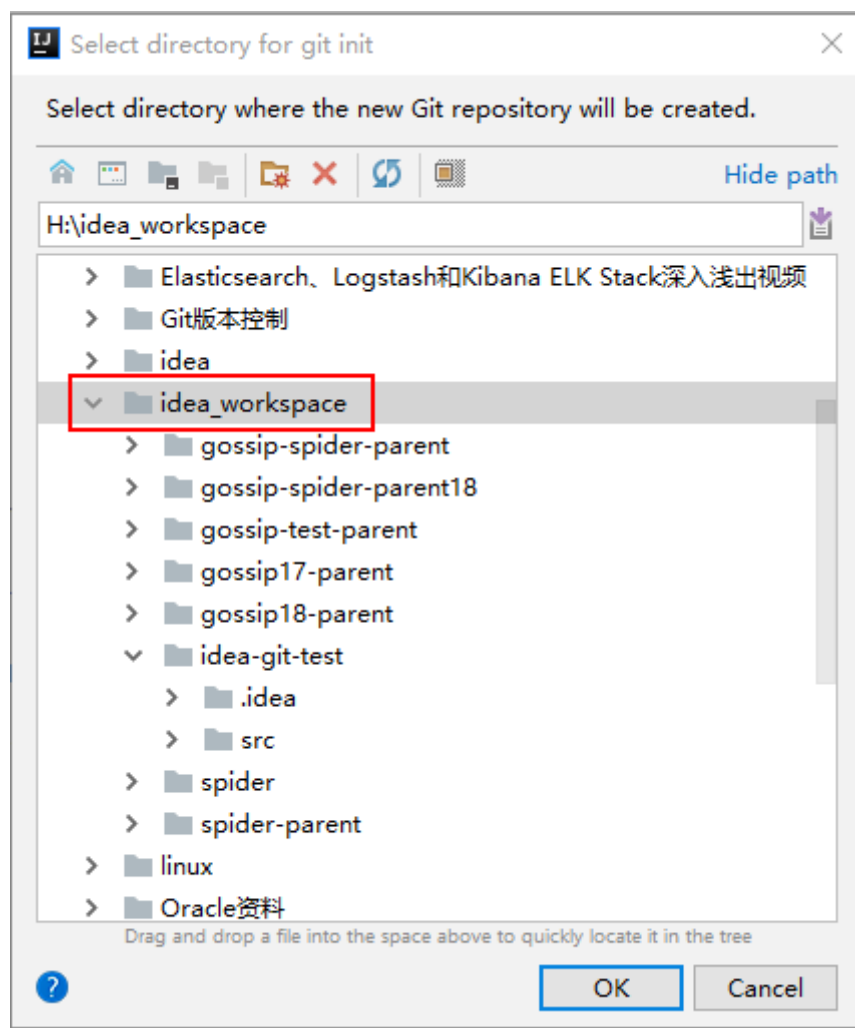


1543399570808

- 2) 创建本地仓库
  - 在菜单中选择“vcs”→Import into Version Control→Create Git Repository...



1543399681865



1543399818615

- 1 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。

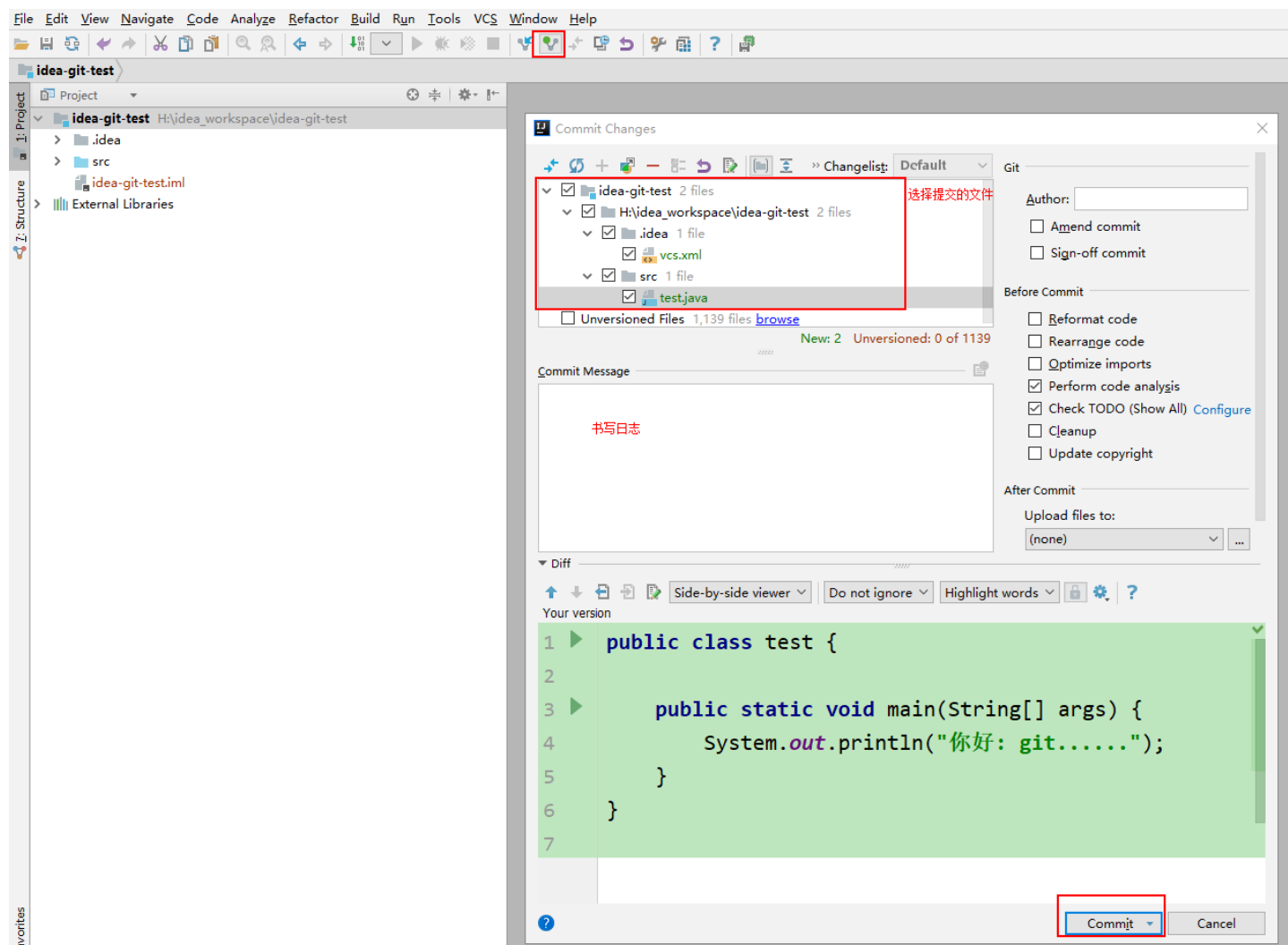


选择之后在工具栏上就多出了git相关工具按钮：



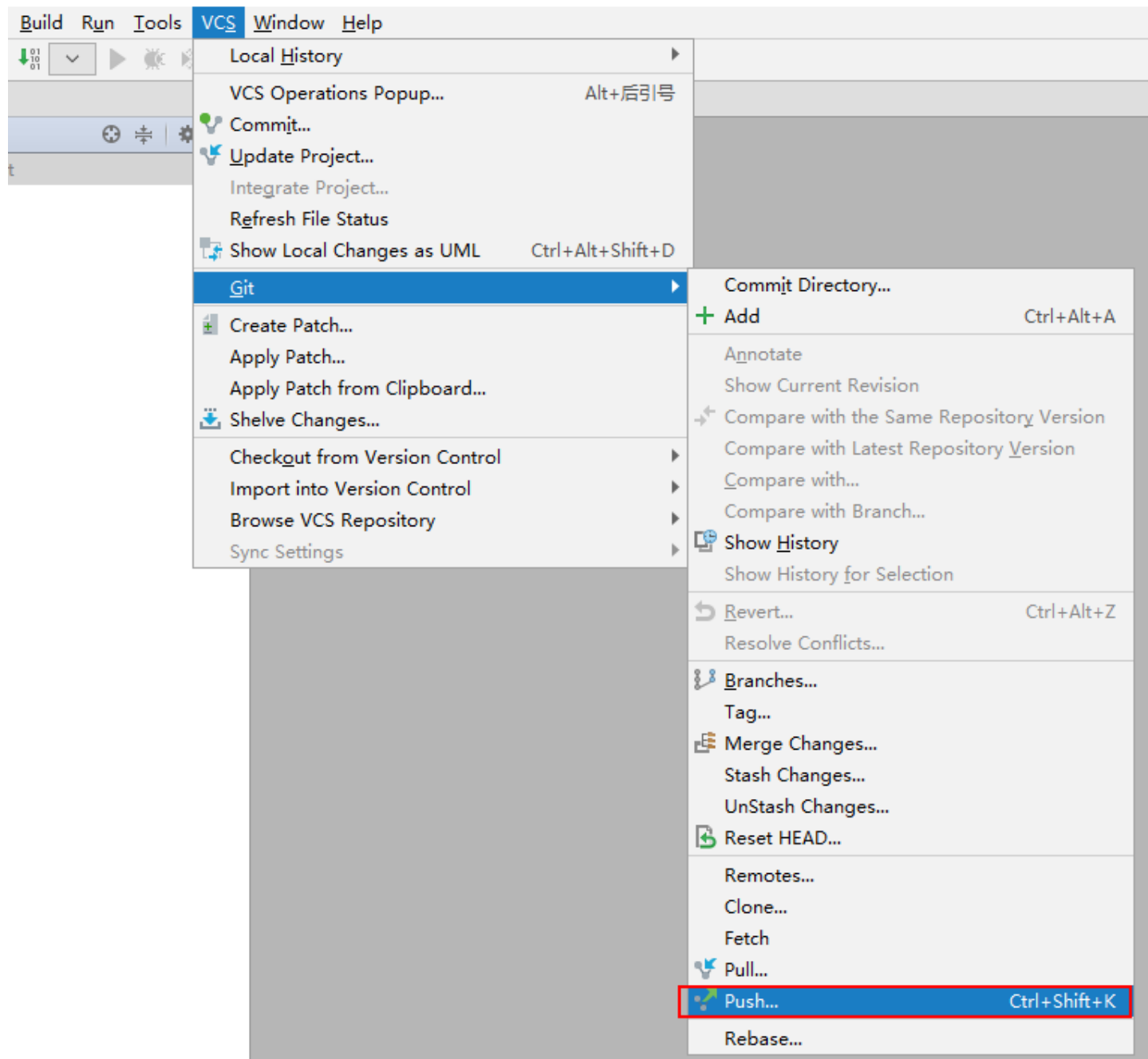
1543399982814

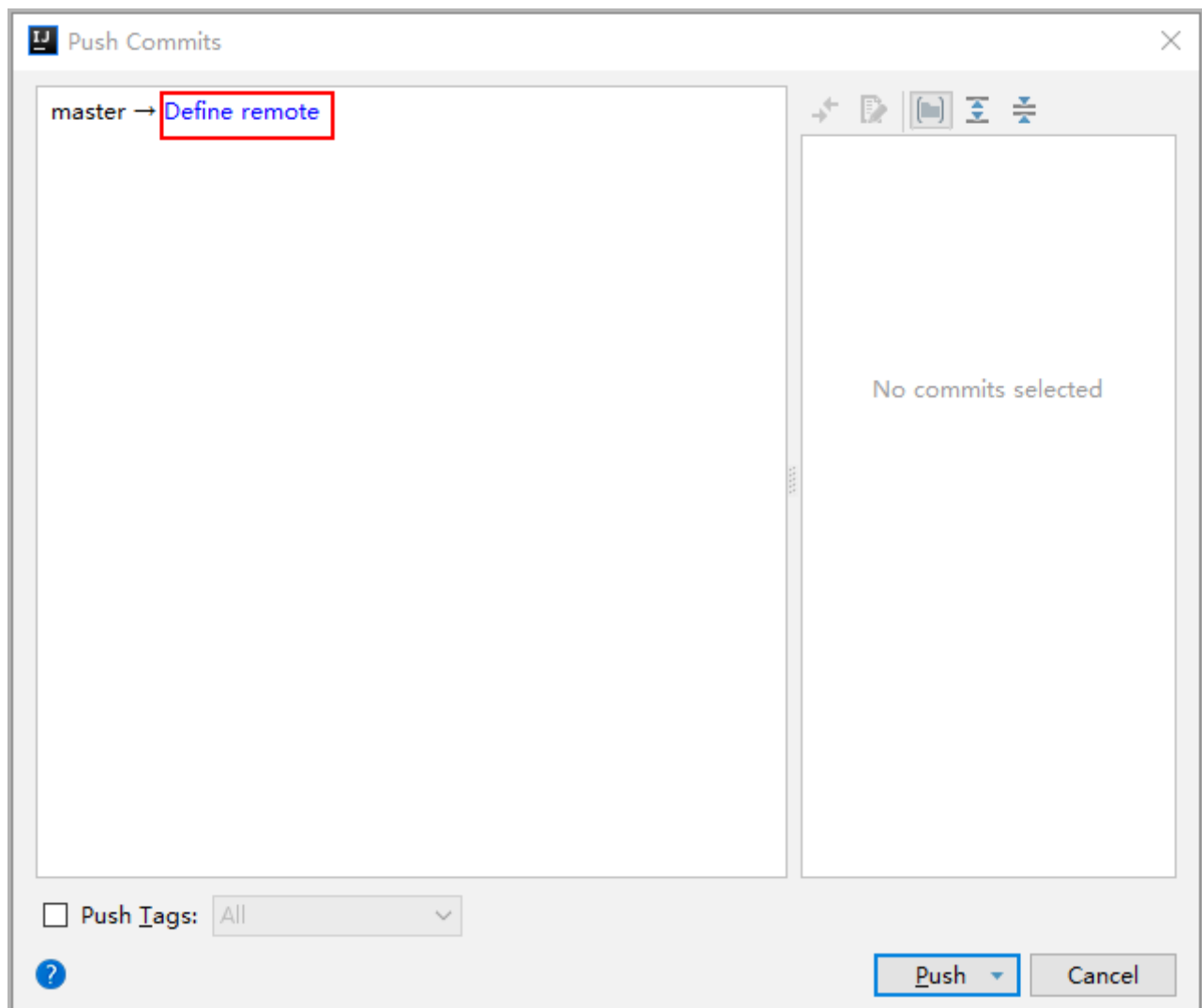
- 将其添加到本地版本库中: 点击commit即可提交到本地的版本库中



1543400525142

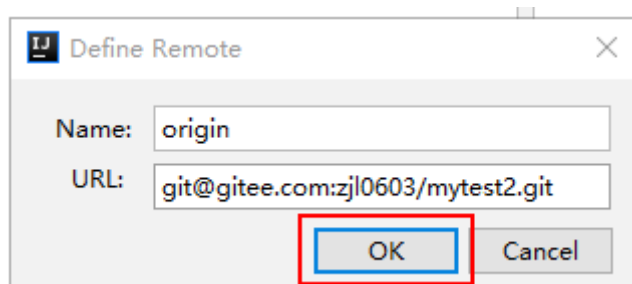
- 推送至远程在码云上创建一个仓库然后将本地仓库推送到远程。在工程上点击右键，选择  
git→Repository→push，或者在菜单中选择vcs→git→push



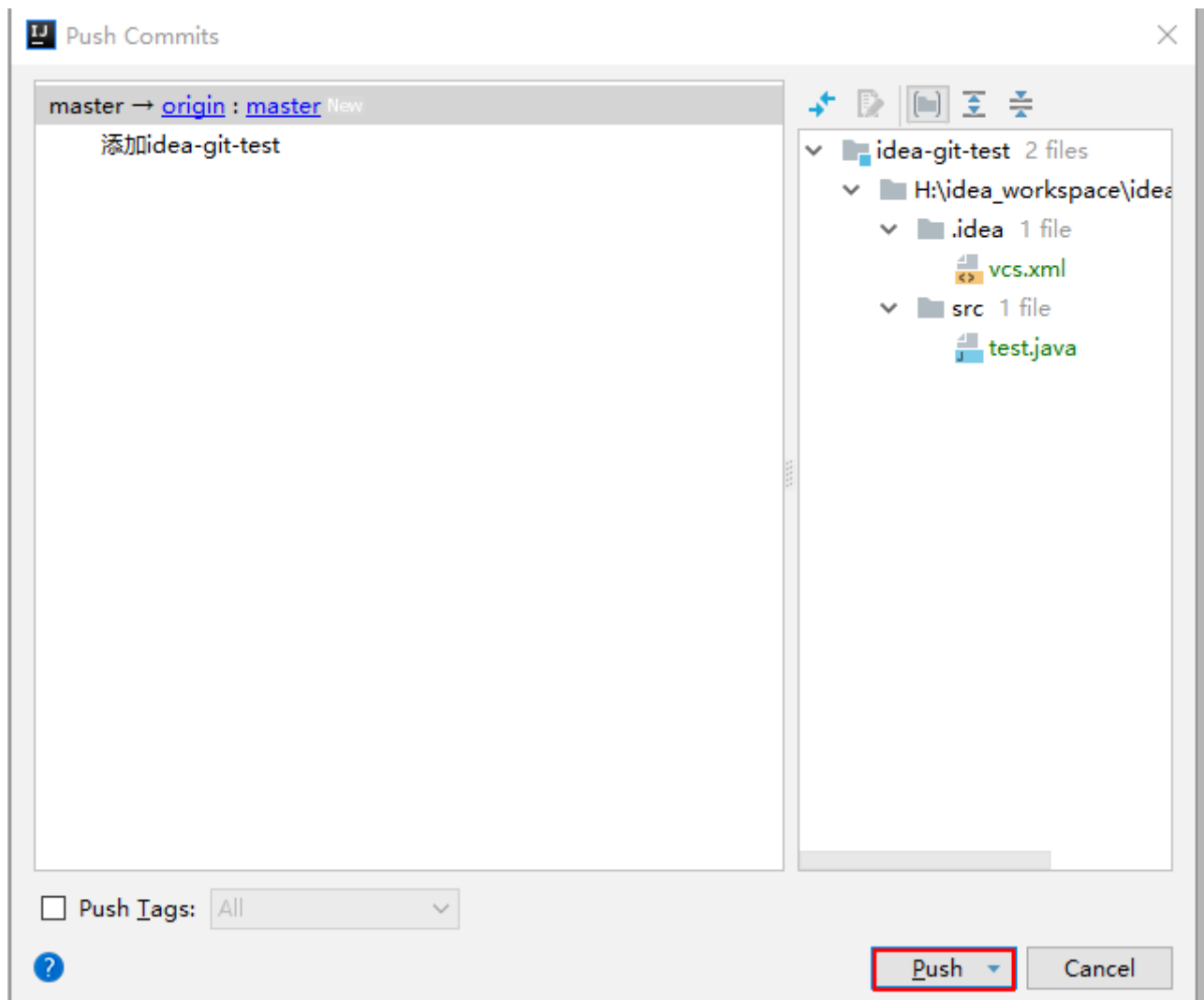


1543400828571

选择Define remote

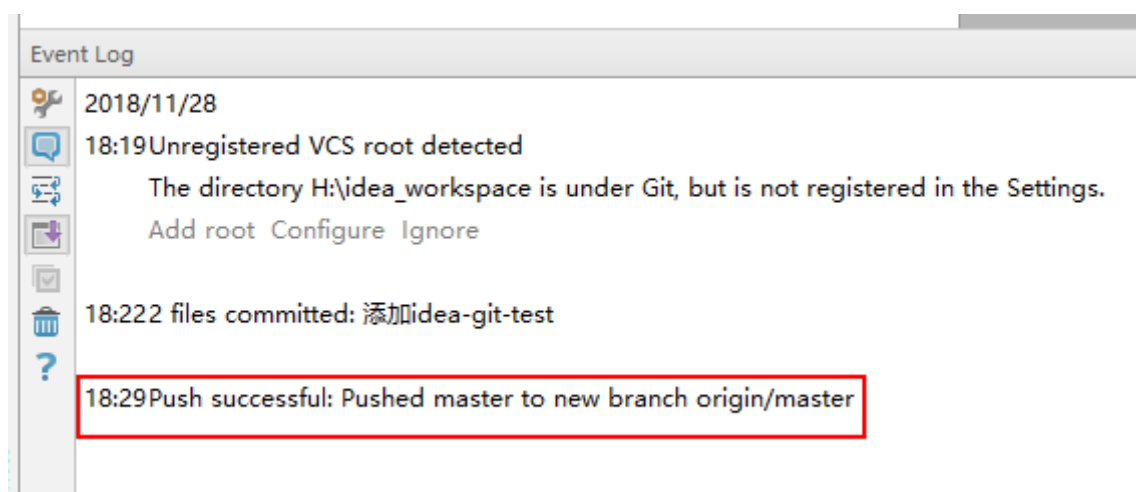


1543400956443



1543400987266

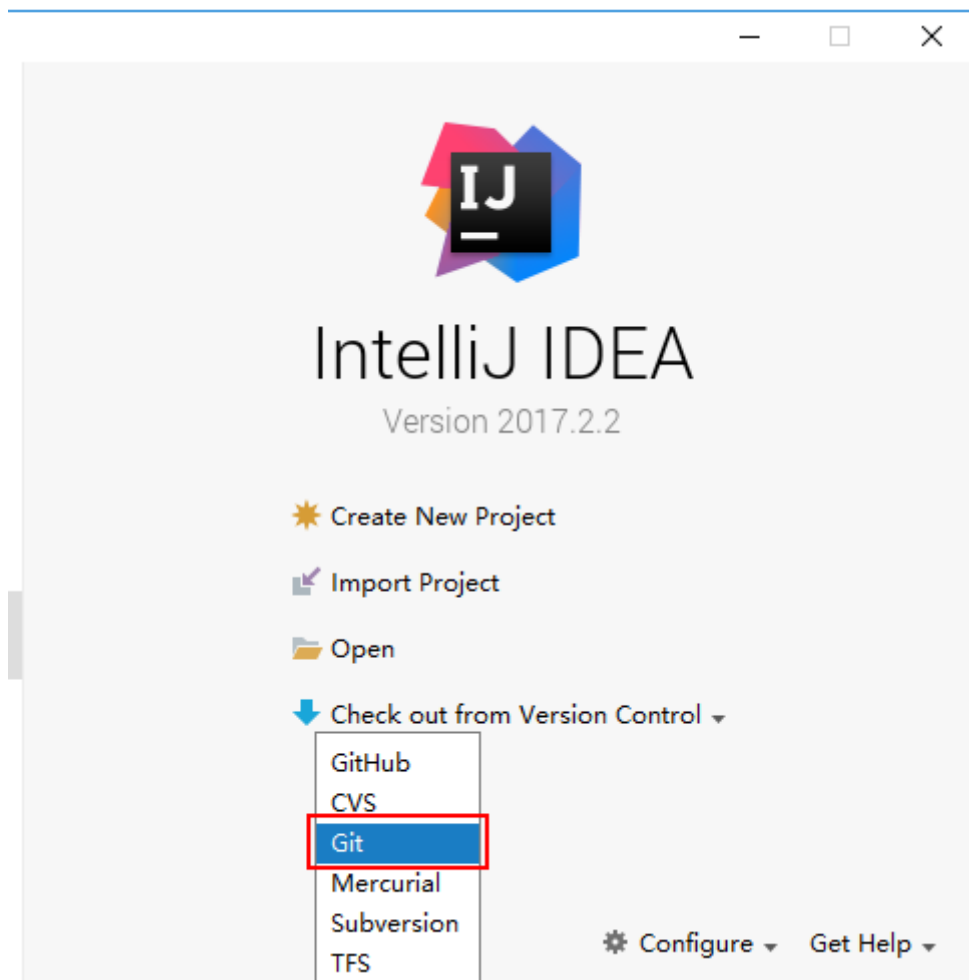
成功后, idea会显示



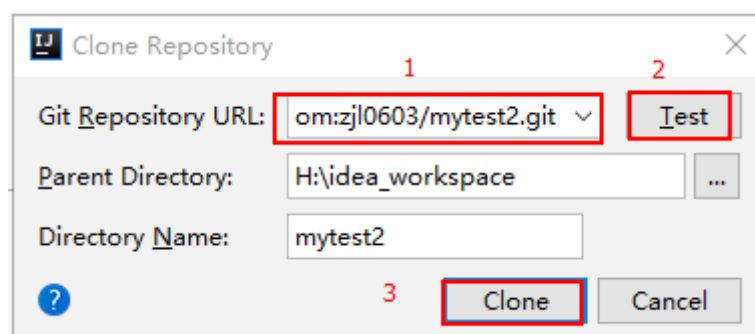
43401045702

## 8.3 从远程仓库克隆

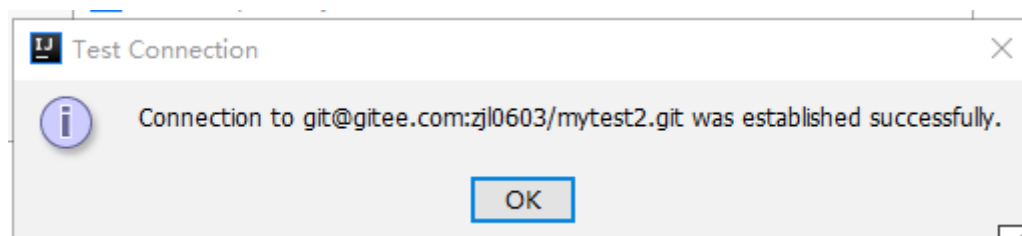
关闭工程后, 在idea的欢迎页上有 “Check out from version control” 下拉框, 选择git



1543401467964

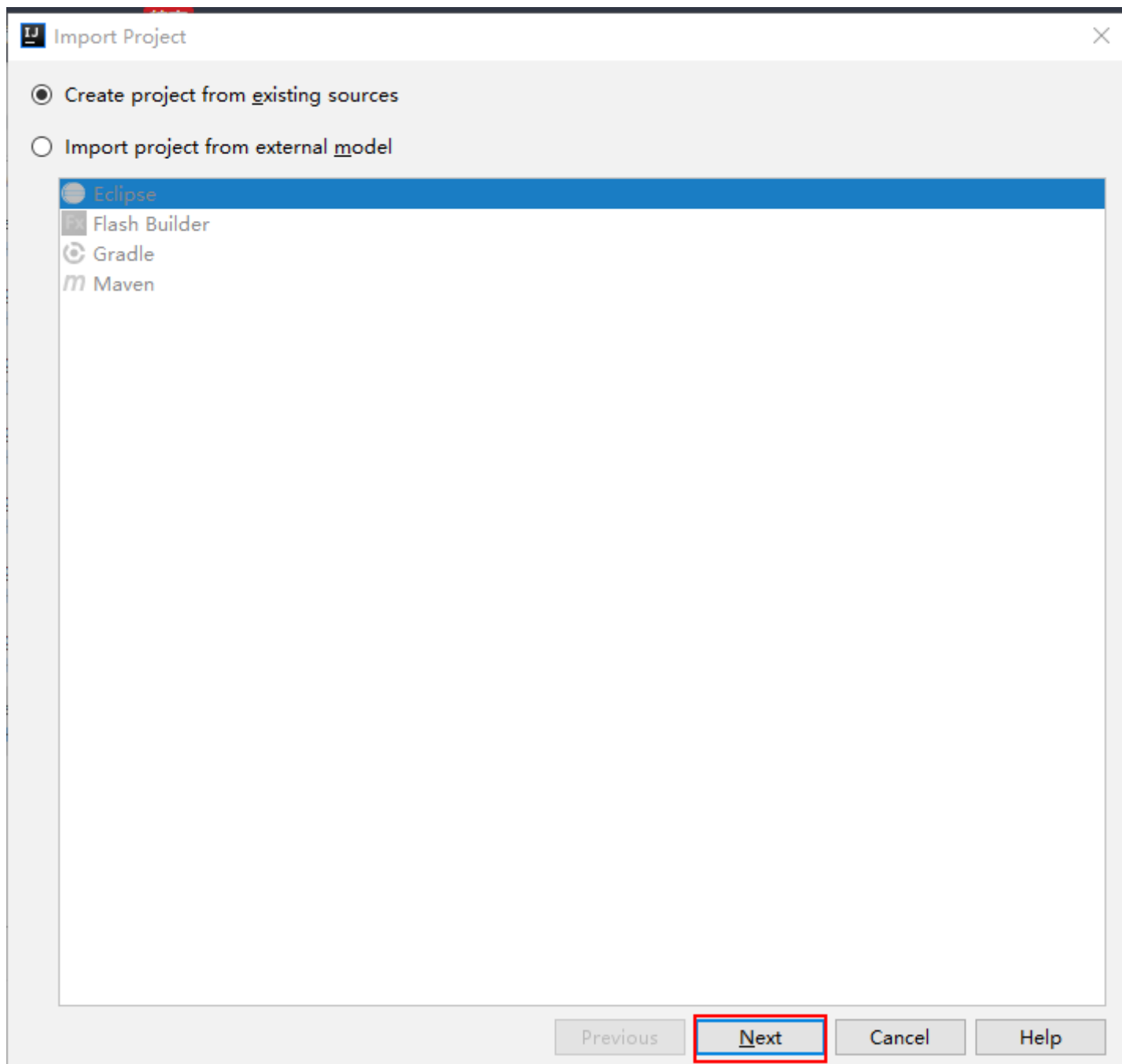


1543401508478



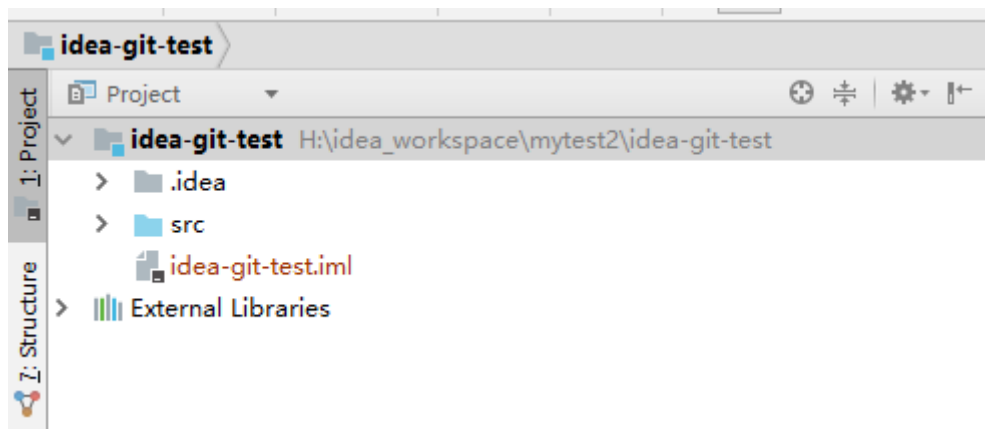
1543401651635

使用idea选择克隆后，会出现如下内容，一直下一步即可



1543401862519

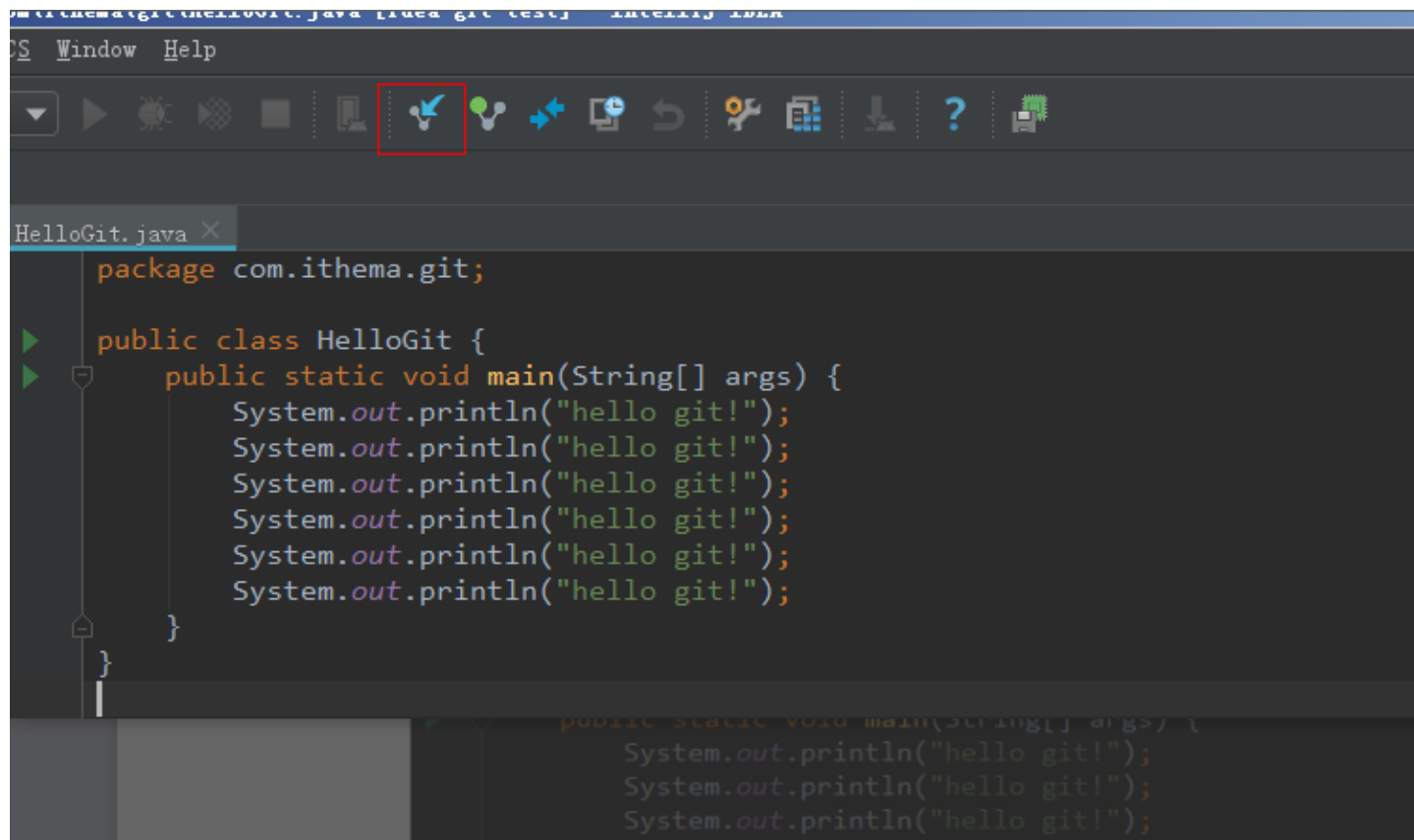
此时就又回来了



1543401893387

## 8.4 从远程拉取代码

如果需要从服务端同步代码可以使用工具条中的“update”按钮



1543401992188