





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）



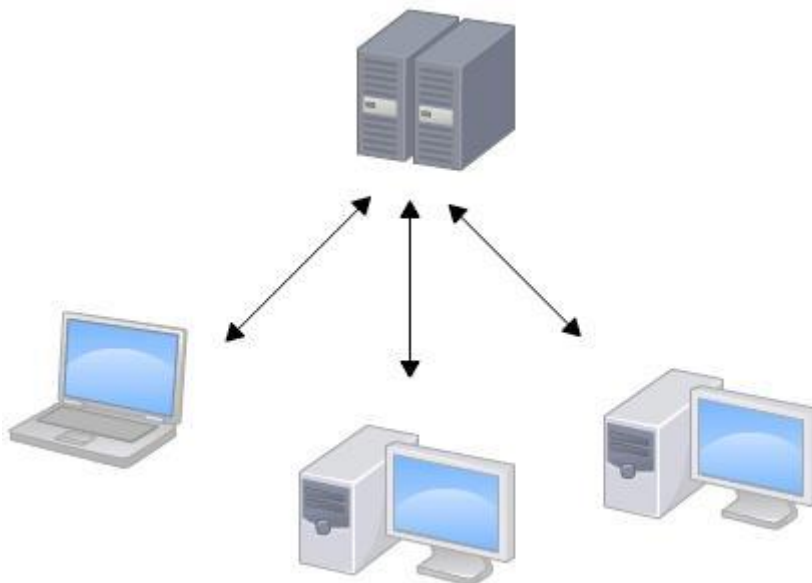
2 Git 与 svn 对比

2.1 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。

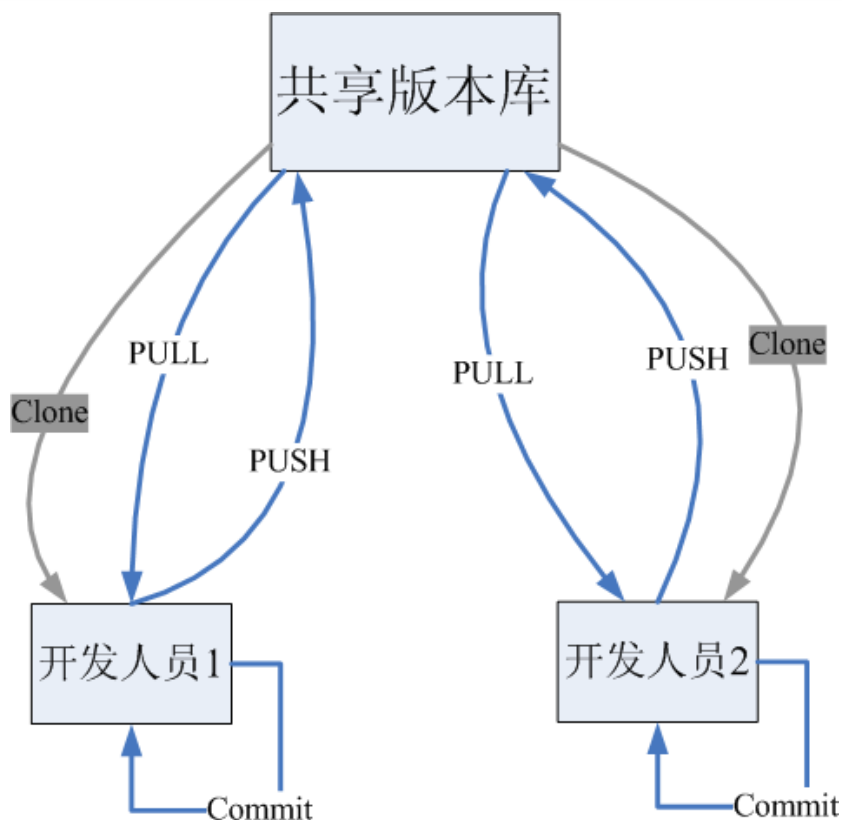
但是相较于其优点而言，集中式版本控制工具缺点很明显：

- 服务器单点故障
- 容错性差

2.2 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：



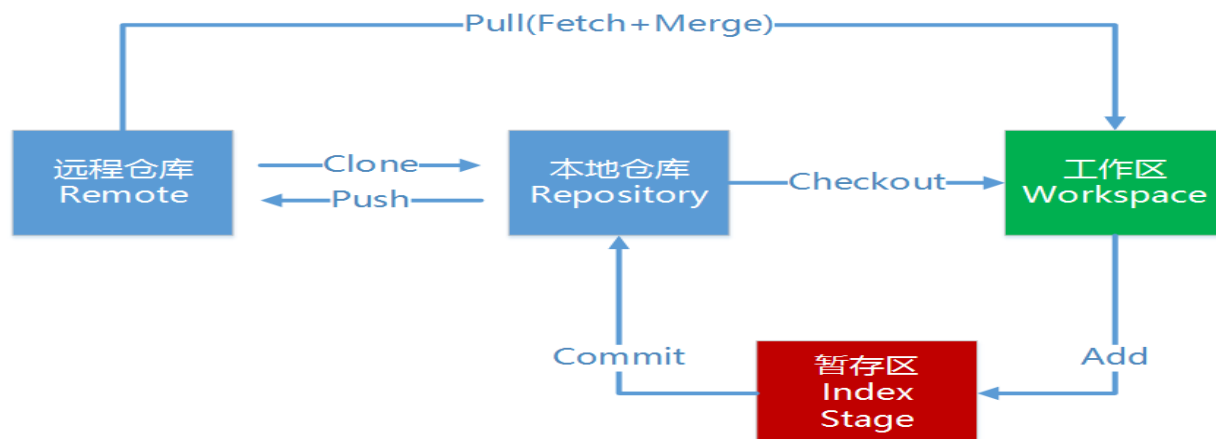
3 git 工作流程

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

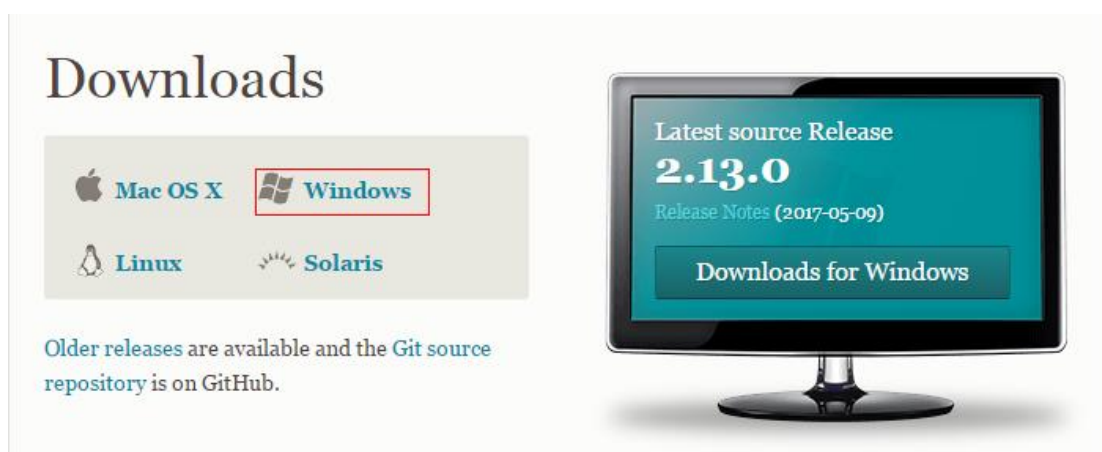


4 Git 的安装

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 这四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>

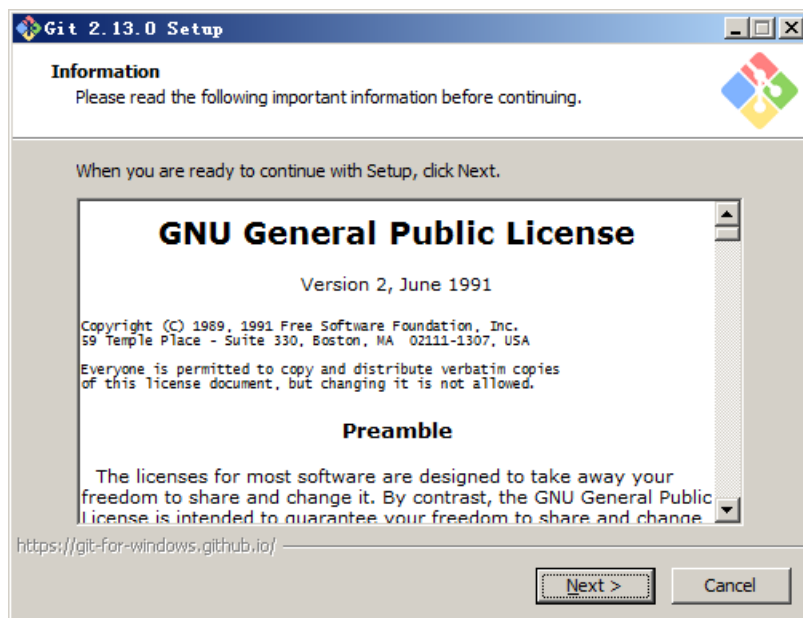




参考资料中安装包已经下载完毕，根据不同的操作系统选择对应的安装包。

4.2 软件安装

4.2.1 安装 git for windows

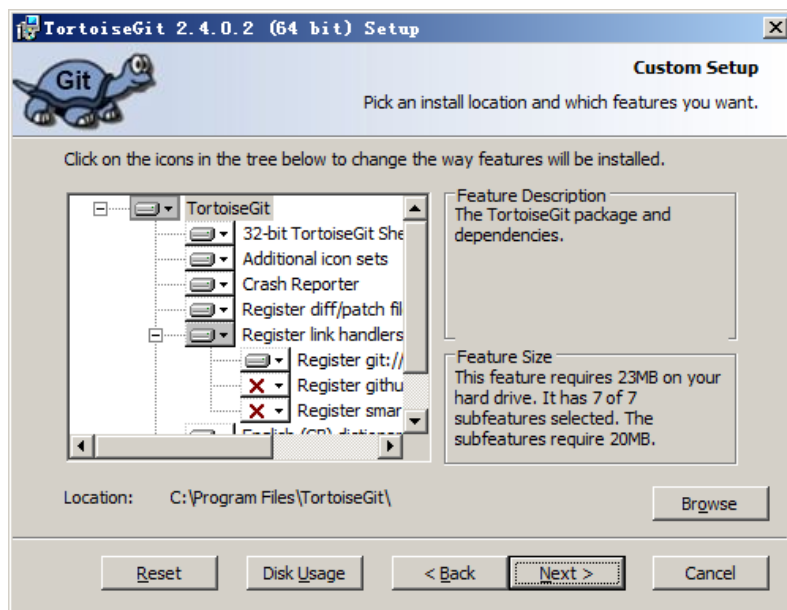


一路“下一步”使用默认选项即可。



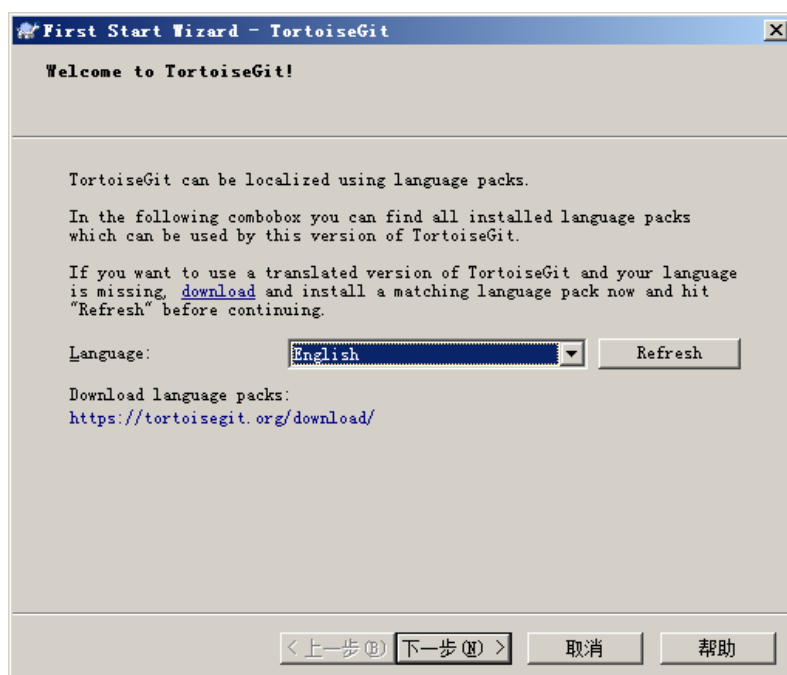
4.2.2 安装 TortoiseGit

名称 ^	修改日期	类型	大小
Git-2.13.0-64-bit.exe	2017/5/17 17:26	应用程序	36,286 KB
TortoiseGit-2.4.0.2-64bit.msi	2017/5/17 17:11	Windows Instal...	19,520 KB
TortoiseGit-LanguagePack-2.4.0.0-6...	2017/5/17 17:19	Windows Instal...	2,996 KB



一路“下一步”使用默认选项即可。

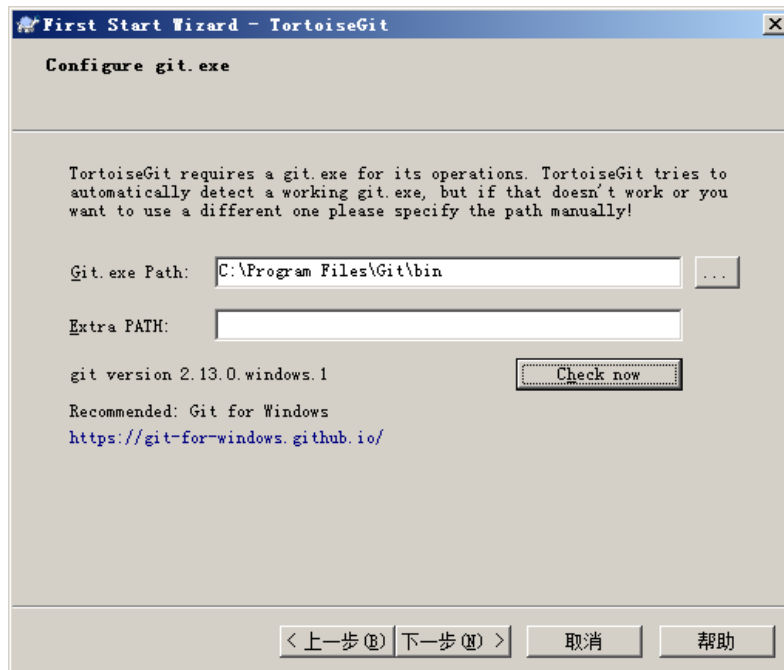
默认选项下会启动配置画面：



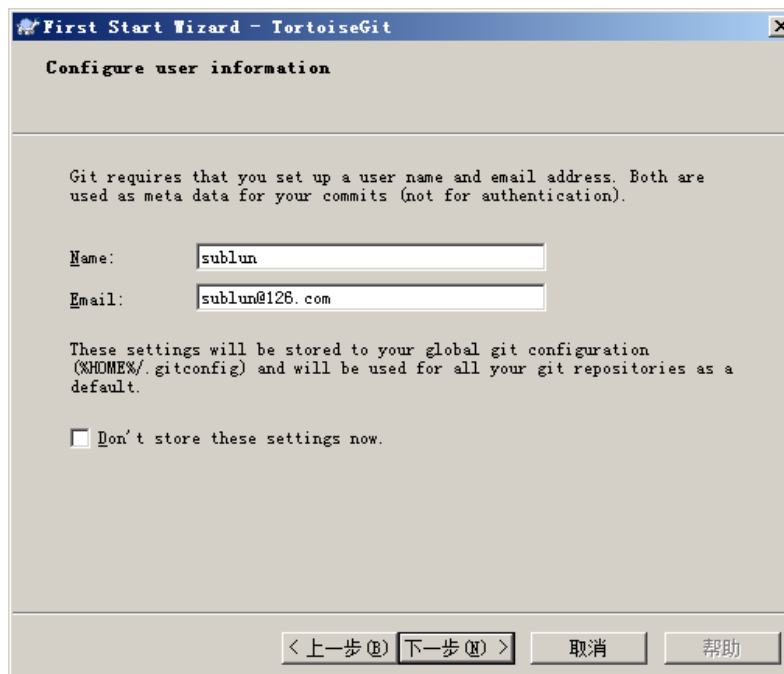


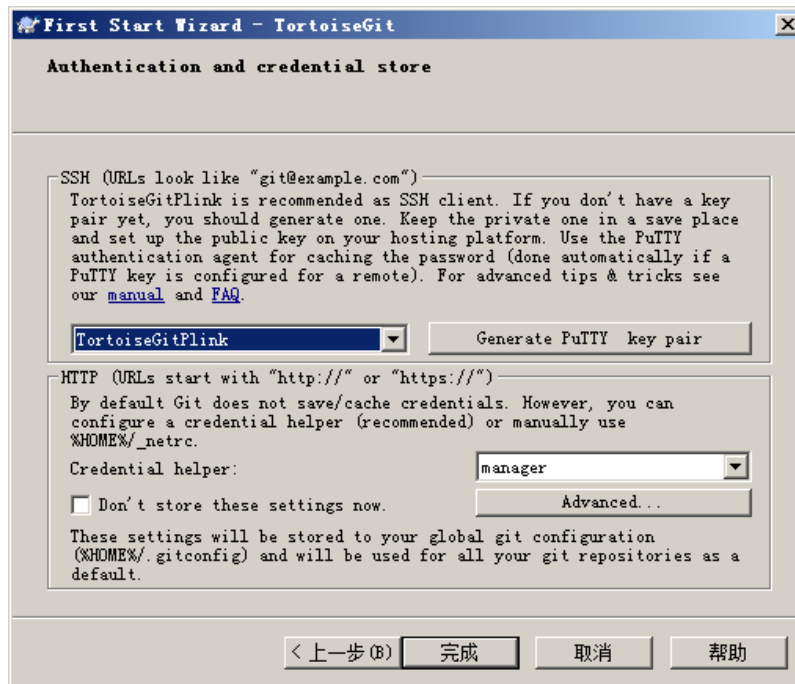
由于目前只有英文语言包，默认即可继续下一步。

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。



4.2.3 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。

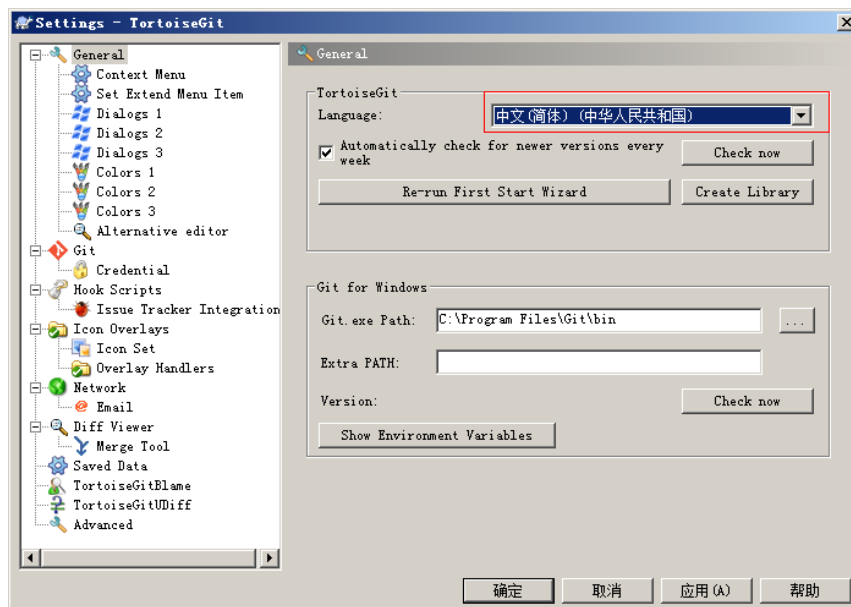


名称	左边距	修改日期	类型	大小
Git-2.13.0-64-bit.exe		2017/5/17 17:26	应用程序	36,286 KB
TortoiseGit-2.4.0.2-64bit.msi		2017/5/17 17:11	Windows Instal...	19,520 KB
TortoiseGit-LanguagePack-2.4.0.0-6...		2017/5/17 17:19	Windows Instal...	2,996 KB



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





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）。

5.1.1 使用 GitBash


在当前目录中点击右键中选择 Git Bash 来启动。





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

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。



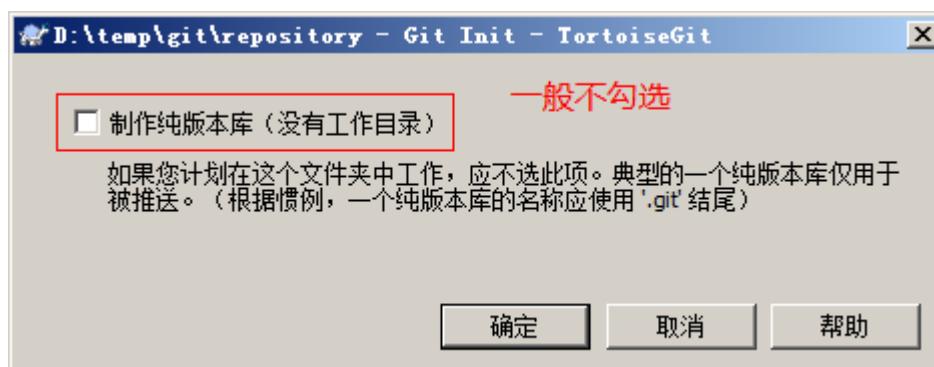
```
Administrator@PC-201311301552 MINGW64 /
$ cd /d/temp/git/repository/
Administrator@PC-201311301552 MINGW64 /d/temp/git/repository (master)
$
```

创建仓库执行命令：

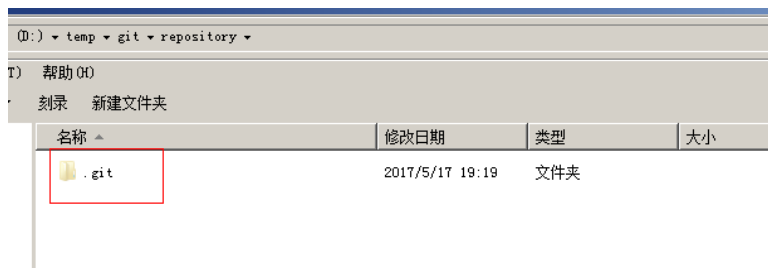
\$ git init

5.1.2 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。

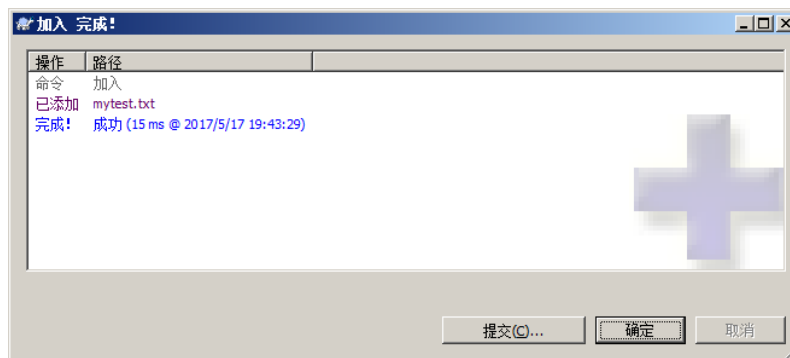
工作目录：包含“.git”目录的目录，也就是.git目录的上一级目录就是工作目录。只有工作目录中的文件才能保存到版本库中。

5.2 添加文件

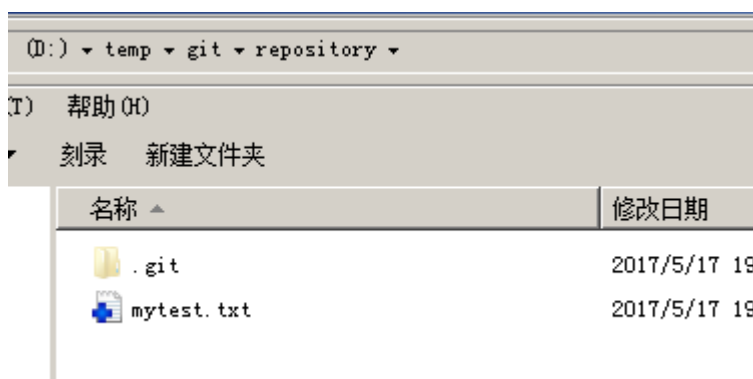
5.2.1 添加文件过程

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

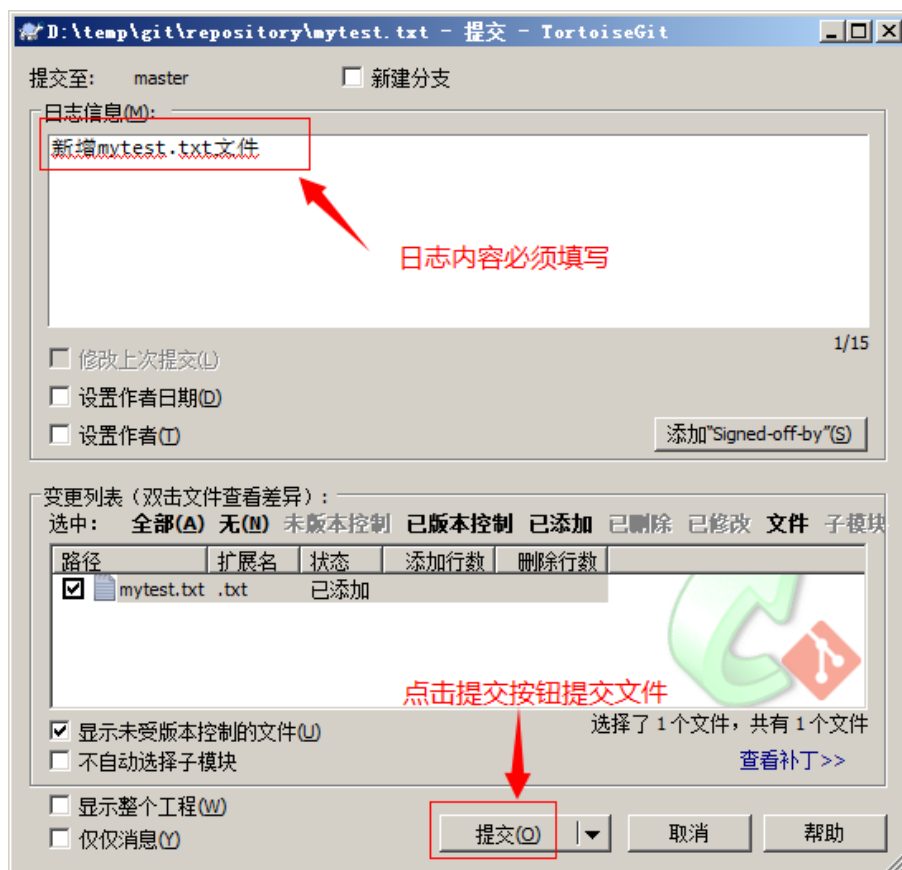
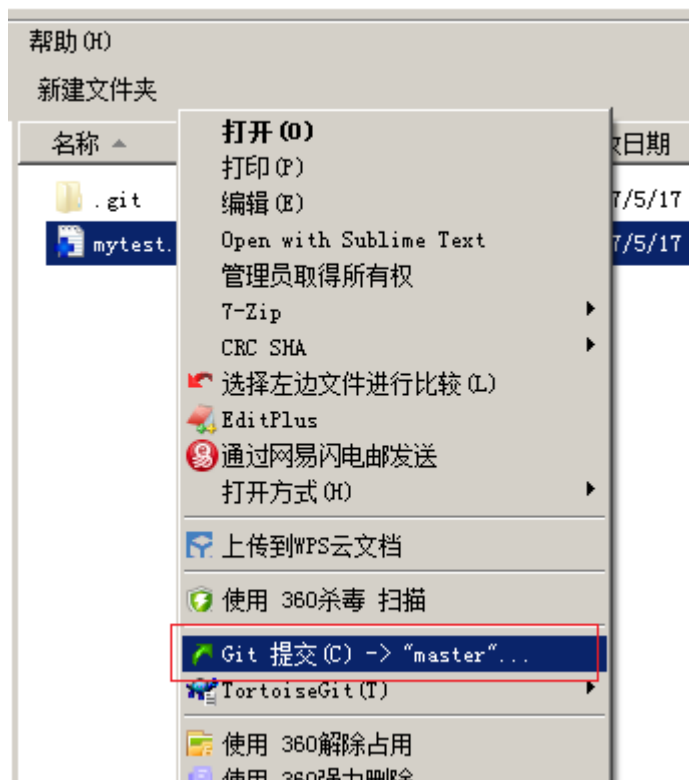


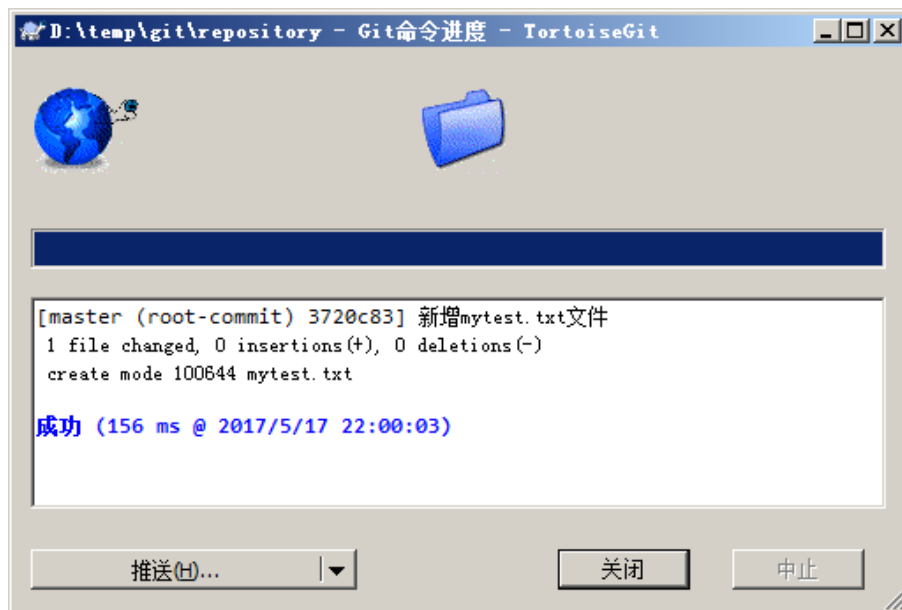


文本文件变为带“+”号的图标：



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。





5.2.2 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。

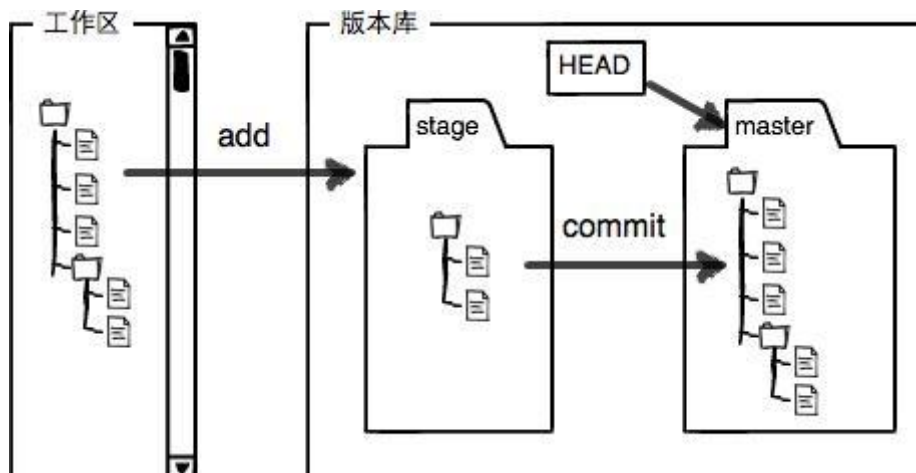
什么是工作区（Working Directory）？

工作区就是你在电脑里能看到的目录，比如我的 repository 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。

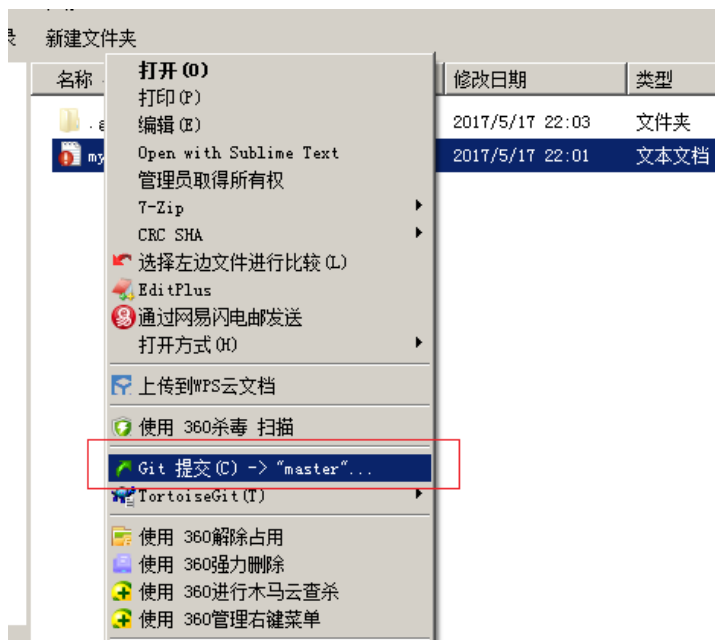
你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

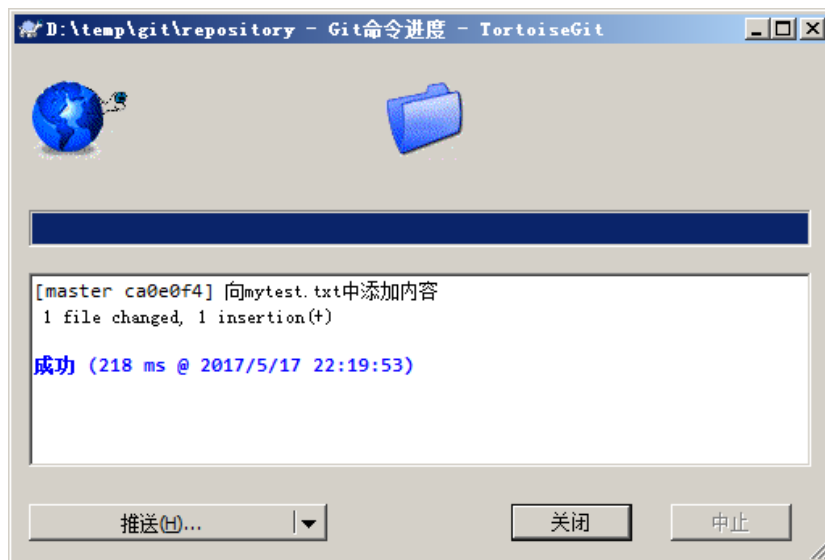
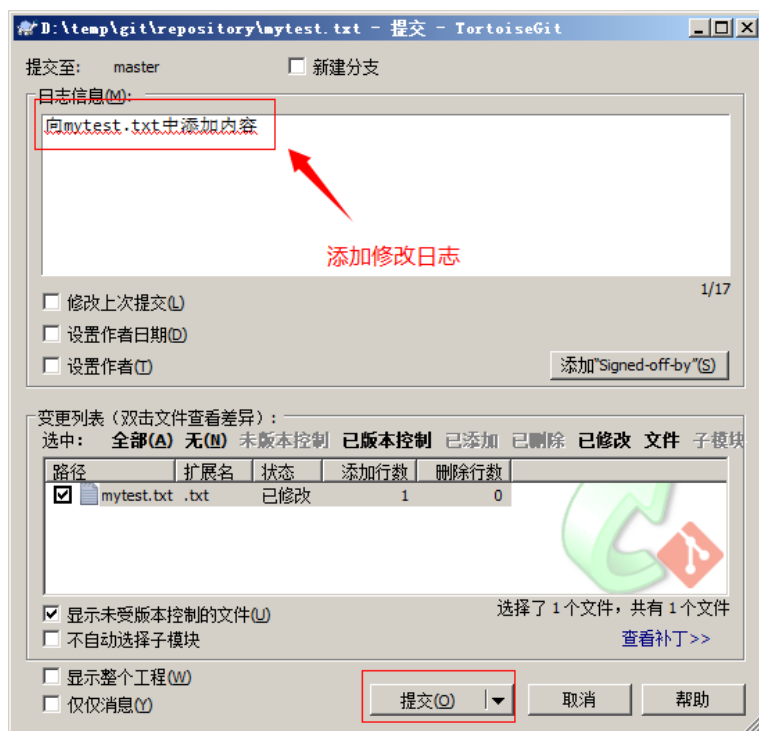
5.3 修改文件

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”

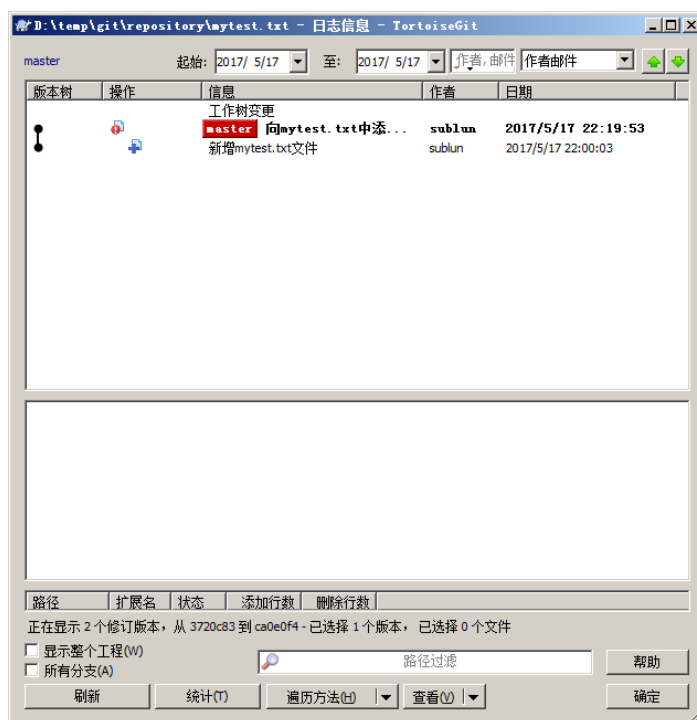
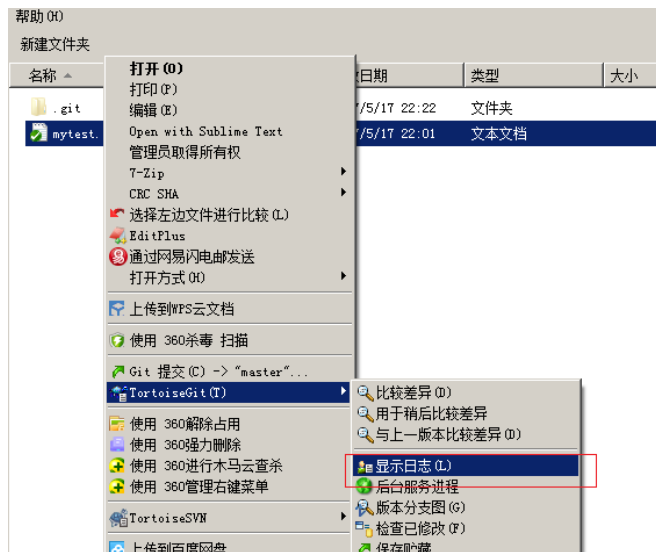




5.3.2 查看修改历史

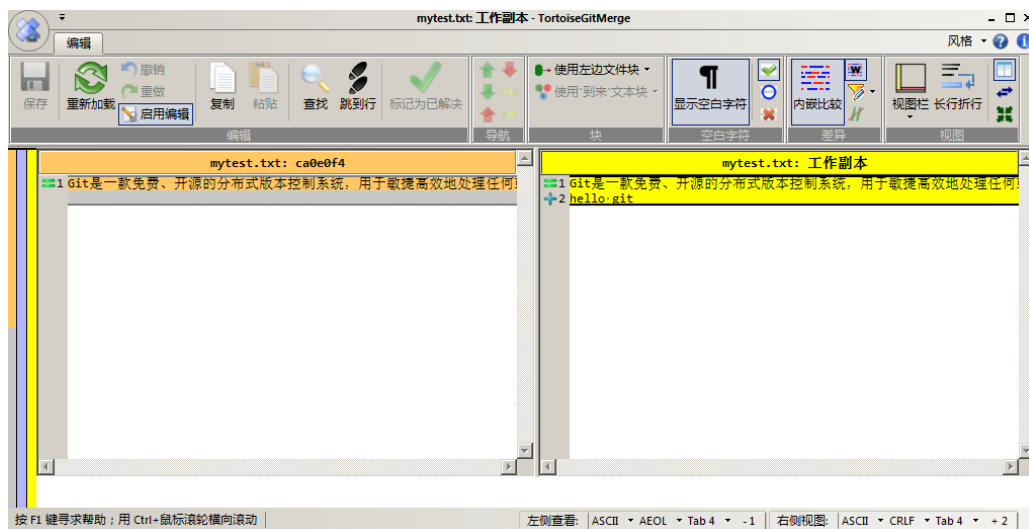
在开发过程中可能会经常查看代码的修改历史，或者叫做修改日志。来查看某个版本是谁修改的，什么时间修改的，修改了哪些内容。

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。



5.3.3 差异比较

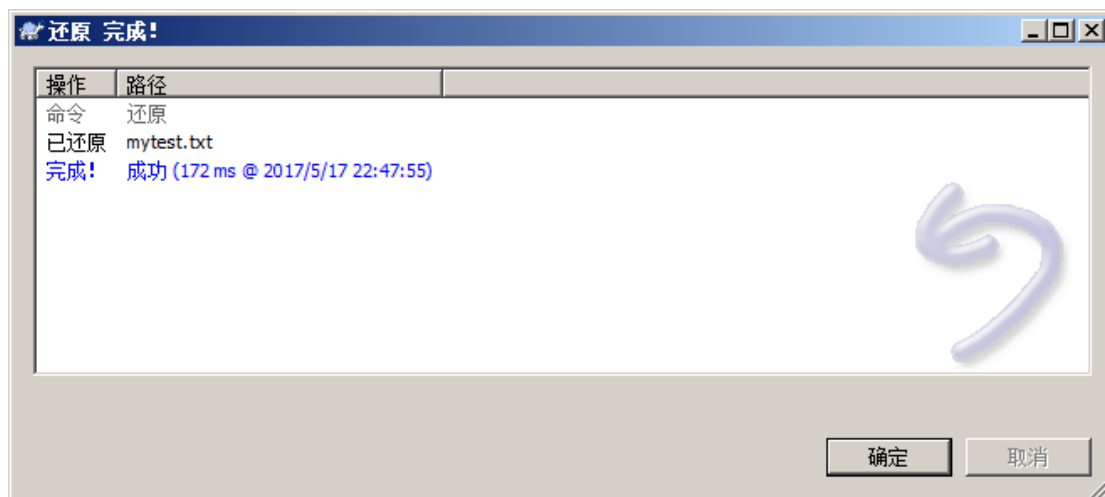
当文件内容修改后, 需要和修改之前对比一下修改了哪些内容此时可以使用“比较差异功能”



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

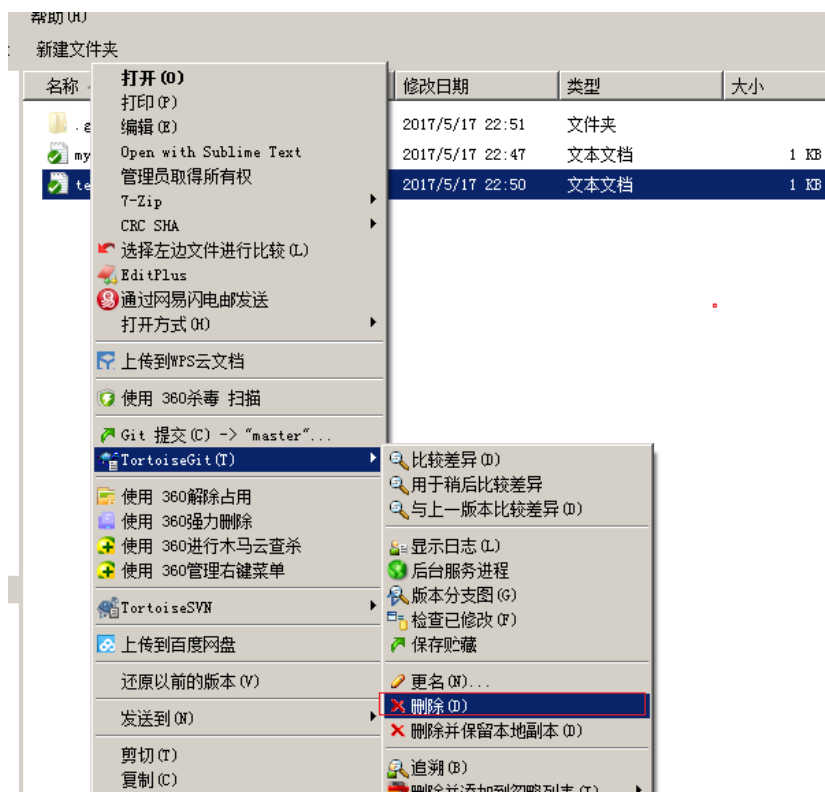




注意：此操作会撤销所有未提交的修改，所以当做还原操作是需要慎重慎重！！

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。



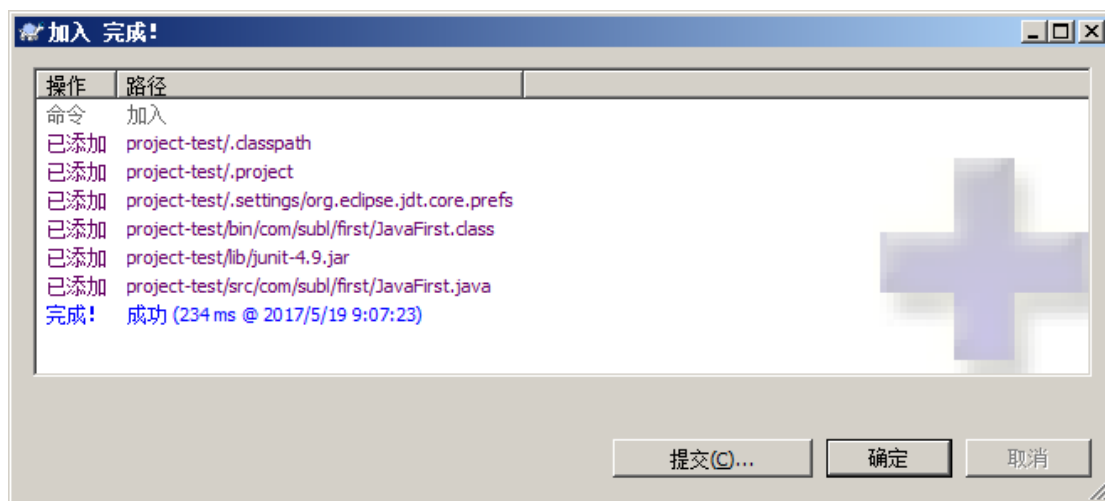
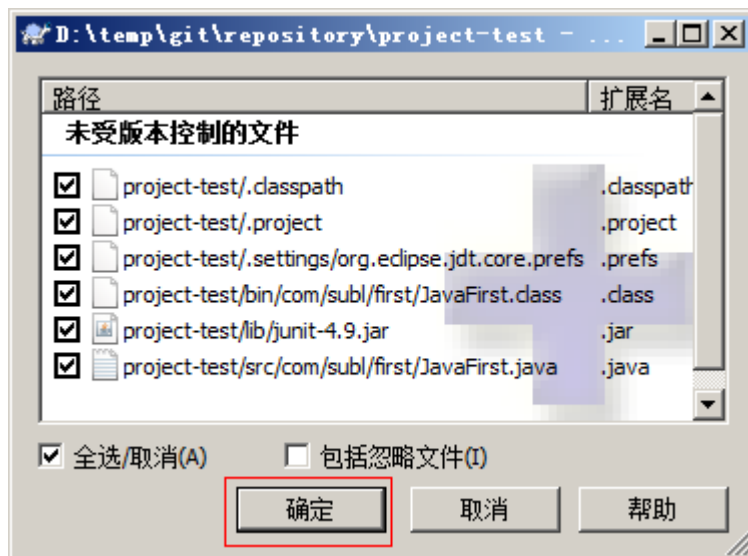
5.5 案例：将 java 工程提交到版本库

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



第二步：将工程添加到暂存区。



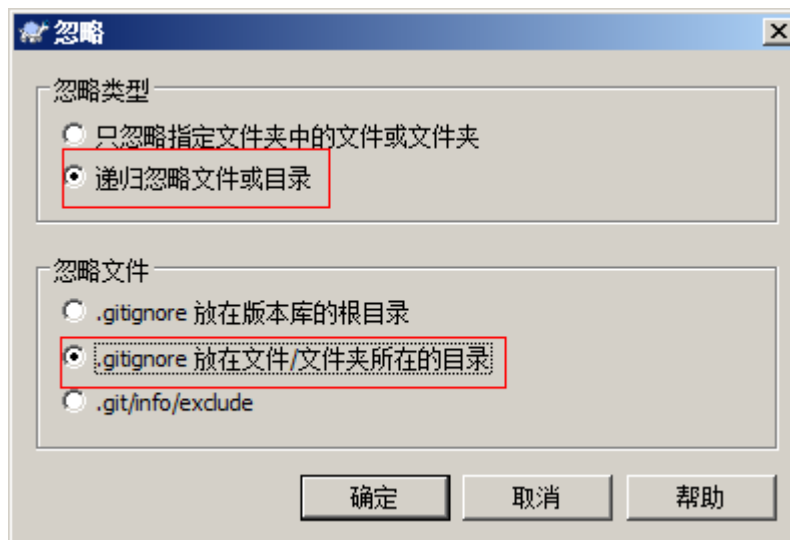
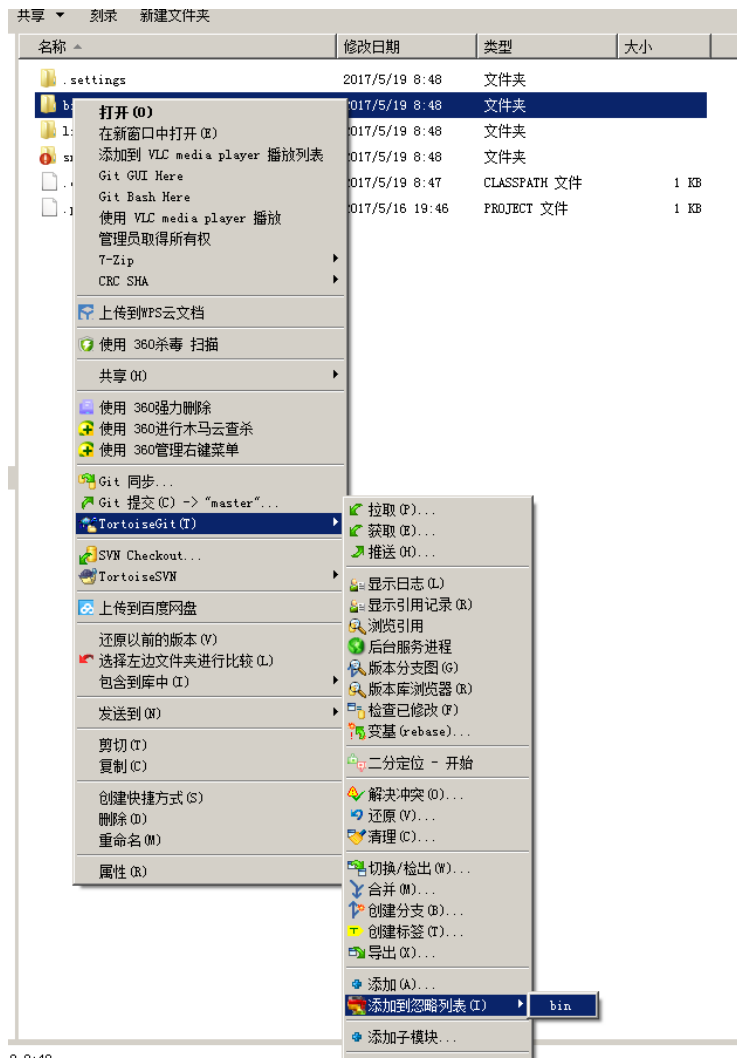


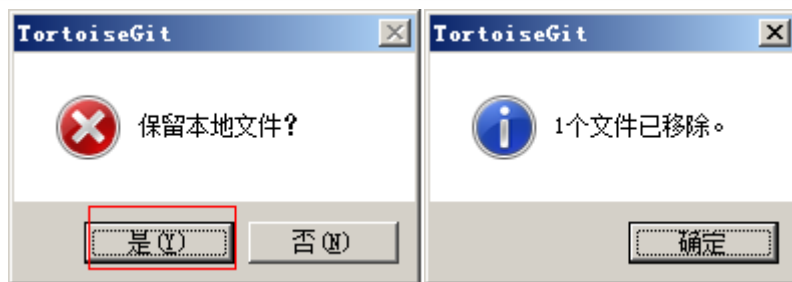
点击确定完成暂存区添加。

三、忽略文件或文件夹

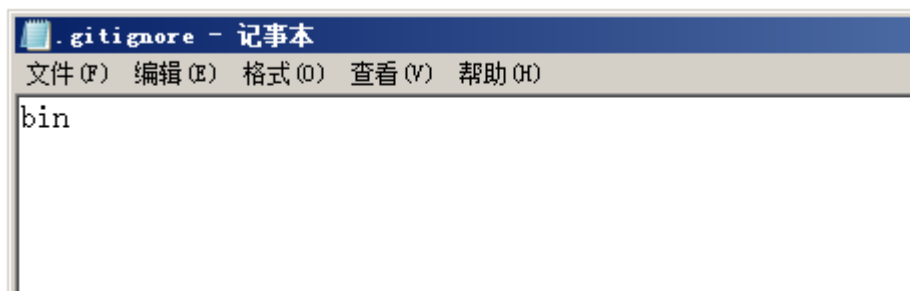
在此工程中，并不是所有文件都需要保存到版本库中的例如“bin”目录及目录下的文件就可以忽略。好在 Git 考虑到了大家的感受，这个问题解决起来也很简单，在 Git 工作区的根目录下创建一个特殊的.gitignore 文件，然后把要忽略的文件名填进去，Git 就会自动忽略这些文件。

如果使用 TortoiseGit 的话可以使用菜单项直接进行忽略。





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。



四、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。

5.6 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，还是应该来个栗子：

忽略 .a 文件

*.a

但否定忽略 lib.a, 尽管已经在前面忽略了 .a 文件



```
!lib.a

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

/TODO

# 忽略 build/ 文件夹下的所有文件

build/

# 忽略 doc/notes.txt, 不包括 doc/server/arch.txt

doc/*.txt

# 忽略所有的 .pdf 文件 在 doc/ directory 下的

doc/**/*.*pdf
```

6 远程仓库

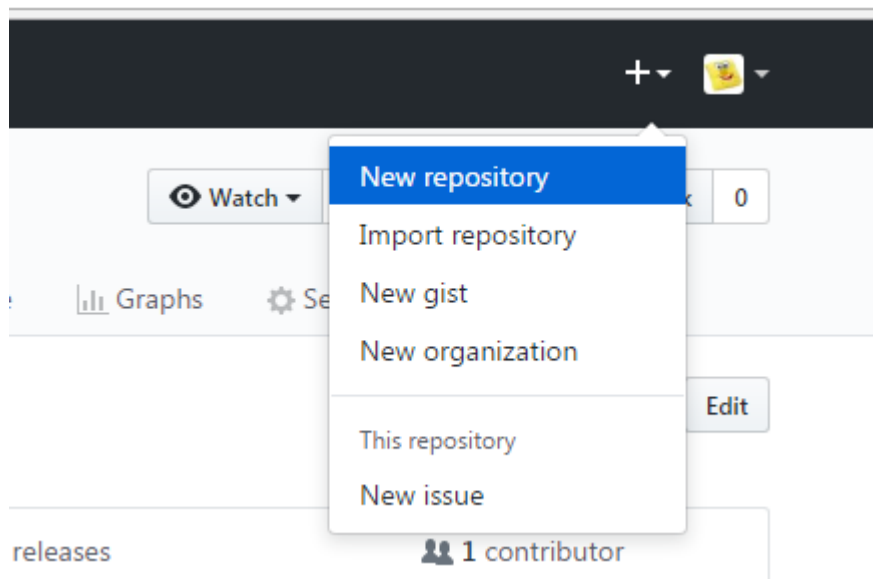
6.1 添加远程库

我们现在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/>，接下来我们演示如何将本地代码同步到 github。


6.1.1 在 github 上创建仓库

首先你得在 github 上创建一个账号，这个就不演示了。然后在 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:  subun / Repository name:

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应用，所以我們都需要掌握。接下来我们先看 ssh 方式。



6.1.2 ssh 协议

6.1.2.1 什么是 ssh?

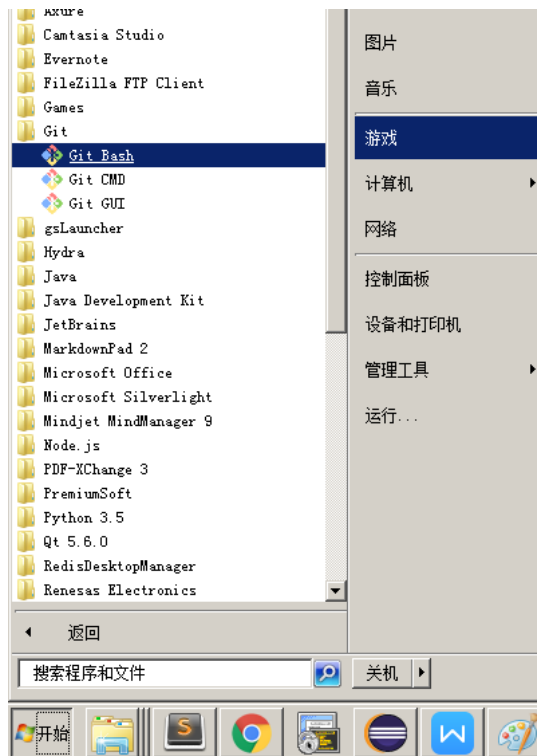
SSH 为 Secure Shell(安全外壳协议)的缩写,由 IETF 的网络小组(Network Working Group)所制定。SSH 是目前较可靠,专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

6.1.2.2 基于密匙的安全验证

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



git bash 执行命令,生命公钥和私钥

命令: **ssh-keygen -t rsa**

```

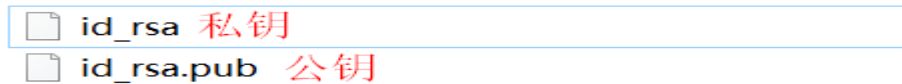
MINGW64: /d/temp/git/repository
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c:/Users/Administrator/.ssh/id_rsa): 敲回车即可
Enter passphrase (empty for no passphrase):
Enter same passphrase again: 一路回车
Your identification has been saved in /c:/Users/Administrator/.ssh/id_rsa.
Your public key has been saved in /c:/Users/Administrator/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:8a0c1ru1hk5J8WN7+pRouB4Jpvt4XINugN02ZJFRG3A 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)
$

```

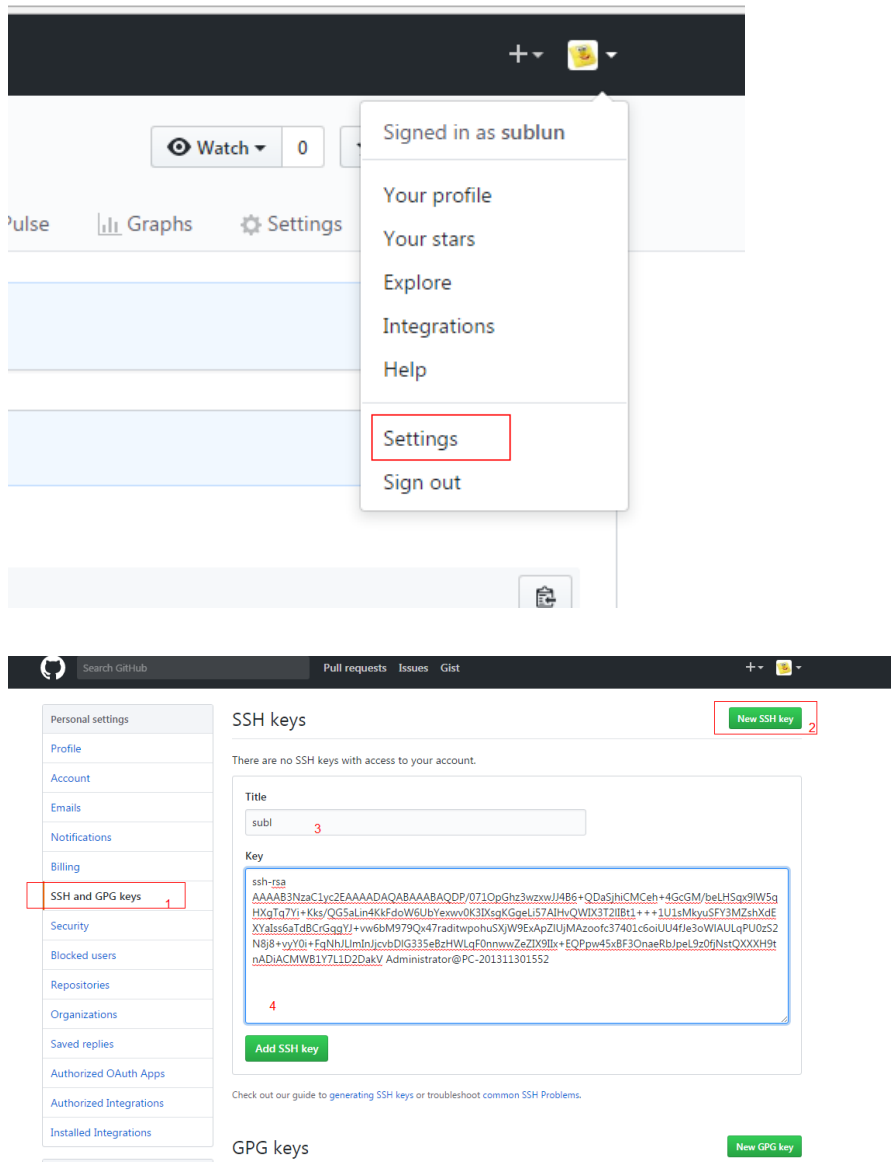


执行命令完成后,在 window 本地用户.ssh 目录 C:\Users\用户名\.ssh 下面生成如下名称的公钥和私钥:



6.1.2.4ssh 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。



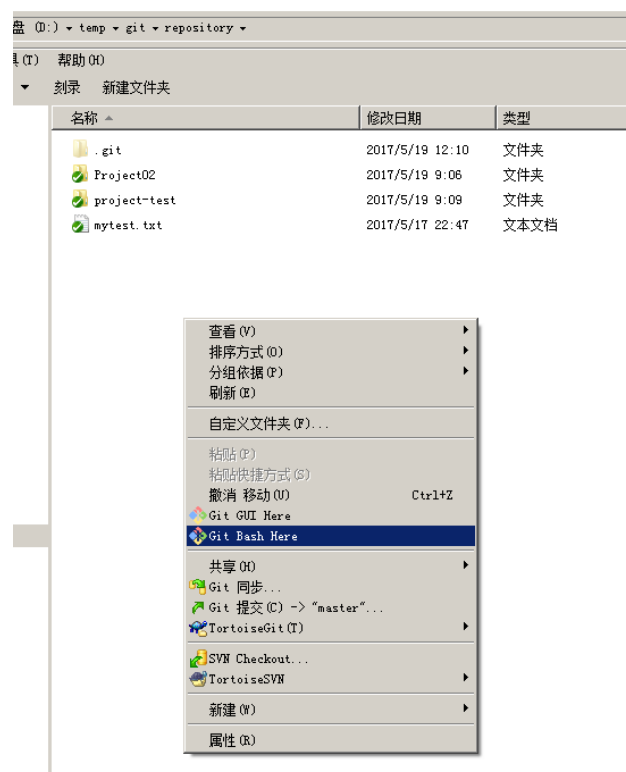
在 key 部分将 id_rsa.pub 文件内容添加进去，然后点击“Add SSH key”按钮完成配置。

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。



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：其中红色字体部分需要替换成个人的用户名。

如何出现如下错误：



```
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

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

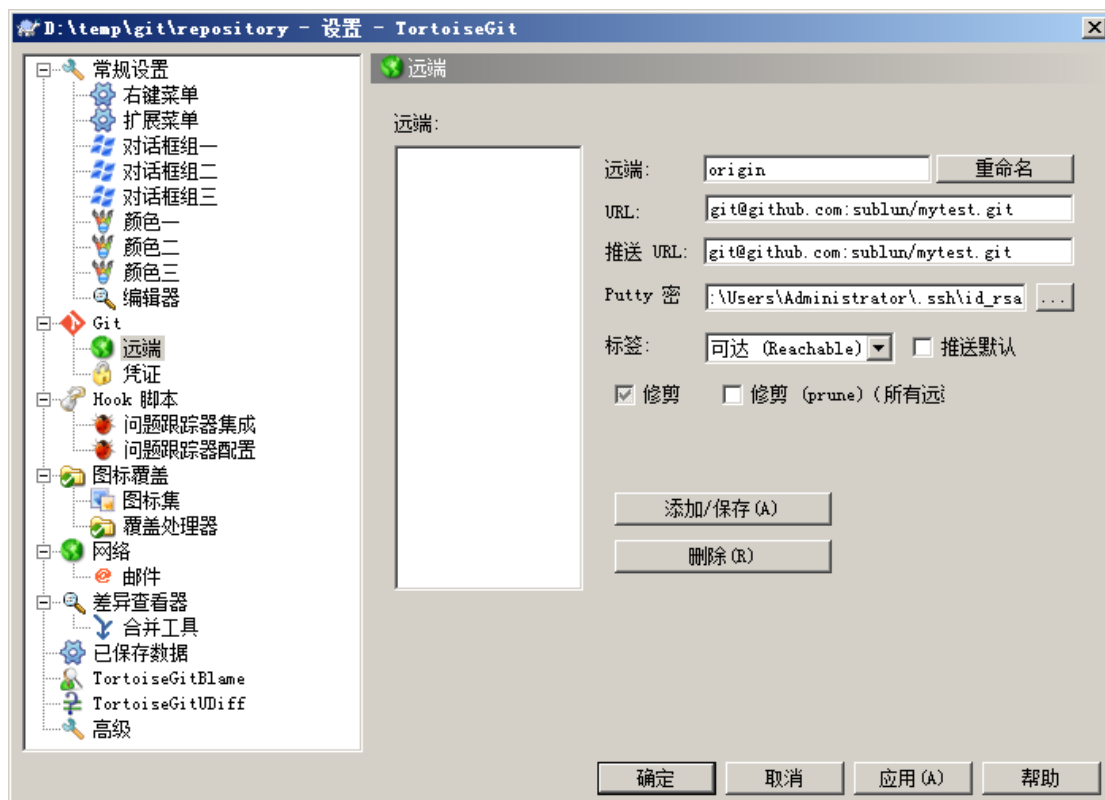
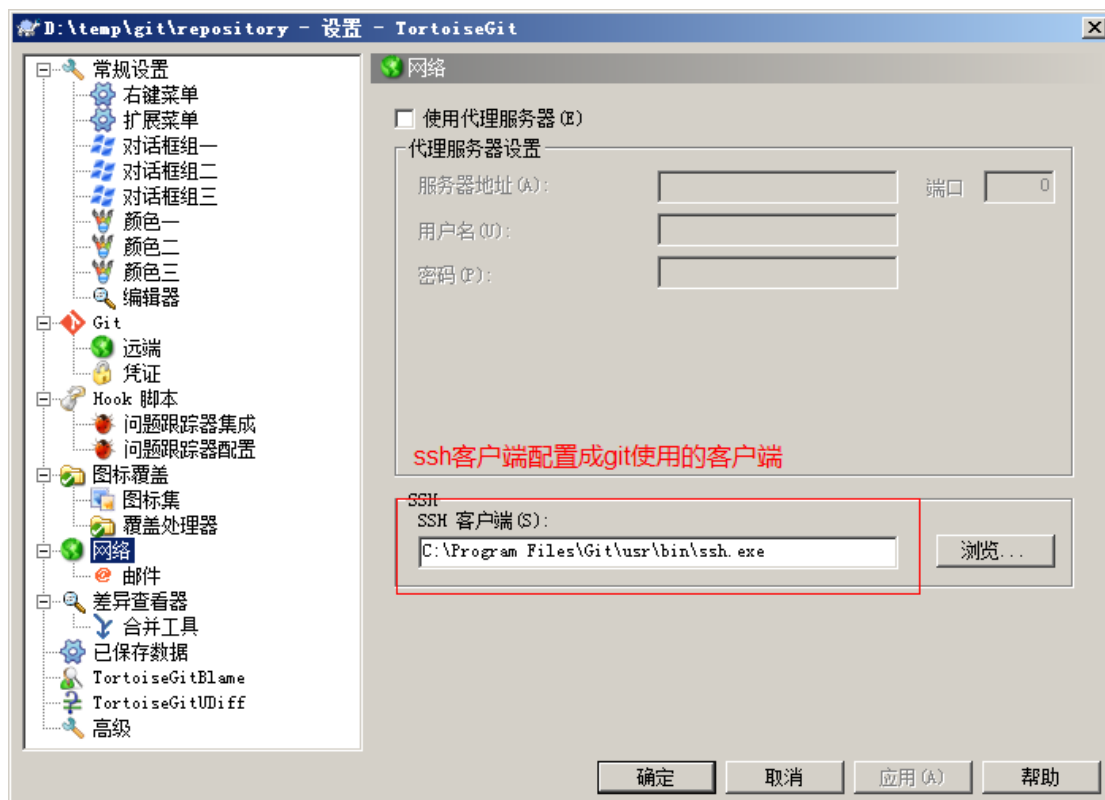
Administrator@PC-201311301552 MINGW64 /d/temp/git/repository (master)
$ git remote add origin git@github.com:sublun/mytest.git

Administrator@PC-201311301552 MINGW64 /d/temp/git/repository (master)
$ git push -u origin master
The authenticity of host 'github.com (192.30.255.113)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWG17EiIGOCspRomTxdCARLviKw6E5sY8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.255.113' (RSA) to the list of known hosts.
Counting objects: 44, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (25/25), done.
Writing objects: 100% (44/44), 215.62 KiB | 0 bytes/s, done.
Total 44 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), done.
To github.com:sublun/mytest.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

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

6.1.3.2使用 TortoiseGit 同步

一、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：



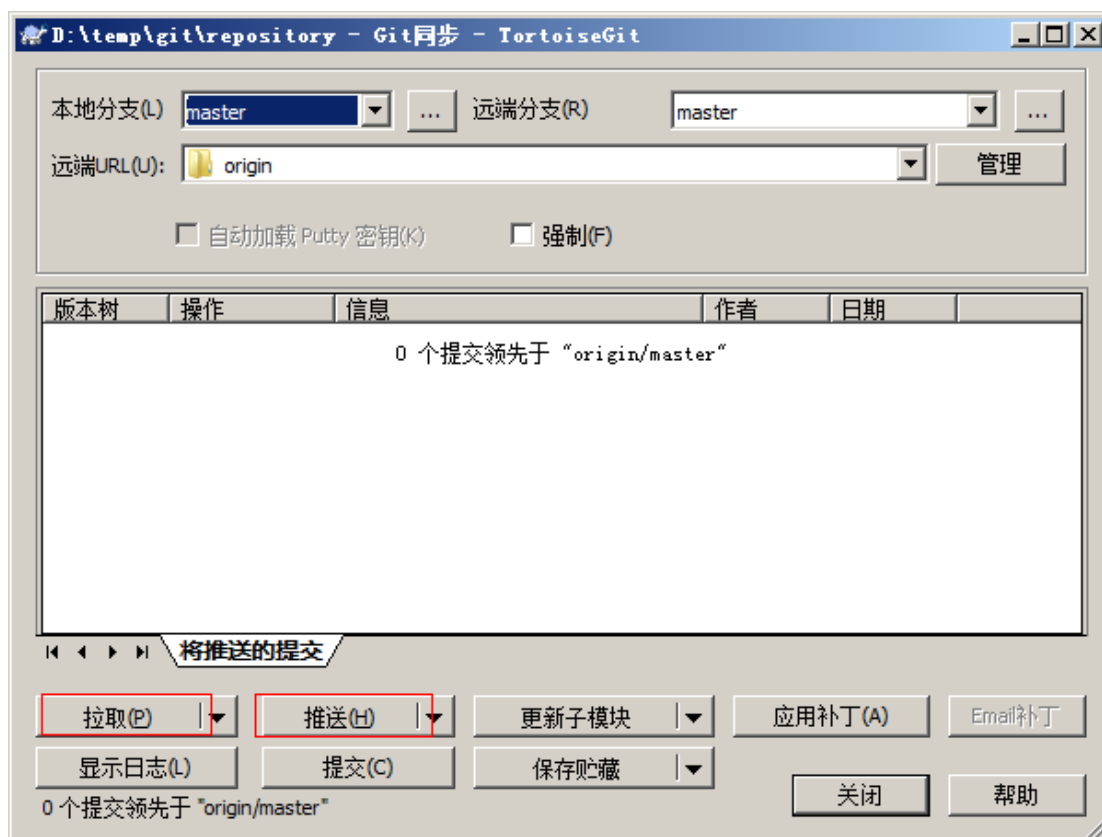
Url: 远程仓库的地址

推送 URL: 也是相同的



Putty 密钥：选择刚才生成的密钥中的私钥

二、同步。在本地仓库的文件夹中单击右键，选择“Git 同步”





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。

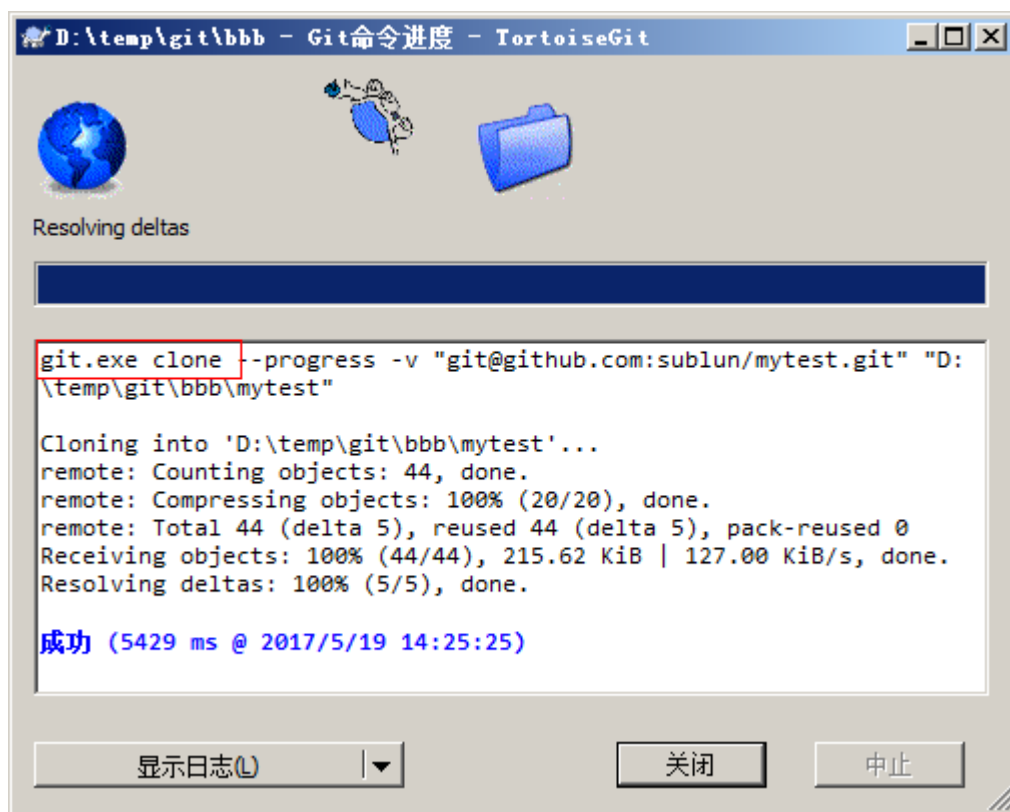
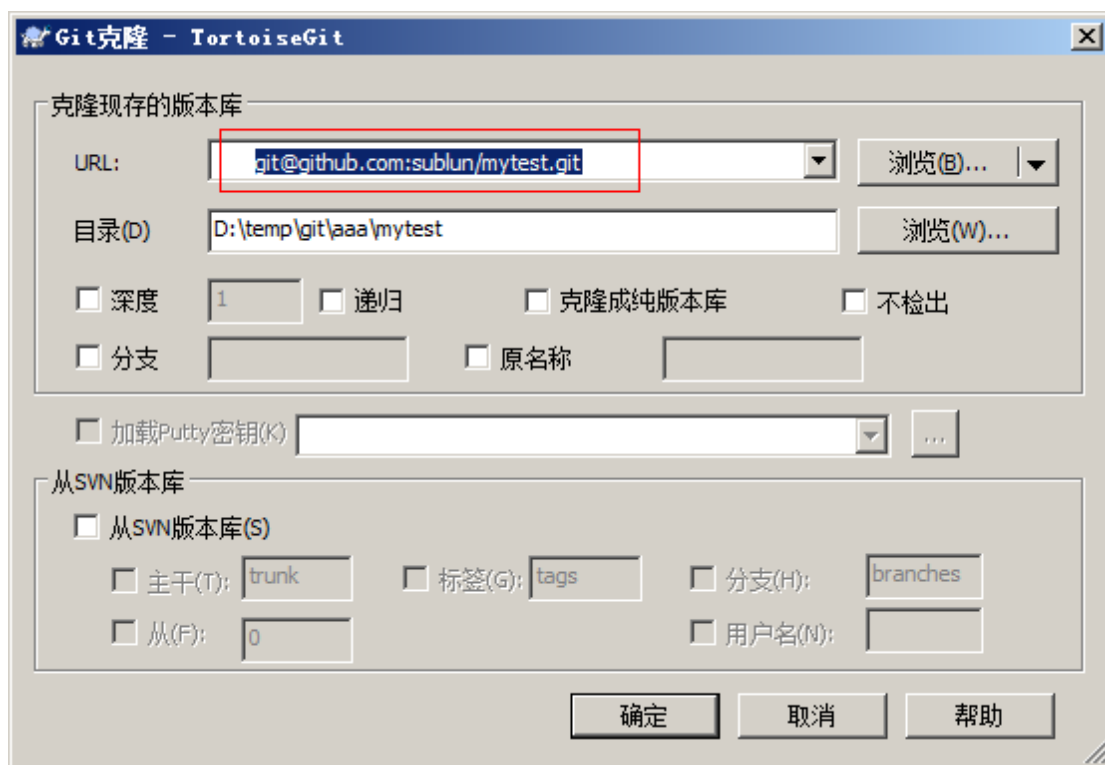
6.2.1 使用 git bash:

```
$ git clone git@github.com:sublun/mytest.git
```

6.2.2 使用 TortoiseGit:

在任意目录点击右键:





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 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

1) 解压缩

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@192.168.25.156/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
```

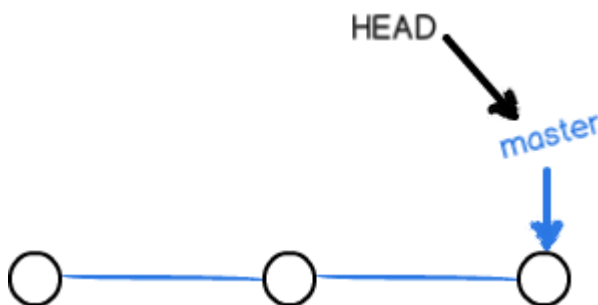

使用 TortoiseGit 同步的话参考上面的使用方法。

7 分支管理

7.1 创建合并分支

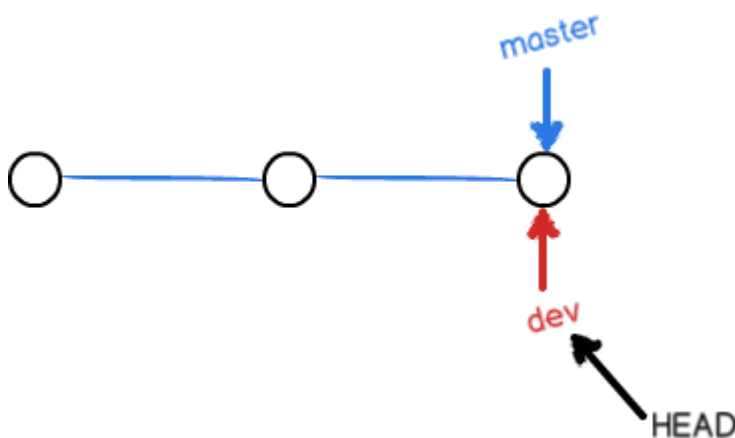
在我们每次的提交，Git 都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在 Git 里，这个分支叫主分支，即 master 分支。HEAD 指针严格来说不是指向提交，而是指向 master，master 才是指向提交的，所以，HEAD 指向的就是当前分支。

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。

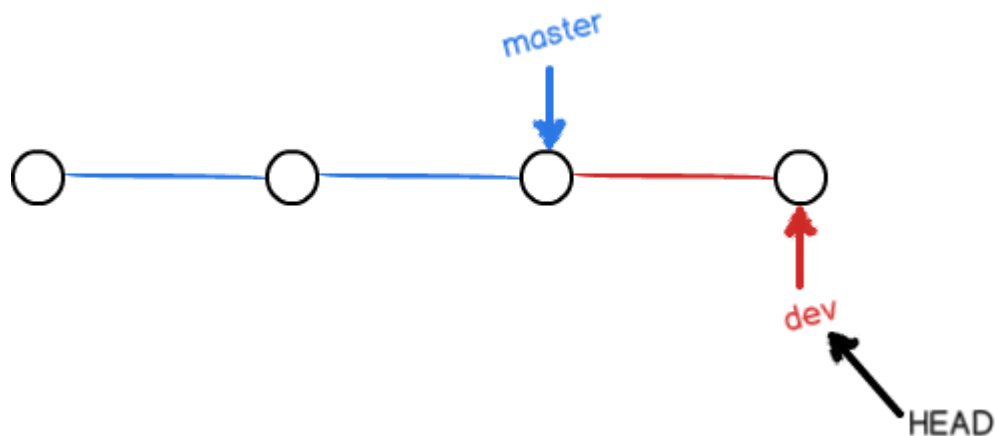
当我们创建新的分支，例如 dev 时，Git 新建了一个指针叫 dev，指向 master 相同的提交，再把 HEAD 指向 dev，就表示当前分支在 dev 上：



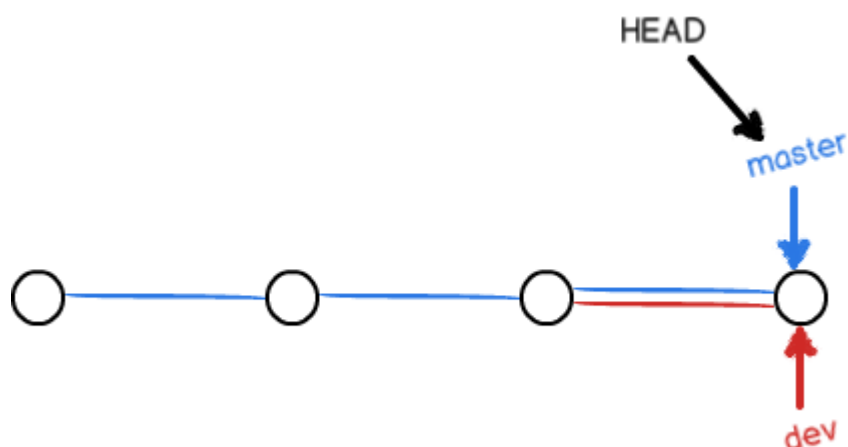
你看，Git 创建一个分支很快，因为除了增加一个 dev 指针，改改 HEAD 的指向，工作区的文件都没有任何变化！



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：

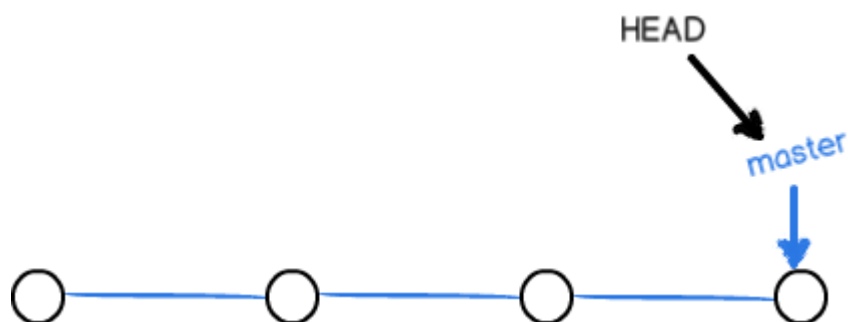


假如我们在 dev 上的工作完成了，就可以把 dev 合并到 master 上。Git 怎么合并呢？最简单的方法，就是直接把 master 指向 dev 的当前提交，就完成了合并：



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：

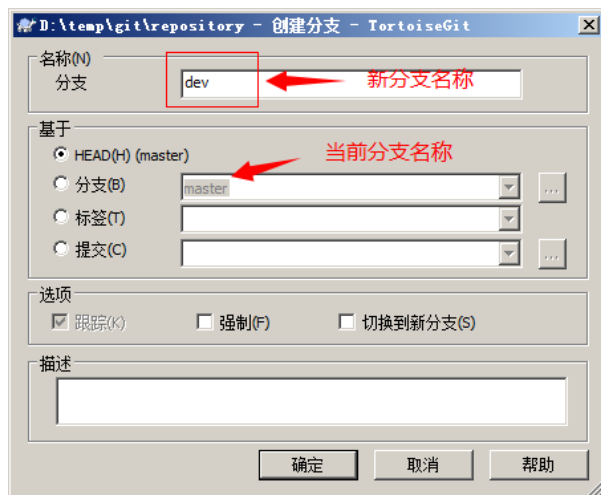


7.2 使用 TortoiseGit 实现分支管理

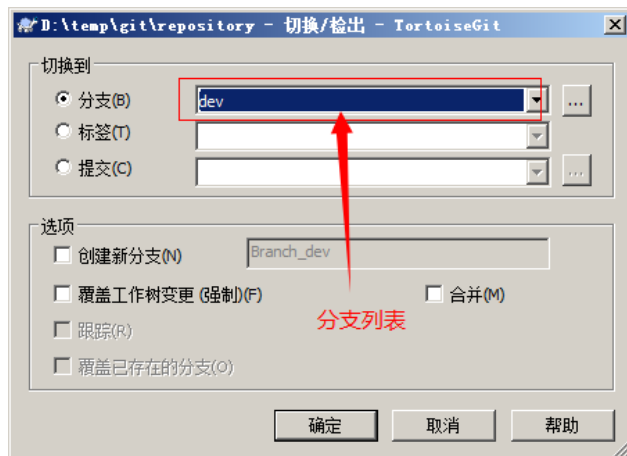
使用 TortoiseGit 管理分支就很简单了。

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”：

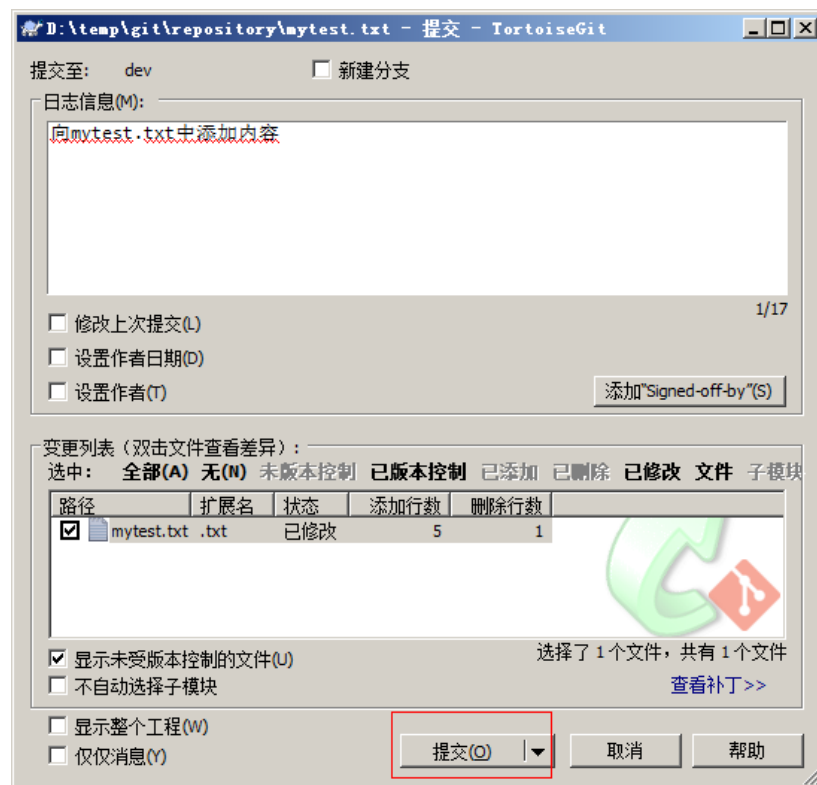


如果想创建完毕后直接切换到新分支可以勾选“切换到新分支”选项或者从菜单中选择“切换/检出”来切换分支：

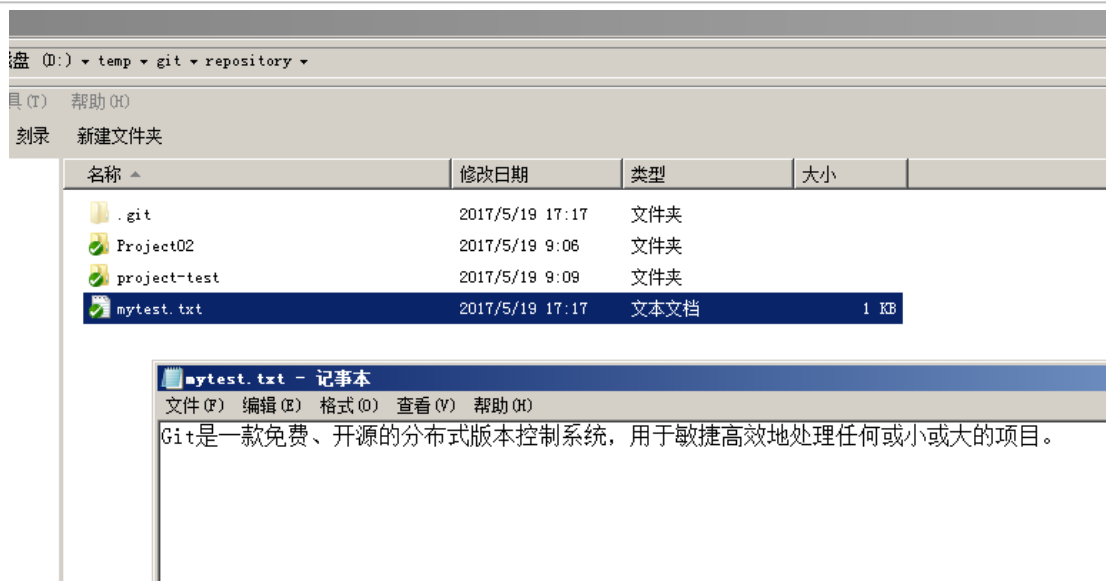


7.2.2 合并分支

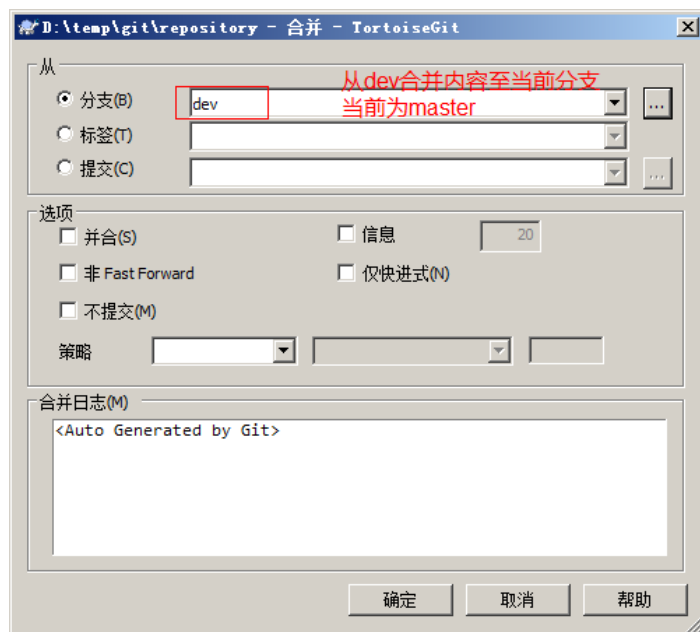
分支切换到 dev 后就可以对工作区的文件进行修改，然后提交到 dev 分支原理的 master 分支不受影响。例如我们修改 mytest.txt 中的内容，然后提交到 dev 分支。



切换到 master 分支后还是原理的内容：



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：



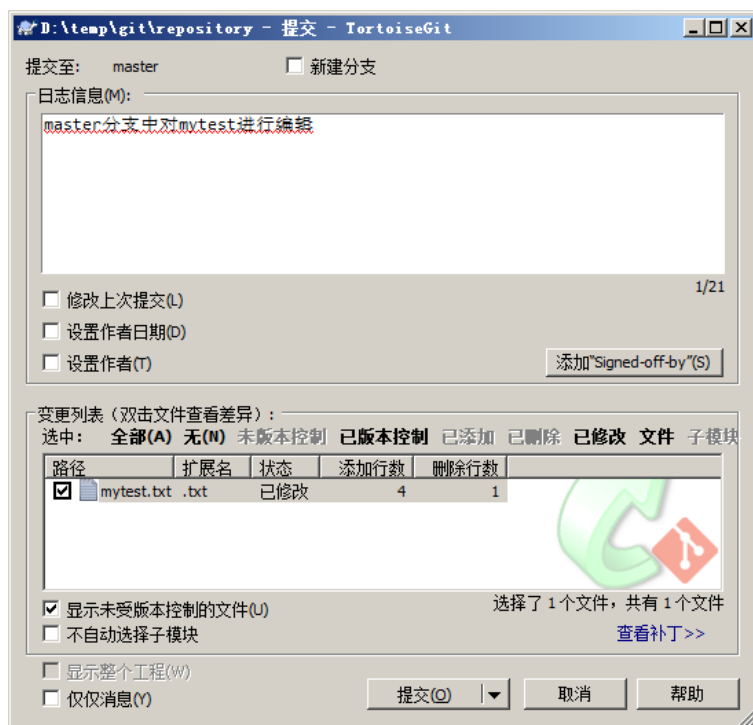
7.3 解决冲突

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：

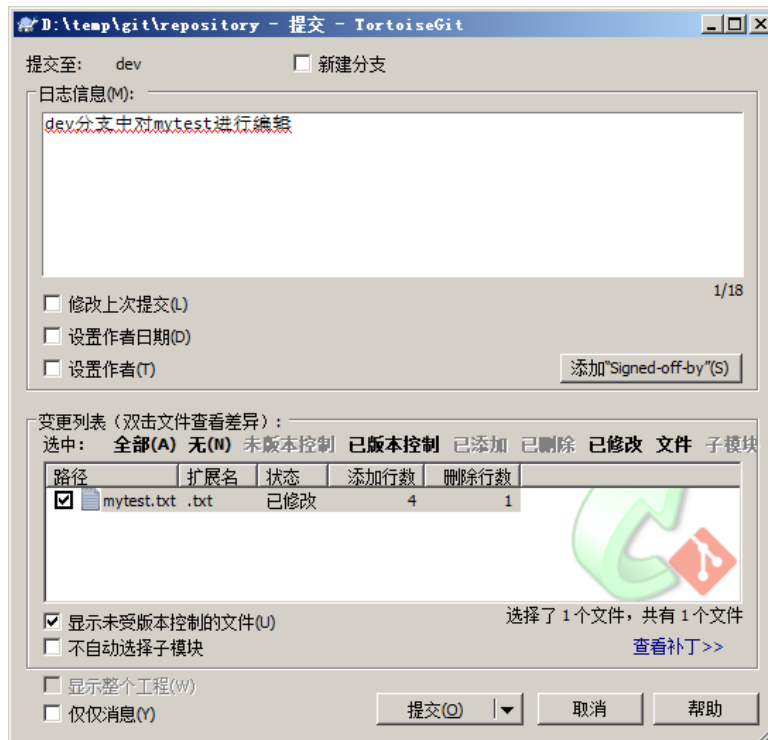


然后提交到版本库。

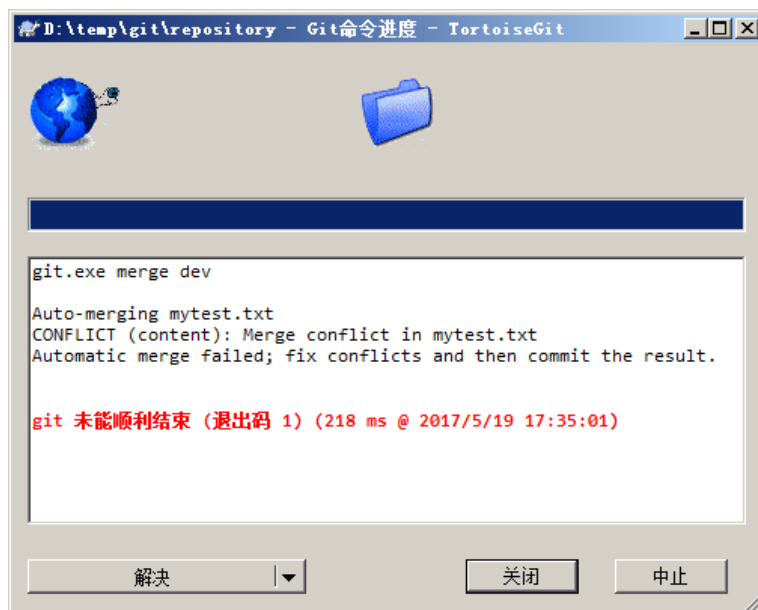


切换到 dev 分支，对 mytest.txt 进行编辑：

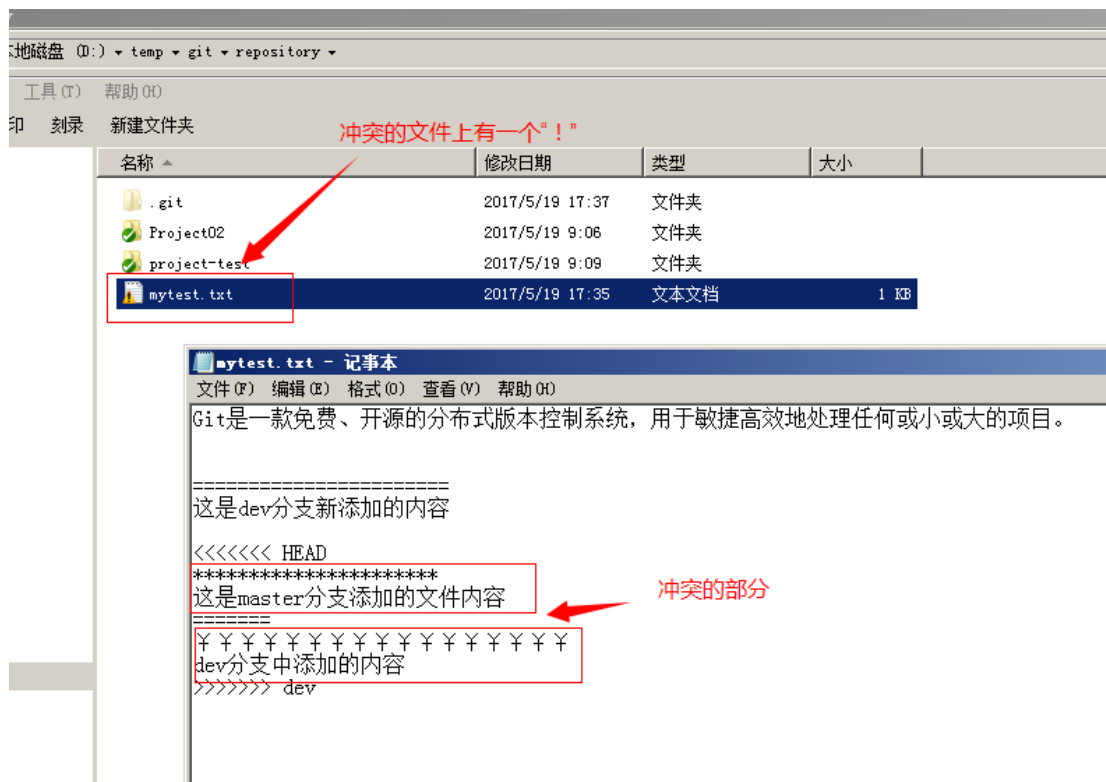




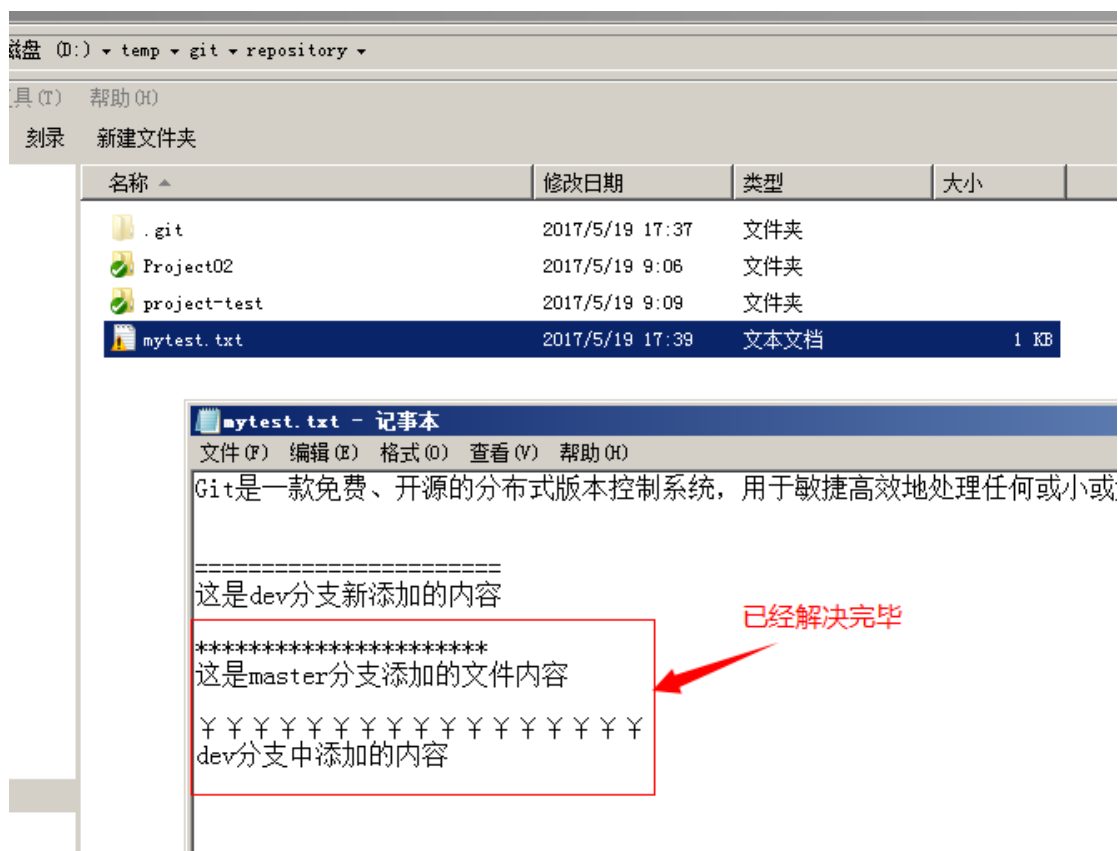
最后进行分支合并，例如将 dev 分支合并到 master 分支。需要先切换到 master 分支然后进行分支合并。



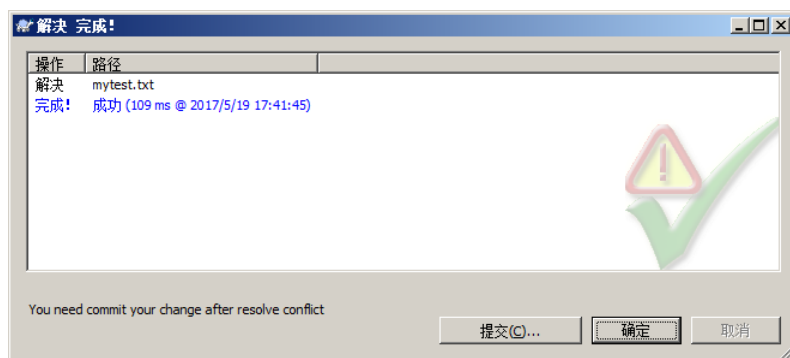
出现版本冲突。



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：



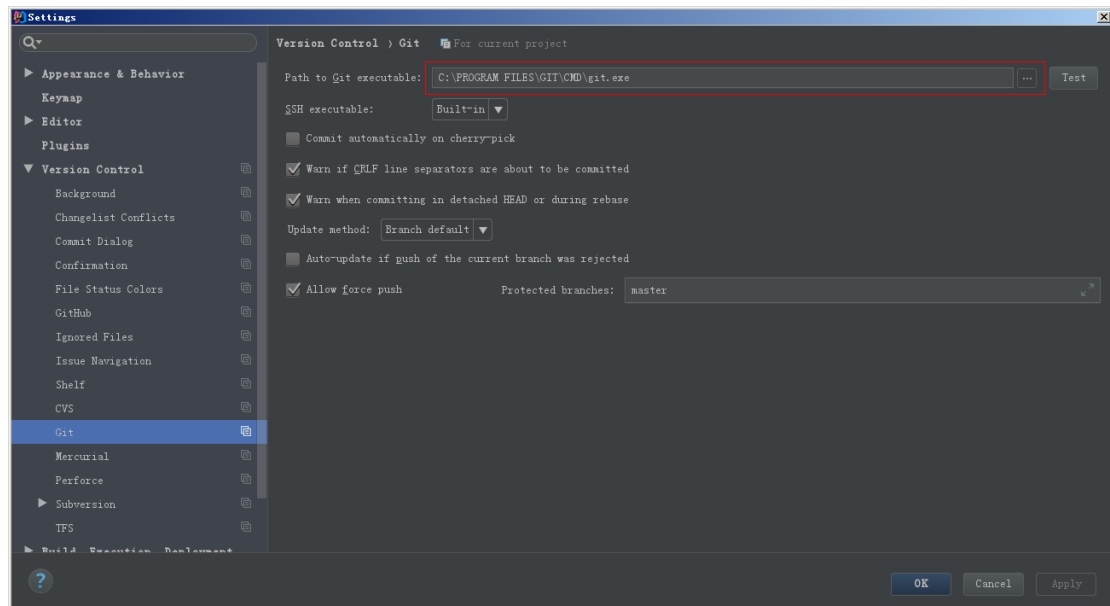
把冲突解决完毕的文件提交到版本库就可以了。

8 在 IntelliJ IDEA 中使用 git

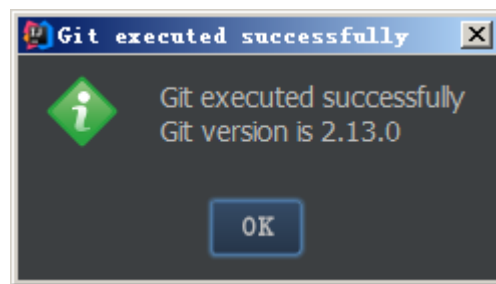
8.1 在 Idea 中配置 git

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：

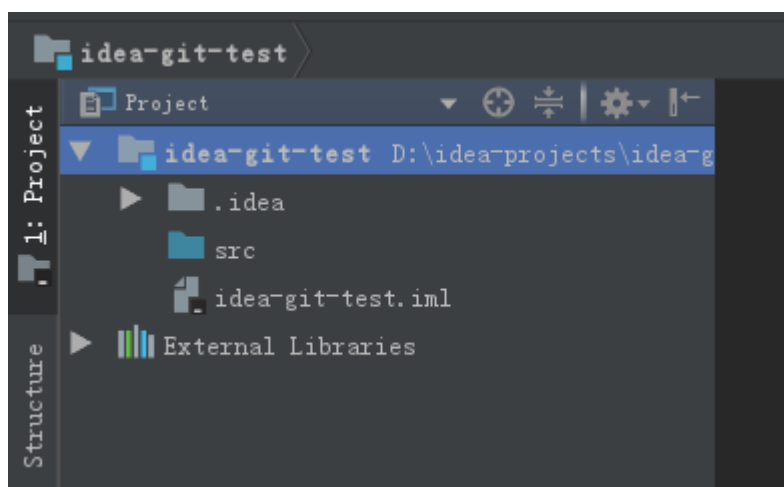


选择 git 的安装目录后可以点击“Test”按钮测试是否正确配置。



8.2 将工程添加至 git

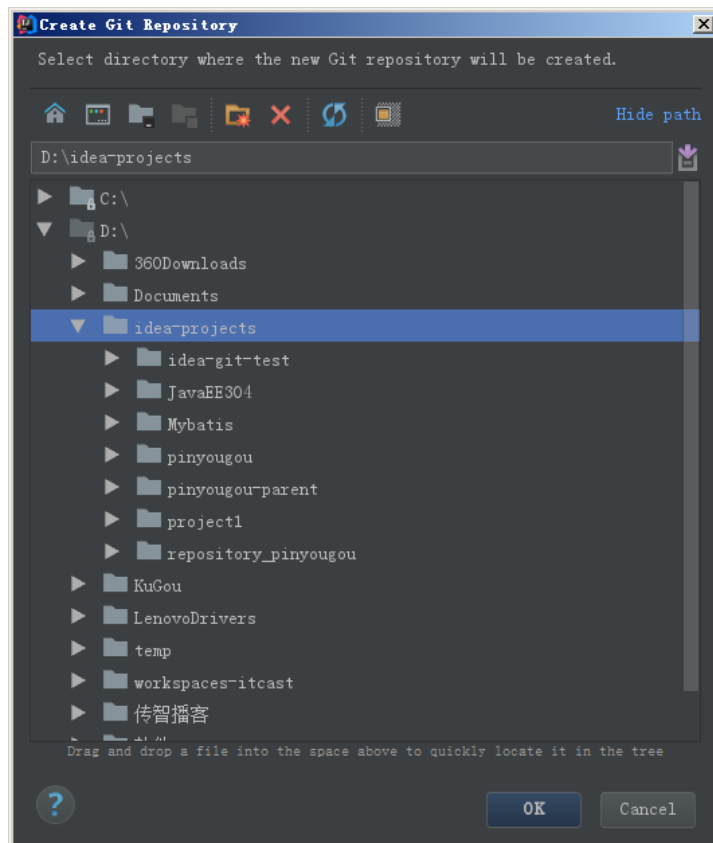
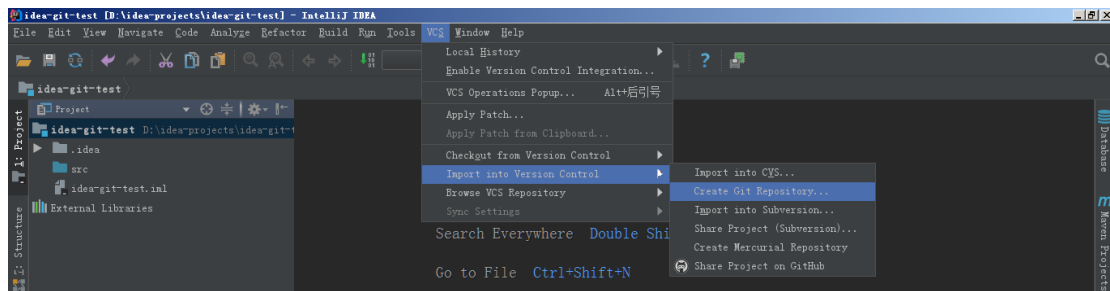
1) 在 idea 中创建一个工程，例如创建一个 java 工程，名称为 idea-git-test，如下图所示：



2) 创建本地仓库

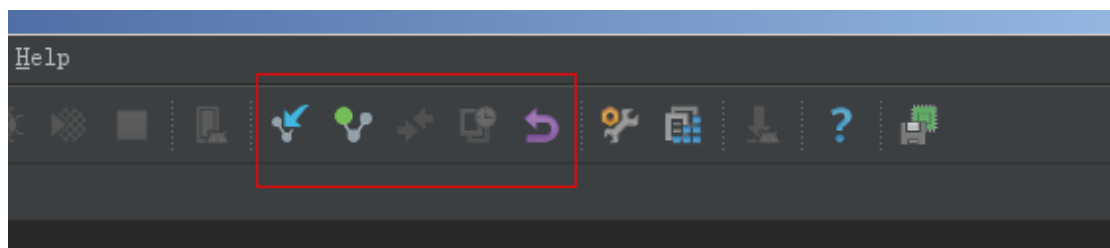


在菜单中选择“vcs”→Import into Version Control→Create Git Repository...



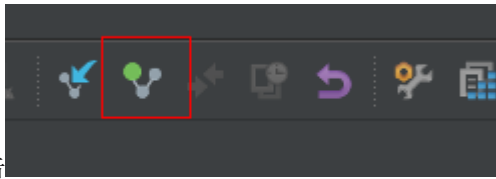
选择工程所在的上级目录。本例中应该选择 idea-projects 目录，然后点击“OK”按钮，在工程的上级目录创建本地仓库，那么 idea-projects 目录就是本地仓库的工作目录，此目录中的工程就可以添加到本地仓库中。也就是可以把 idea-git-test 工程添加到本地仓库中。

选择之后在工具栏上就多出了 git 相关工具按钮：

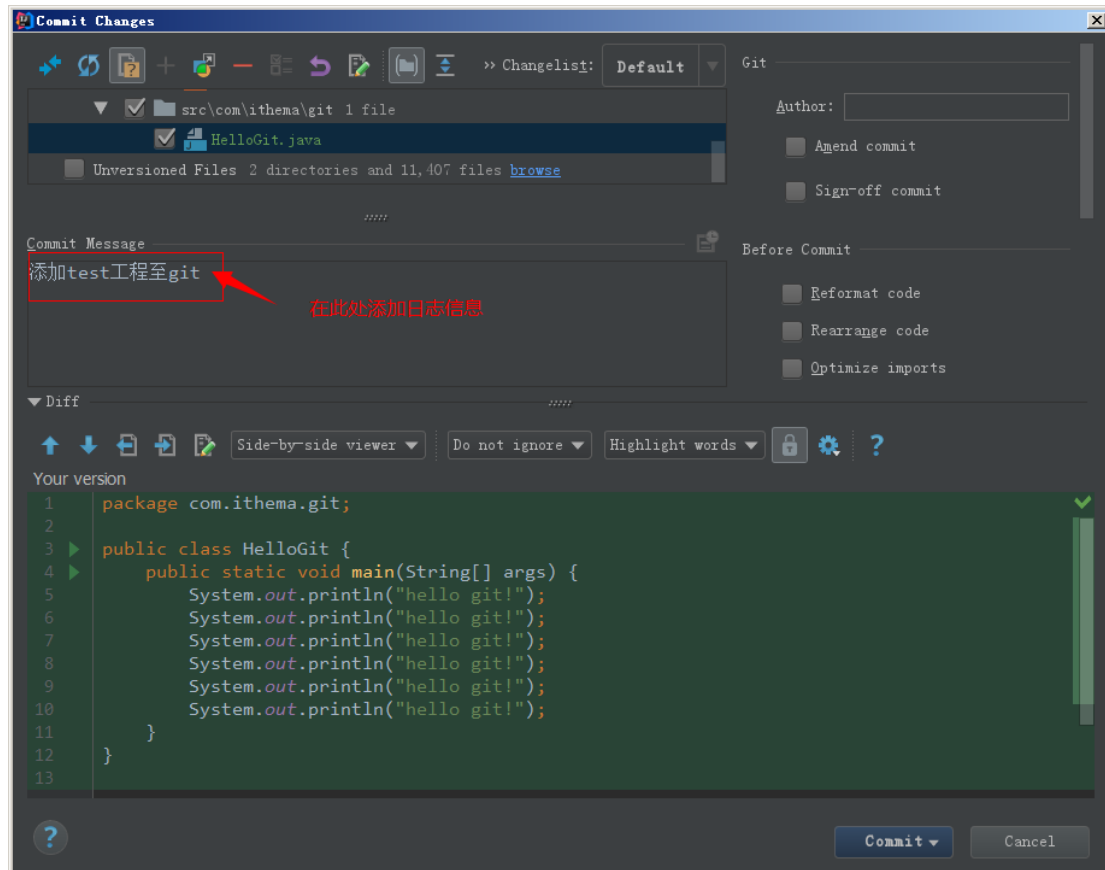




3) 将工程添加至本地仓库



直接点击 **commit** 按钮，将工程提交至本地仓库。



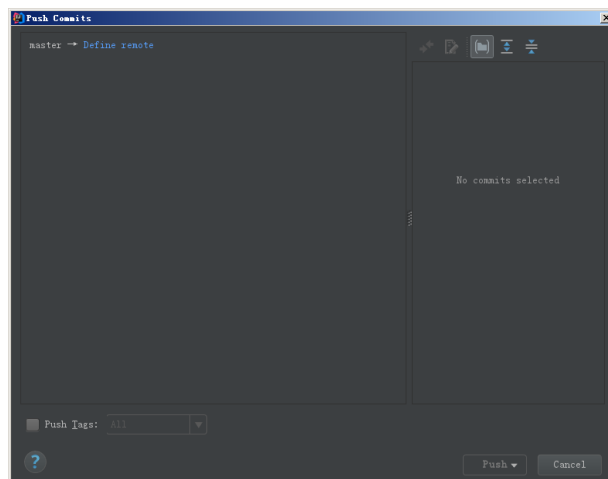
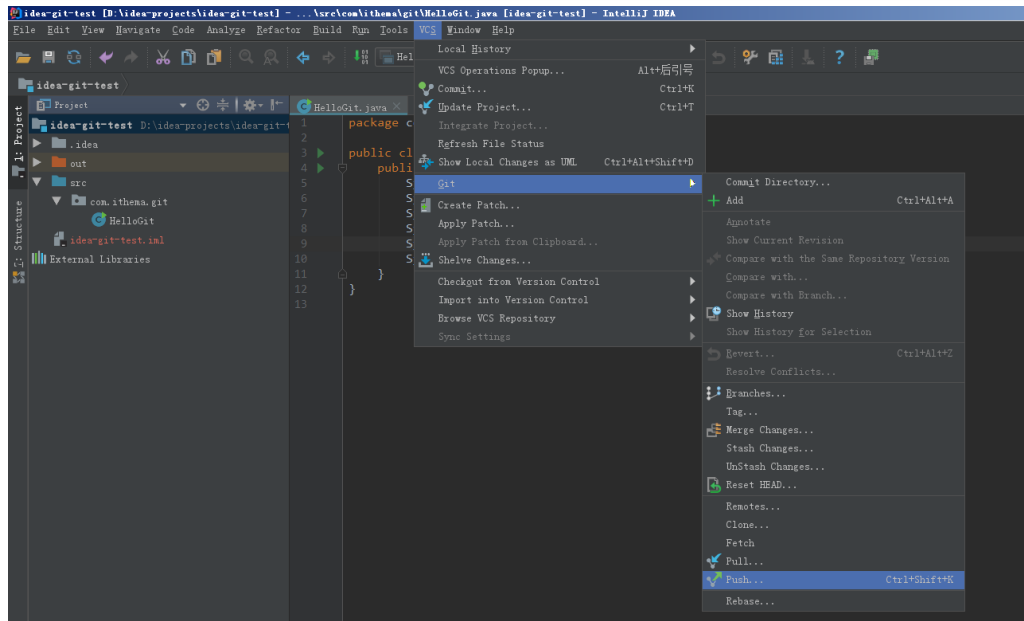
然后点击“commit”按钮，将工程添加至本地仓库。

4) 推送到远程

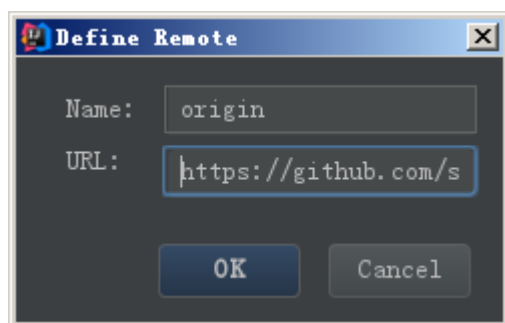
在 github 上创建一个仓库然后将本地仓库推送到远程。

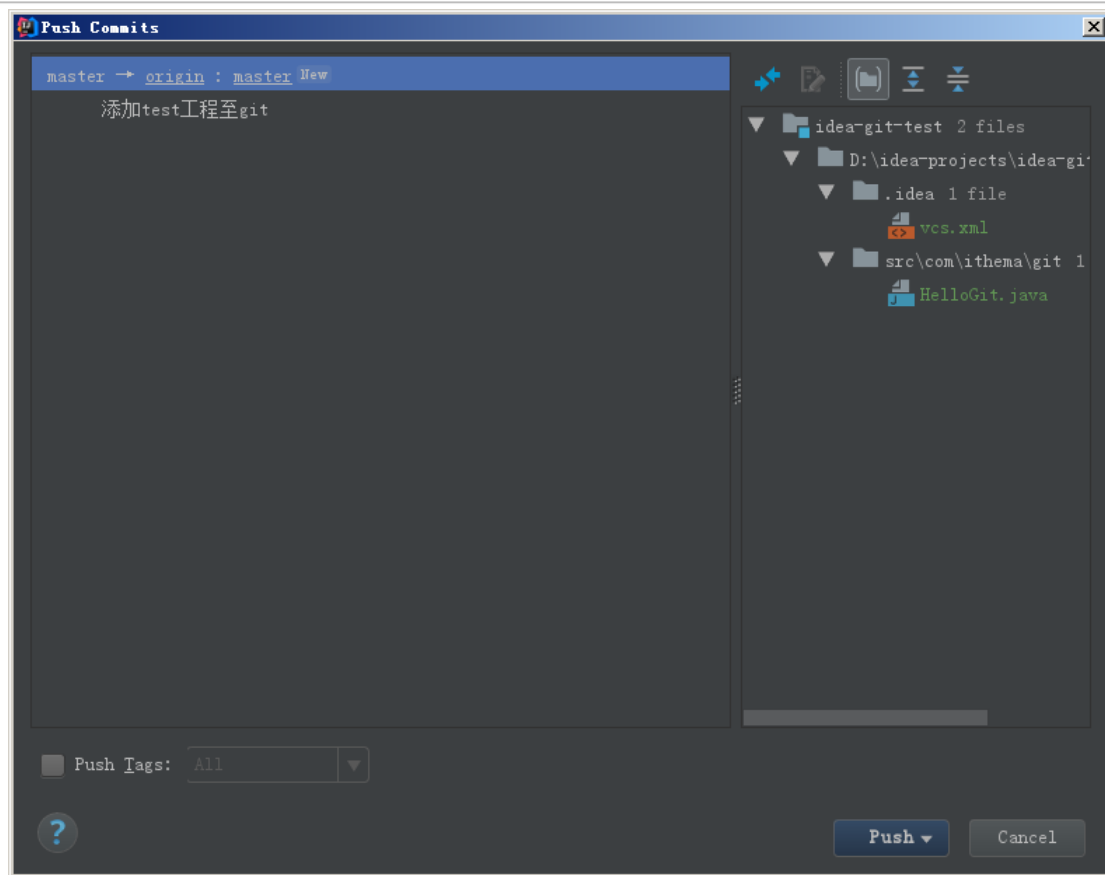
在工程上点击右键，选择 git→Repository→push,

或者在菜单中选择 vcs→git→push



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

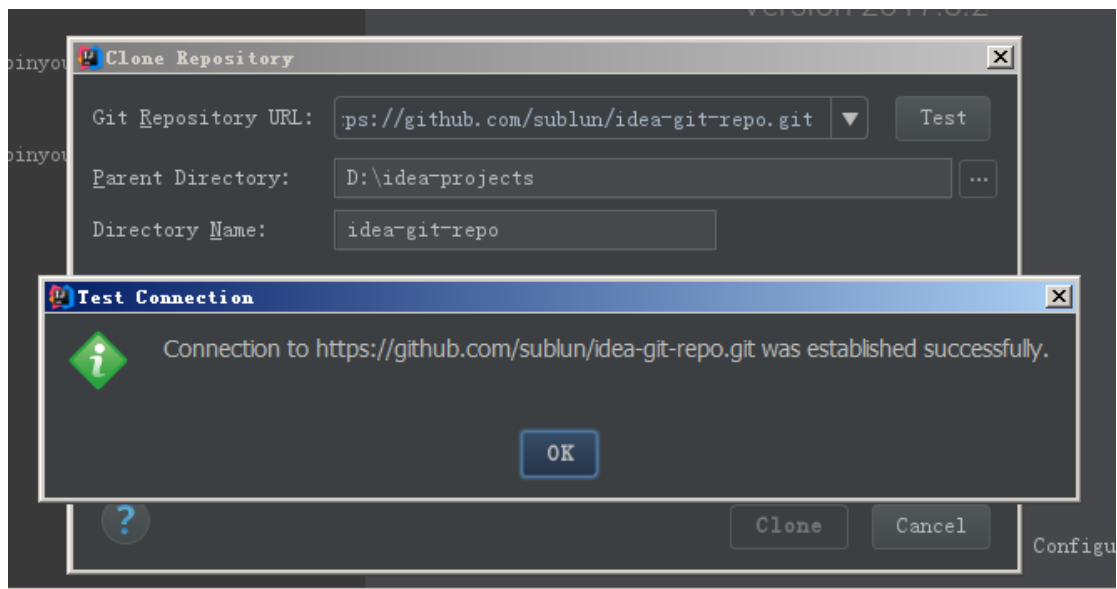
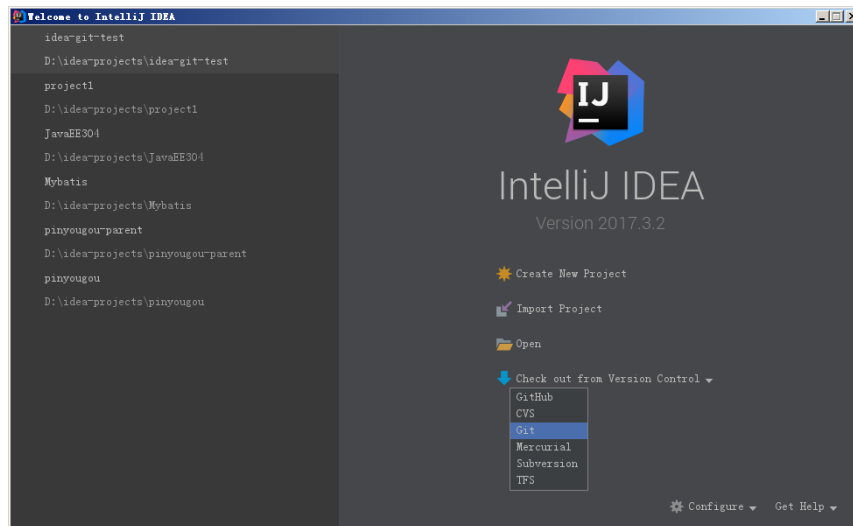




点击“push”按钮就将本地仓库推送到远程，如果是第一次配置推送需要输入 github 的用户名和密码。

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

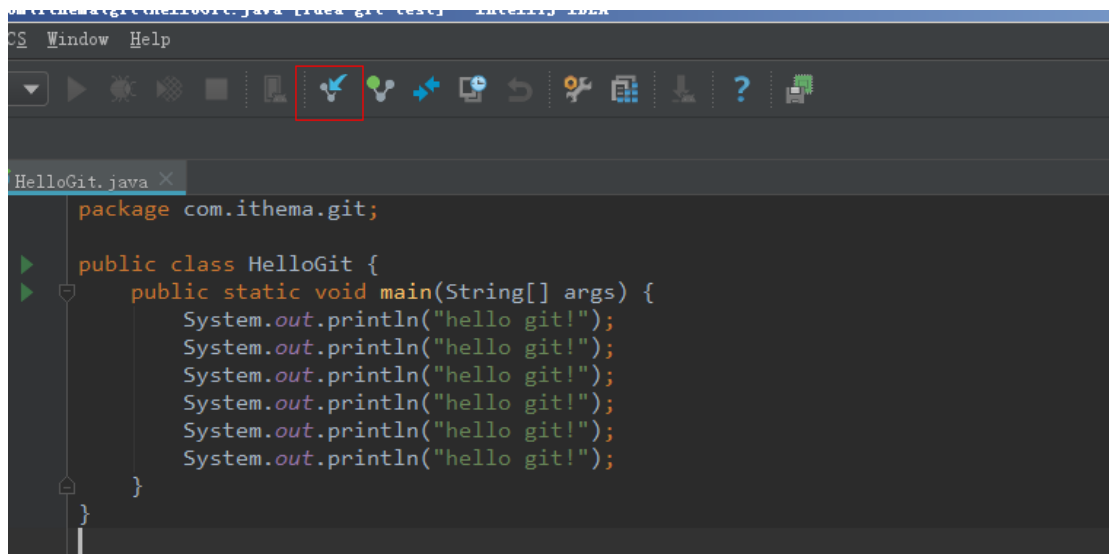


此处仍然推荐使用 https 形式的 url，点击“test”按钮后显示连接成功。

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。

8.4 从服务端拉取代码

如果要从服务端同步代码可以使用工具条中的“update”按钮



```
package com.ithema.git;

public class HelloGit {
    public static void main(String[] args) {
        System.out.println("hello git!");
        System.out.println("hello git!");
        System.out.println("hello git!");
        System.out.println("hello git!");
        System.out.println("hello git!");
        System.out.println("hello git!");
    }
}
```