

1. 学习版本控制的原因

1.1 没有版本控制出现的问题

- 备份多个版本，浪费存储空间，花费时间长。
- 难以恢复至以前的历史版本，容易引发BUG，解决代码冲突困难。
- 难于追溯问题代码的修改人和修改时间、修改内容、日志信息。
- 项目升级，版本发布困难。
- 无法进行权限控制。比如 测试人员：只读；开发人员：模块权限。
- 开发团队在工作过程中无法多条生产线同时推进任务，效率慢。

1.2 版本控制的简介

- 版本控制(Revision control)是维护工程蓝图的标准做法，能追踪工程蓝图从诞生一直到定案的过程。是一种记录若干文件内容变化，以便将来查阅特定版本修订情况的系统。

1.3 版本控制工具

- 集中式版本控制工具：**SVN**、VSS、CVS、
- 分布式版本控制工具：
Git、Mercurial、Bazaar.....

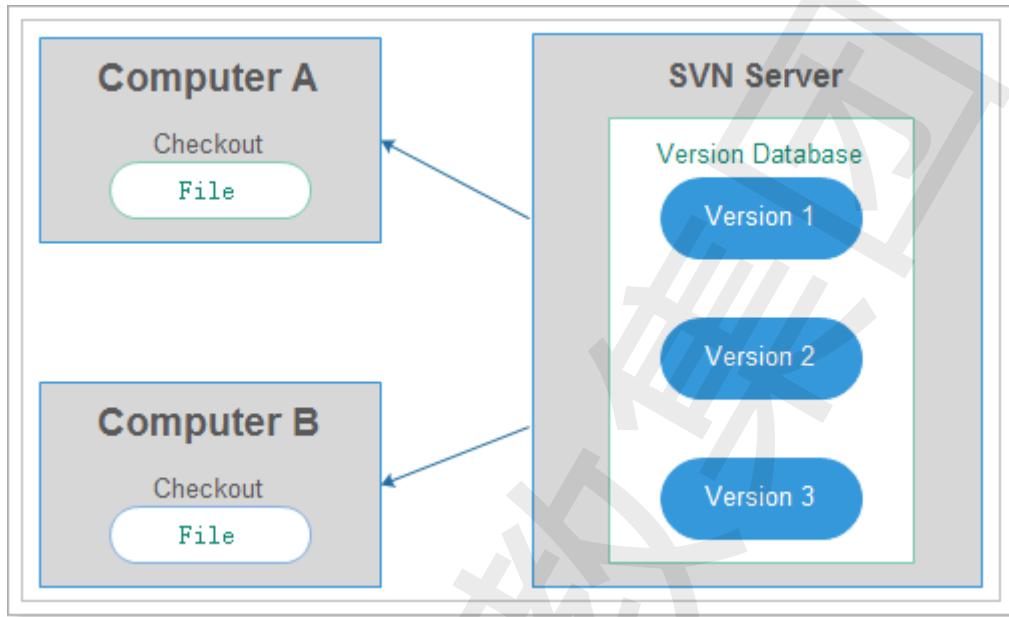
2. Git 和 Svn 比较

2.1 SVN介绍

2.1.1 SVN简介

- SVN 属于集中式版本管理控制系统，服务器中保存了所有文件的不同版本，而协同工作人员通过连接svn服务器，提取出最新的文件，获取提交更新。Subversion 项目的初衷是为了替换当年开源社区最为流行的版本控制软件CVS，在CVS的功能的基础上有很多的提升同时也能较好的解决CVS系统的一些不足。

2.1.2 SVN基本交互流程图



2.1.3 SVN缺点

- 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
 - 必须连接在SVN服务器上，否则不能提交、对比、还原等

2.2 Git 介绍

2.2.1 Git与Svn记录具体差异

- Git 和其他版本控制系统的主要差别在于，Git 只关心文件的整体是否发生变化。而SVN这类版本控制系统则只关心文件内容的具体差异。
- SVN这类系统每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容。然而 Git 并不保存这些前后变化的差异数据。
- 实际上，Git更像是把 变化的文件 作一个快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息 (Hash值) 并对文件作一快照，然后保存一个指向这次快照的索引。为提高性能，若文件没有变化，Git 不会再次保存，而只对上次保存的快照作一链接。

2.2.2 Git的优势是什么

- 分布式，强调个体
- 公共服务器压力和数据量都不会太大
- 离线工作，每个人的本地仓库，大部分操作在本地库完成，不需要联网（SVN做不到）
- 分支操作非常快捷流畅(重点介绍)
- 可以吃后悔药，尽可能添加数据而不是删除或修改数据（删除或修改不容易恢复，而每次添加一个版本，历史版本都有）
- 速度快、灵活，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）

2.2.3 Git 历史发展

- 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。
一开始 linus(莱纳斯)本人手动合并代码，Linux 开源项目有着来自世界各地的开发者参与，绝大多数的 Linux 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，商业软件 Bitkeeper 出于人道语义精神，授权给linux免费使用，要求是不能进行破解。于是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，linux社区开发者试图破解BitKeeper协议，但是被公司发现了，于是开发 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 ）不得不吸取教训，于是 莱纳斯 自己用C语言开发了一套属于自己的版本控制系统，不至于重蹈覆辙。Git主体程序只发了两周，一个月后linux系统代码由git管理。
- 2008年，Github上线了，很多开源软件都放在Github上进行公布：Linux , Android, jQuery, Ruby , PHP , vue.js... 目前使用Git的项目数量也已经超过了使用SVN的仓库数。

3. Git 下载与安装

3.1 Git 下载



- 官网：<https://git-scm.com/>
- 软件下载地址：<https://git-scm.com/downloads>

Downloads



- 根据自己的电脑版本下载对应版本：

Other Git for Windows downloads

Git for Windows Setup
32-bit Git for Windows Setup. 32位操作系统

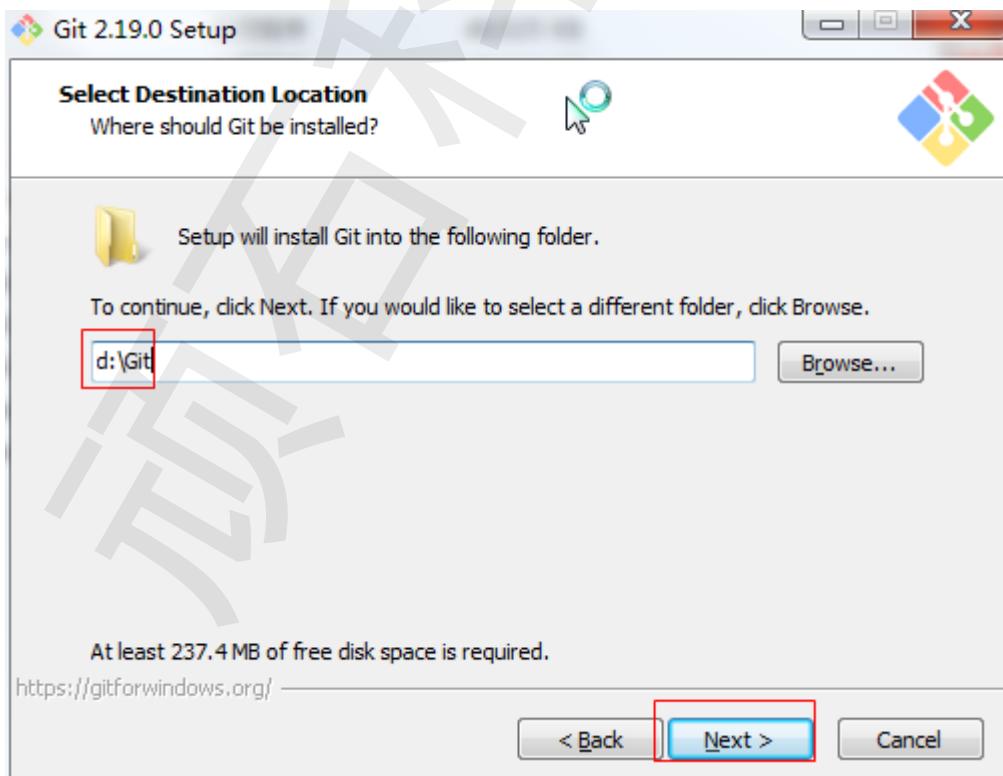
64-bit Git for Windows Setup. 64位操作系统

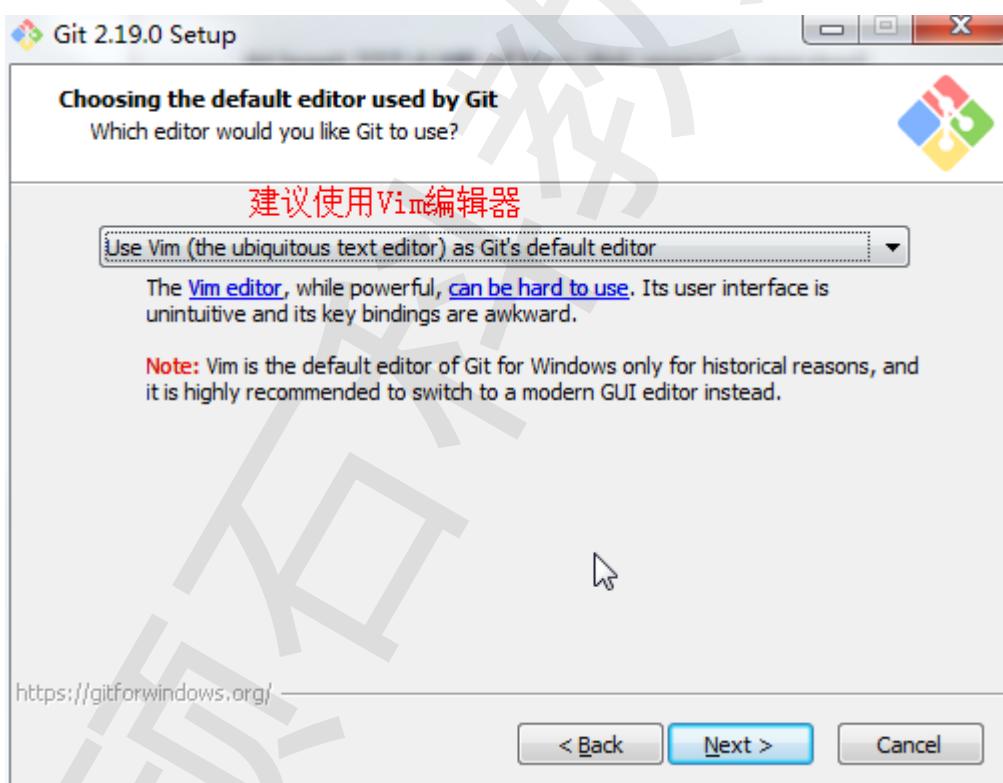
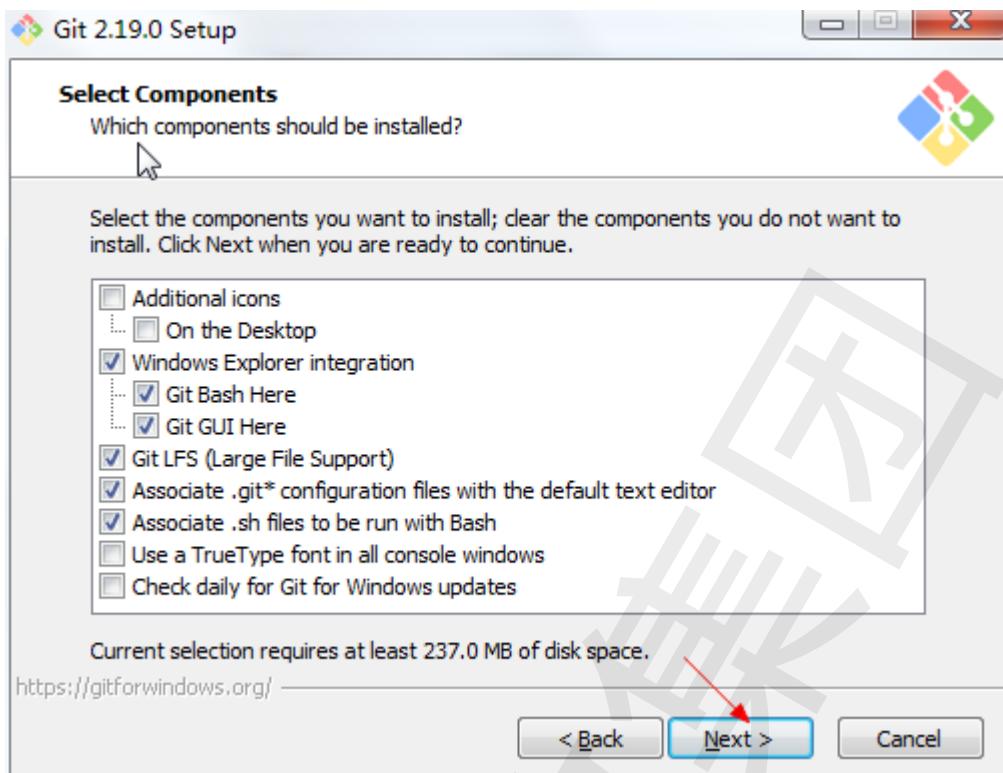
Git for Windows Portable ("thumbdrive edition")
32-bit Git for Windows Portable.

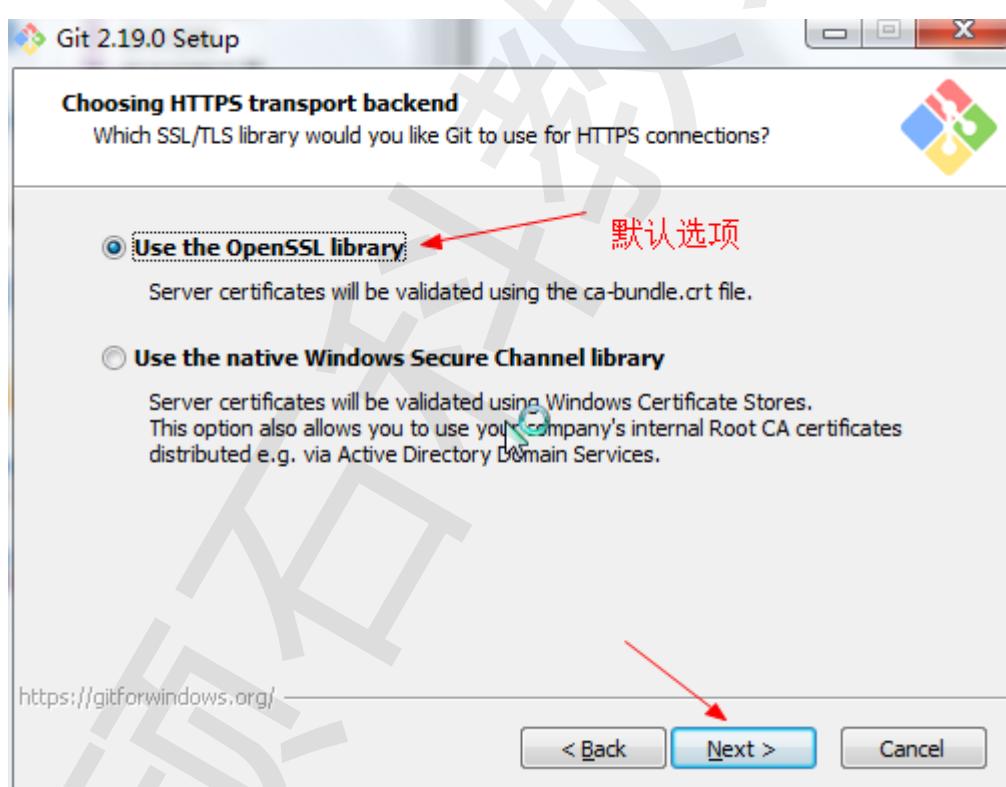
64-bit Git for Windows Portable.

The current source code release is version 2.18.0. If you want the newer version, you can [download it from the source code](#).

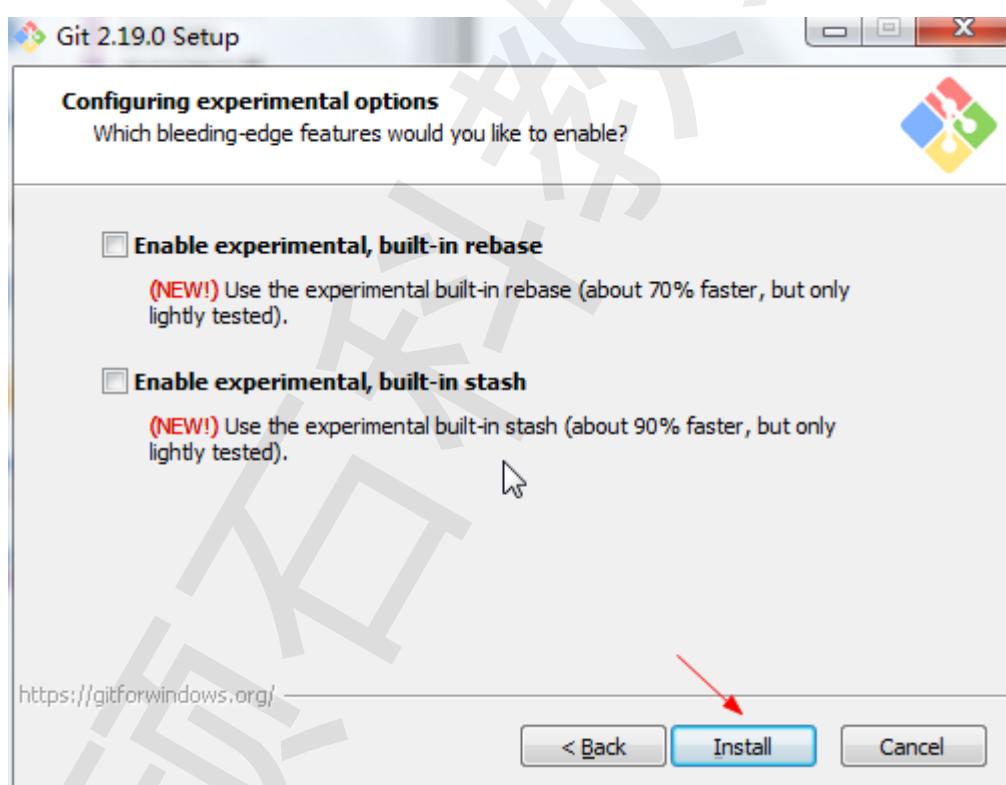
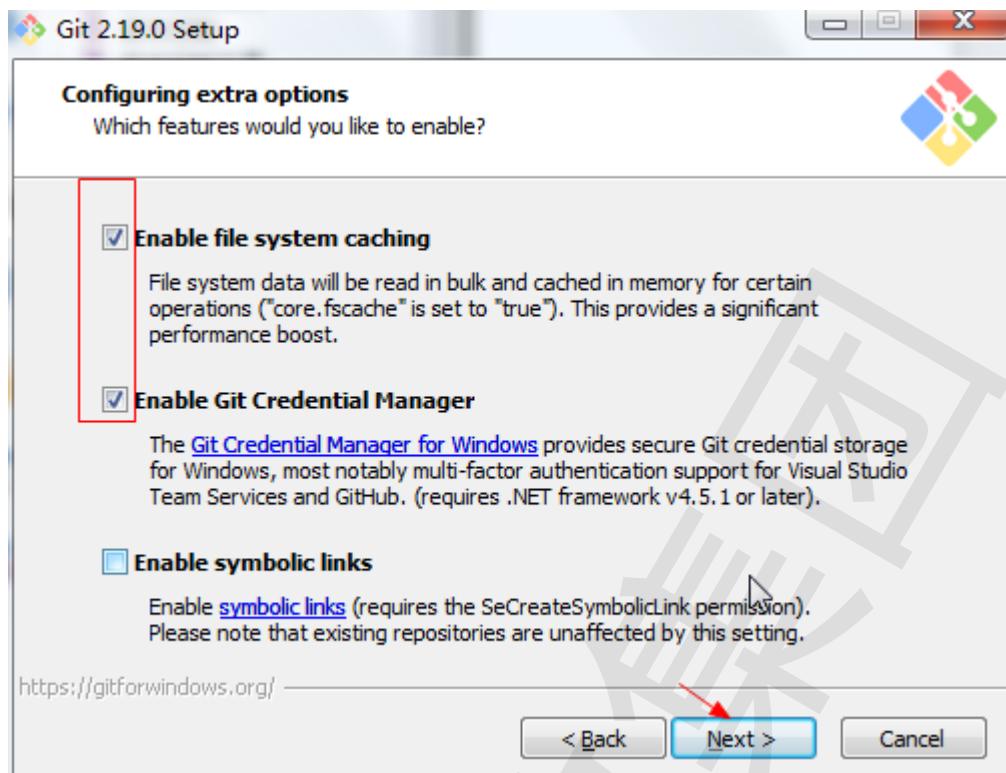
3.2 Git 安装

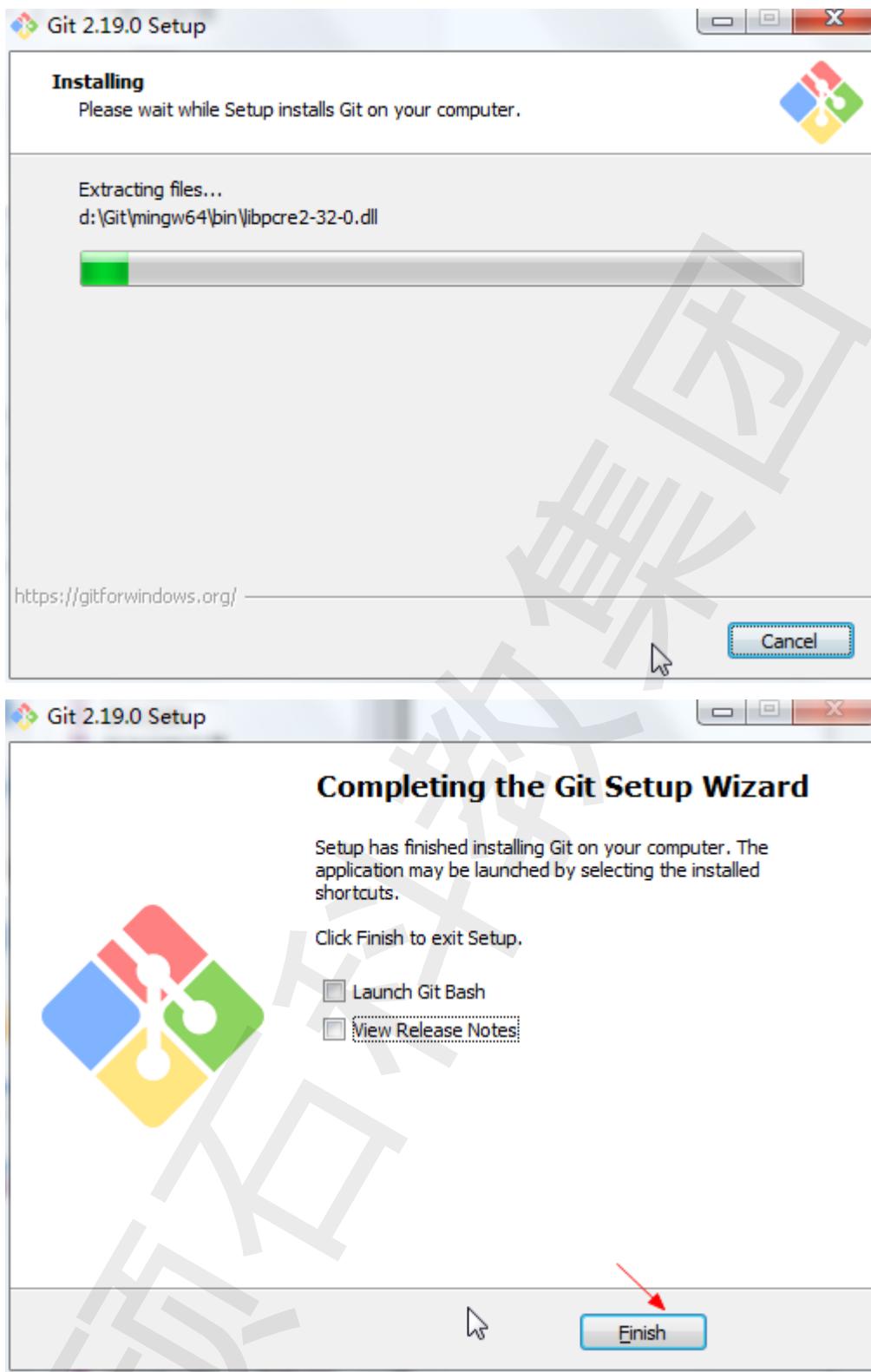












3.3 Git 本地工作区域

请注意，接下来要讲的概念非常重要。

- 对于任何一个文件，在 Git 内都只有三种区域：**工作区，暂存区和本地仓库**。
 - 工作区：表示新增或修改了某个文件，但还没有提交保存；

- 暂存区：表示把已新增或修改的文件，放在下次提交时要保存的清单中；
- 本地仓库：文件已经被安全地保存在本地仓库中了。



4 Git与代码托管平台

4.1 Git 与 GitHub比较

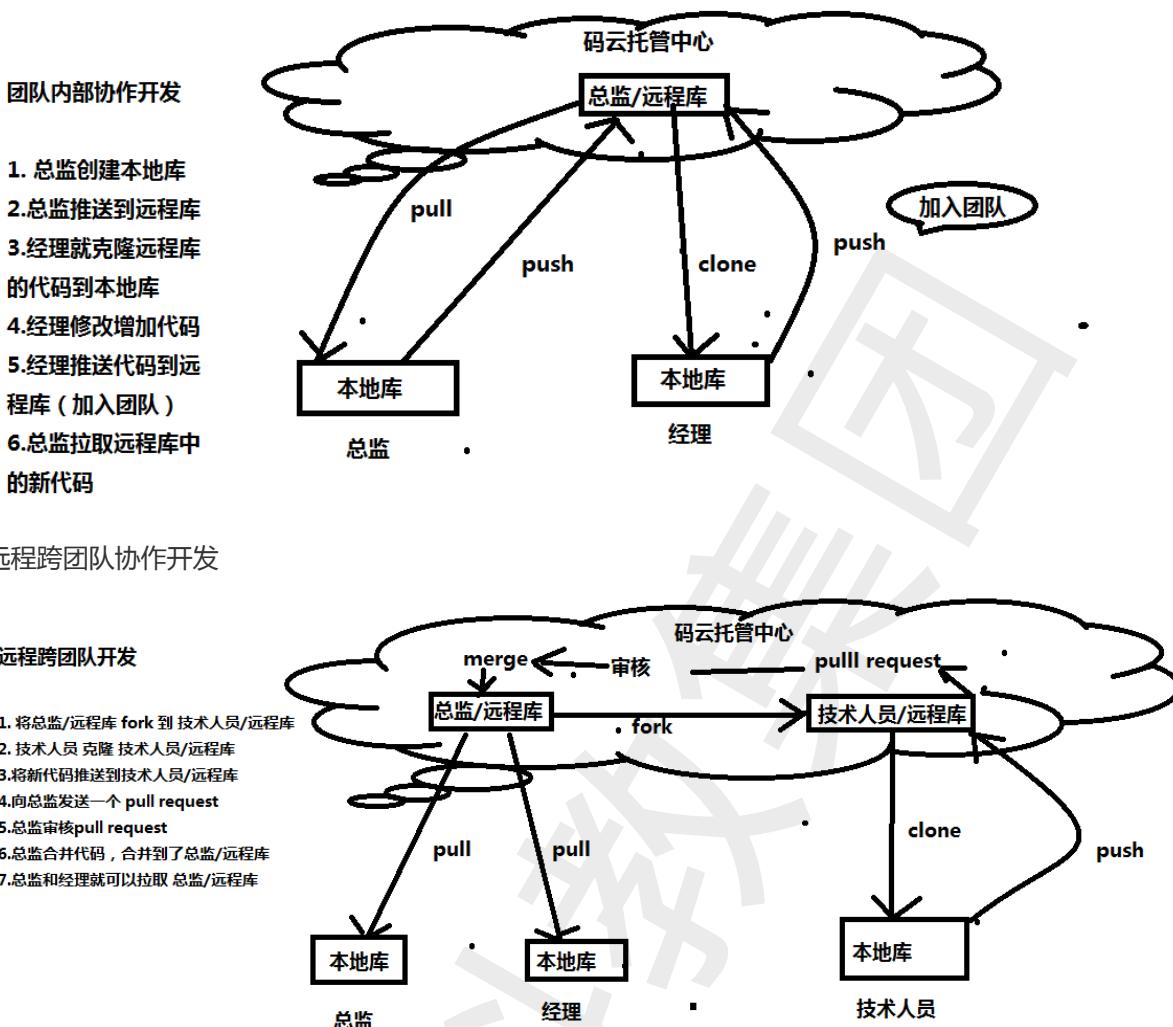
- Git：
是一个版本管理工具，只在本地使用的一个版本管理工具，其作用就是可以让你更好的管理你的程序，比如你原来提交过的内容，后面虽然修改过，但是通过git这个工具，可以把你原来提交的内容重现出来，这样对于你后来才意识到的一些错误进行更改，进行还原。
- GitHub (官网: <https://github.com/>)
是一个基于Git的远程代码托管平台（网站），可以在github上建立一个远程库，可以将本地库的代码提交到远程库，这样你的每次提交，别人也都可以看到你的代码，同时别人也可以帮你修改你的代码，这种开源的方式非常方便程序员之间的交流和学习。

4.2 代码托管平台

- 局域网：
 - GitLab (可自行搭建)
- 外网环境：
 - GitHub
 - 码云

4.3 本地库和远程库

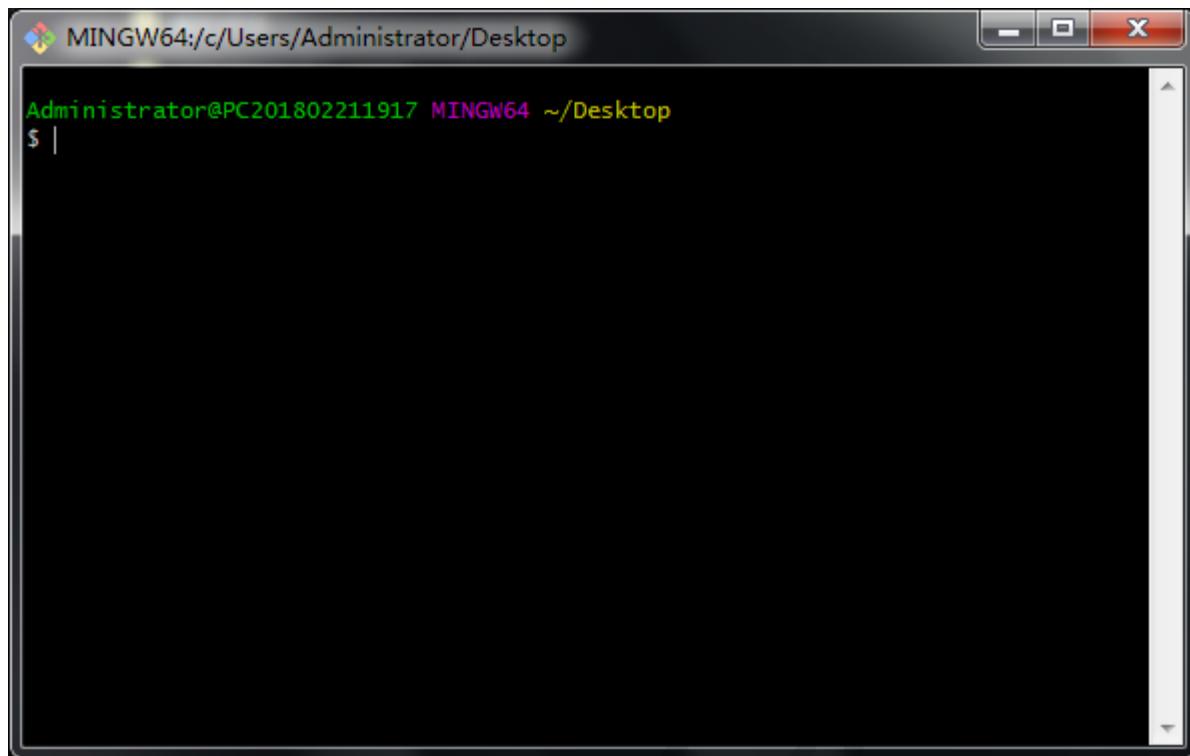
- 团队内部协作开发



5. Git 命令行操作

5.1 打开命令行窗口

- 安装Git后，在资源管理器的空白处，单击鼠标右键打开窗口，点击 `Git Bash Here`，打开Git命令行窗口，在窗口中可直接使用Linux命令操作：



5.2 初始化Git本地库

- 命令：`git init`
- 效果：

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01
$ pwd
/d/gitStudy/git01    查看下当前所在目录

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01
$ git init 在当前目录下，初始化本地Git仓库
Initialized empty Git repository in D:/gitStudy/git01/.git/

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ ll -A 由于.git是隐藏文件，所以需要指定 -A 进行查看
total 4
drwxr-xr-x 1 Administrator 197121 0 七月  5 13:37 .git/

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ ll .git/    查看.git核心配置目录
total 7
-rw-r--r-- 1 Administrator 197121 130 七月  5 13:37 config
-rw-r--r-- 1 Administrator 197121  73 七月  5 13:37 description
-rw-r--r-- 1 Administrator 197121   23 七月  5 13:37 HEAD
drwxr-xr-x 1 Administrator 197121     0 七月  5 13:37 hooks/
drwxr-xr-x 1 Administrator 197121     0 七月  5 13:37 info/
drwxr-xr-x 1 Administrator 197121     0 七月  5 13:37 objects/
drwxr-xr-x 1 Administrator 197121     0 七月  5 13:37 refs/
```

- 注意：`.git` 目录中存放的是本地库相关核心配置文件，也不要随意删除与修改
- `.git` 目录仓库目录说明：
 - hooks目录：脚本文件的目录。
 - info目录：保存了不希望在 `.gitignore` 文件中管理的忽略模式的全局可执行文件

- logs目录：日志目录
- objects目录：存储所有数据内容
- refs目录：存储指向数据（分支）的提交对象的指针
- config文件包含了项目特有的配置选项
- description文件仅供 GitWeb 程序使用
- HEAD文件指向当前分支

5.3 设置签名信息

- 作用：只为区分不同开发人员的身份信息
- 格式：

用户名：mengxuegu
Email: mengxuegu@163.com
- 注意：
 - 这里的签名信息和登录远程库的帐号和密码没有任何关系(码云，Github)
- 命令：
 - 项目级别/仓库级别：仅在当前目录的本地Git仓库范围内有效
 - `git config user.name mengxuegu_pro`
 - `git config user.email mengxuegu888@163.com`
 - 签名信息保存位置：`./.git/config` 文件中

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ pwd
/d/gitStudy/git01

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git config user.name mengxuegu_pro

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git config user.email mengxuegu888@163.com

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ cat .git/config
[core]
  repositoryformatversion = 0
  filemode = false
  bare = false
  logallrefupdates = true
  symlinks = false
  ignorecase = true
[user]
  name = mengxuegu_pro
  email = mengxuegu888@163.com
```

- 系统用户级别：登录当前操作系统的用户范围
 - `git config --global user.name mengxuegu_glo`
 - `git config --global user.email mengxuegu666@163.com`
 - 签名信息保存位置：`~/.gitconfig`

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git config --global user.name mengxuegu_glo

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git config --global user.email mengxuegu666@163.com

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ cd ~
Administrator@PC201802211917 MINGW64 ~
$ pwd
/c/Users/Administrator

Administrator@PC201802211917 MINGW64 ~
$ cat .gitconfig
[user]
    name = mengxuegu_glo
    email = mengxuegu666@163.com
    signingkey = ""
[core]
    excludesfile = D:\\\\Documents\\\\gitignore_global.txt
    autocrlf = true
```

- 级别优先级：

- 就近原则：项目级别 优先于 系统用户级别
- 如果只有 系统用户级别 的签名，则采用 系统用户级别 的签名信息
- 二者都不存在是不允许的。

5.4 Git 基本操作

5.4.1 查看状态

- 用于查看工作区、暂存区的状态

```
1 $ git status
2
3 On branch master # 默认在master(主干)分支上
4 No commits yet # 当前没有任何的提交
5 nothing to commit (create/copy files and use "git add" to track)
6 # 没有什么需要提交的 ( 创建/复制文件 , 使用“git add”命令可追踪 , 也就是用git去管理文件 )
```

- 根据状态提示，往仓库中创建一个 demo01.txt 文件，文件保存一些内容（按 i 插入内容，按 :wq 保存并退出，按 ':q!' 不保存强制退出）：

```
1 $ vim demo01.txt
```

- 再 git status 查看状态提示 untracked files (有未追踪文件)：

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ vim demo01.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ ls
demo01.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    demo01.txt

nothing added to commit but untracked files present (use "git add" to track)
```

5.4.2 添加到暂存区

- 将工作区的“新建/修改”添加到暂存区

- 命令：`git add <file name>`

```
$ git add demo01.txt
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   demo01.txt
```

- 恢复，不放到暂存区

- 命令：`git rm --cached <file name>`

```
$ git rm --cached demo01.txt
rm 'demo01.txt'

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    demo01.txt

nothing added to commit but untracked files present (use "git add" to track)
```

5.4.3 提交到本地库

- 将暂存区的内容提交到本地库

- 命令 : `git commit [-m "提交说明信息"] <file name>`

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git commit -m demo01.txt
[master (root-commit) 7630922] demo01.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 demo01.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

- 修改 demo01.txt 文件内容，再查看状态：

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   demo01.txt

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git add demo01.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   demo01.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git commit -m "second commid" demo01.txt
[master 2f21f2a] second commid
 1 file changed, 1 insertion(+)
```

5.4.4 查看版本历史记录

- 显示最详细的日志信息

- 命令 : `git log`

```
$ git log
commit a23672a0ab60cdffcf83c0abe4e9b10d970103fd (HEAD -> master)
Author: mengxuegu_pro <mengxuegu888@163.com>
Date:   Fri Oct 9 17:18:53 2099 +0800

    insert eeeee

commit 0e4edd5c19b2ef1bcd7acdd1fd11fc063cea441
Author: mengxuegu_pro <mengxuegu888@163.com>
Date:   Fri Oct 9 17:18:35 2099 +0800

    insert ddddd

commit c92ed489890ea8e8c70ca3ba5d811649d0a4137d
Author: mengxuegu_pro <mengxuegu888@163.com>
Date:   Fri Oct 9 17:18:16 2099 +0800

    insert cccc

commit 86d64dd1f66a7fb34416e65f9ad7188ecd551f67
Author: mengxuegu_pro <mengxuegu888@163.com>
Date:   Fri Oct 9 17:17:50 2099 +0800

    insert bbbbb
.
```

- 如果内容太长, 多屏显示控制方式 :
 - 空格键: 向下查看
 - b : 向上查看
 - q : 退出查看
- 以漂亮的格式显示 : 即每条日志只显示一行

- 命令 : `git log --pretty=oneline`

```
$ git log --pretty=oneline
a23672a0ab60cdffcf83c0abe4e9b10d970103fd (HEAD -> master) insert eeeee
0e4edd5c19b2ef1bcd7acdd1fd11fc063cea441 insert ddddd
c92ed489890ea8e8c70ca3ba5d811649d0a4137d insert cccc
86d64dd1f66a7fb34416e65f9ad7188ecd551f67 insert bbbbb
4d4c54c75eccad68ce3d351525fa1b16556aad28 first demo01.txt
```

- 简约的格式显示 :
 - 命令 : `git log --oneline`
- 显示回滚版本步数[推荐] :
 - 命令 : `git reflog`

HEAD@{回滚对应版本, 底层操作需要移动多少步}

```
$ git reflog
a23672a (HEAD -> master) HEAD@{0}: commit: insert eeeee
0e4edd5 HEAD@{1}: commit: insert dddddd
c92ed48 HEAD@{2}: commit: insert cccc
86d64dd HEAD@{3}: commit: insert bbbbb
4d4c54c HEAD@{4}: commit (initial): first demo01.txt
```

5.4.5 前进后退版本

通过HEAD指针来移动回滚版本



```
$ git reflog
a23672a (HEAD -> master) HEAD@{0}: commit: insert eeeee
0e4edd5 HEAD@{1}: commit: insert dddddd
c92ed48 HEAD@{2}: commit: insert cccc
86d64dd HEAD@{3}: commit: insert bbbbb
4d4c54c HEAD@{4}: commit (initial): first demo01.txt
```

- 基于索引值操作[推荐方式]

- 命令 : `git reset --hard <局部索引值>`
 - 举例 : `git reset --hard 64d3d2a`

- 使用`^`(异或)符号 : 只能后退

- 命令 : `git reset --hard HEAD^`
 - **注 : 一个^表示后退一步 , n个表示后退n步**

- 使用`~`符号 : 只能后退

- 命令 : `git reset --hard HEAD~n`
 - **注 : n指定步数 , 表示后退n步**

5.4.6 删 除文件并恢复

- 前提 : 删除文件前 , 此文件需要已经提交过本地库 , 才可恢复
- 删 除 : `rm 文件名.txt`
- 命令 : `git reset --hard <历史记录索引值>`
 - 删除操作已经提交到本地库 : 指针位置指向历史记录
 - 删除操作尚未提交到本地库 : 无法恢复

5.4.7 对比文件差异

- 将工作区中的文件和暂存区进行比较
- 命令 : `git diff <文件名>`
 - 举例 : 向`apply.txt`文件添加了两行 , 使用 `git diff apple.txt` 查看

```
Administrator@PC201802211917 MINGW64 /d/gitstudy/git01 (master)
$ git diff apply.txt
warning: LF will be replaced by CRLF in apply.txt.
The file will have its original line endings in your working directory
diff --git a/apply.txt b/apply.txt
index 354e3b6..436a567 100644
--- a/apply.txt
+++ b/apply.txt
@@ -1 +1,3 @@
 add apply01.....
+
+add apply02. . . .

```

工作区中新添加的内容

- 将工作区中的文件和本地库历史记录比较

- 命令 : `git diff <本地库中历史版本> <文件名>`

- 举例 :

```
Administrator@PC201802211917 MINGW64 /d/gitstudy/git01 (master)
$ git reflog
16cbbc2 (HEAD -> master) HEAD@{0}: commit: add apply.....
86d64dd HEAD@{1}: reset: moving to 86d64dd
0e4edd5 HEAD@{2}: reset: moving to 0e4edd5
4d4c54c HEAD@{3}: reset: moving to HEAD~1
86d64dd HEAD@{4}: reset: moving to HEAD^^
0e4edd5 HEAD@{5}: reset: moving to HEAD^
a23672a HEAD@{6}: reset: moving to a23672a
c92ed48 HEAD@{7}: reset: moving to c92ed48
a23672a HEAD@{8}: commit: insert eeeee
0e4edd5 HEAD@{9}: commit: insert ddddd
c92ed48 HEAD@{10}: commit: insert cccc
86d64dd HEAD@{11}: commit: insert bbbbb
4d4c54c HEAD@{12}: commit (initial): first demo01.txt

Administrator@PC201802211917 MINGW64 /d/gitstudy/git01 (master)
$ git diff 16cbbc2 apply.txt
warning: LF will be replaced by CRLF in apply.txt.
The file will have its original line endings in your working directory
diff --git a/apply.txt b/apply.txt
index 436a567..e98ff8e 100644
--- a/apply.txt
+++ b/apply.txt
@@ -1,3 +1,5 @@
 add apply01.....
+
+add apply02. . . .
+
+add apply03!!!!!
```

本地库中的内容

工作区中新增的内容

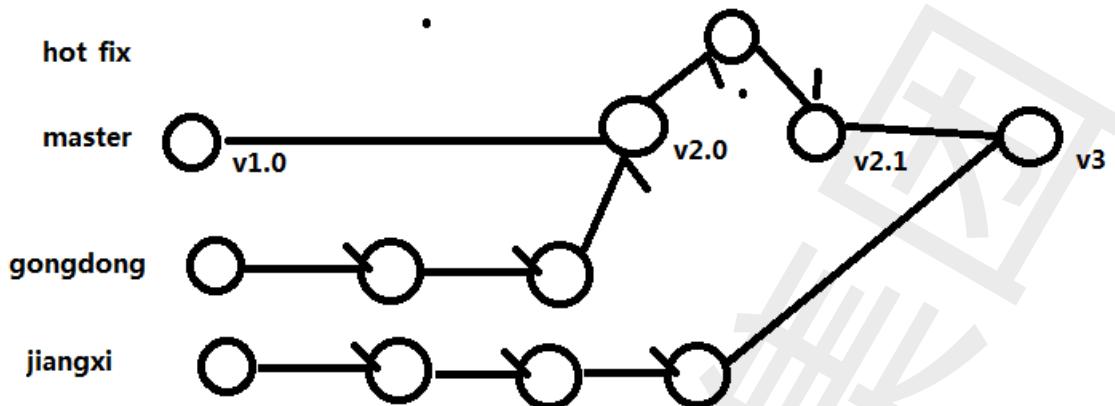
- 不带文件名比较多个文件

6. Git 分支管理

6.1 什么是Git分支？

在版本控制过程中，使用多条线同时推进多个任务

如下图：



6.2 Git分支的好处？

- 同时并行推进多个功能开发，提高开发效率
- 各个分支在开发过程中，如果某一个分支开发失败，不会对其他分支有任何影响。失败的分支删除重新开始即可。

6.3 Git分支操作

- 查看分支：`git branch -v`
- 创建分支：`git branch 新分支名`
- 删除分支(删除的分支不是当前正在打开的分支)：`git branch -d 分支名`
- 切换分支：`git checkout 分支名`
- 合并分支：
 - 第1步：切换到接受修改的分支上
 - 命令：`git checkout 需要接受的分支名`
 - 第2步：执行`merge`命令
 - 命令：`git merge 有新内容的分支名`
- 解决冲突：
 - 冲突的表现，如下图：

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master)
$ git merge hot_fix
Auto-merging apply.txt
CONFLICT (content): Merge conflict in apply.txt
Automatic merge failed; fix conflicts and then commit the result.

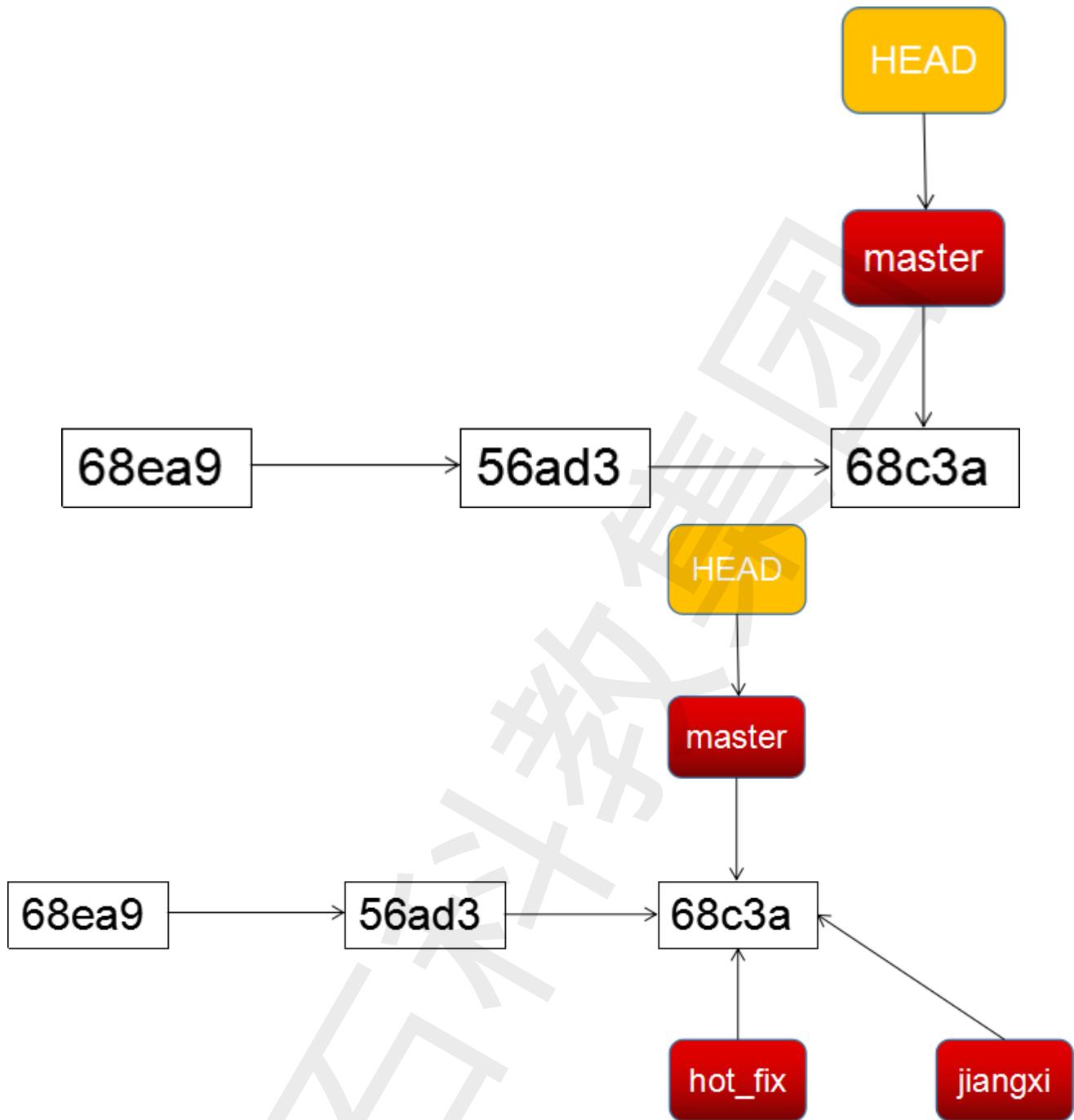
Administrator@PC201802211917 MINGW64 /d/gitStudy/git01 (master|MERGING)
$ cat apply.txt
add apply01.....
add apply02.....
my branch is hot_fix update
<<<<< HEAD

                                     当前分支上的内容
aaaaaaaaaaaaaaaaaaaaaaaaaa
=====
11111111111111111111
22222222222222222222      hot_fix 分支上的内容 ,
hot_fix update11111
>>>>> hot_fix
```

- 冲突的解决：
 - 第1步：编辑文件，删除特殊符号
 - 第2步：把文件修改到满意为止，保存退出
 - 第3步：git add 文件名
 - 第4步：git commit -m "日志信息"
 - 注意：此时 commit 后面一定不要有文件名

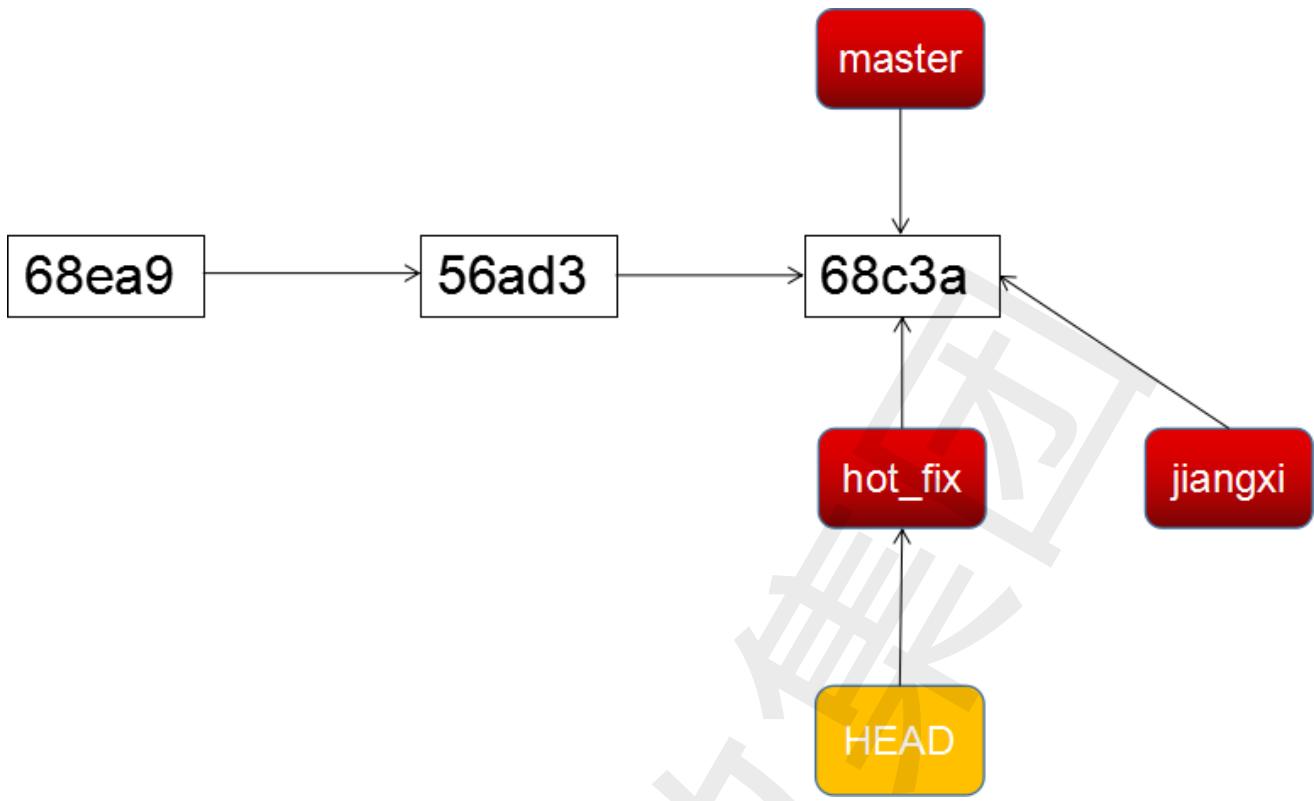
7. Git分支管理机制

7.1 创建分支

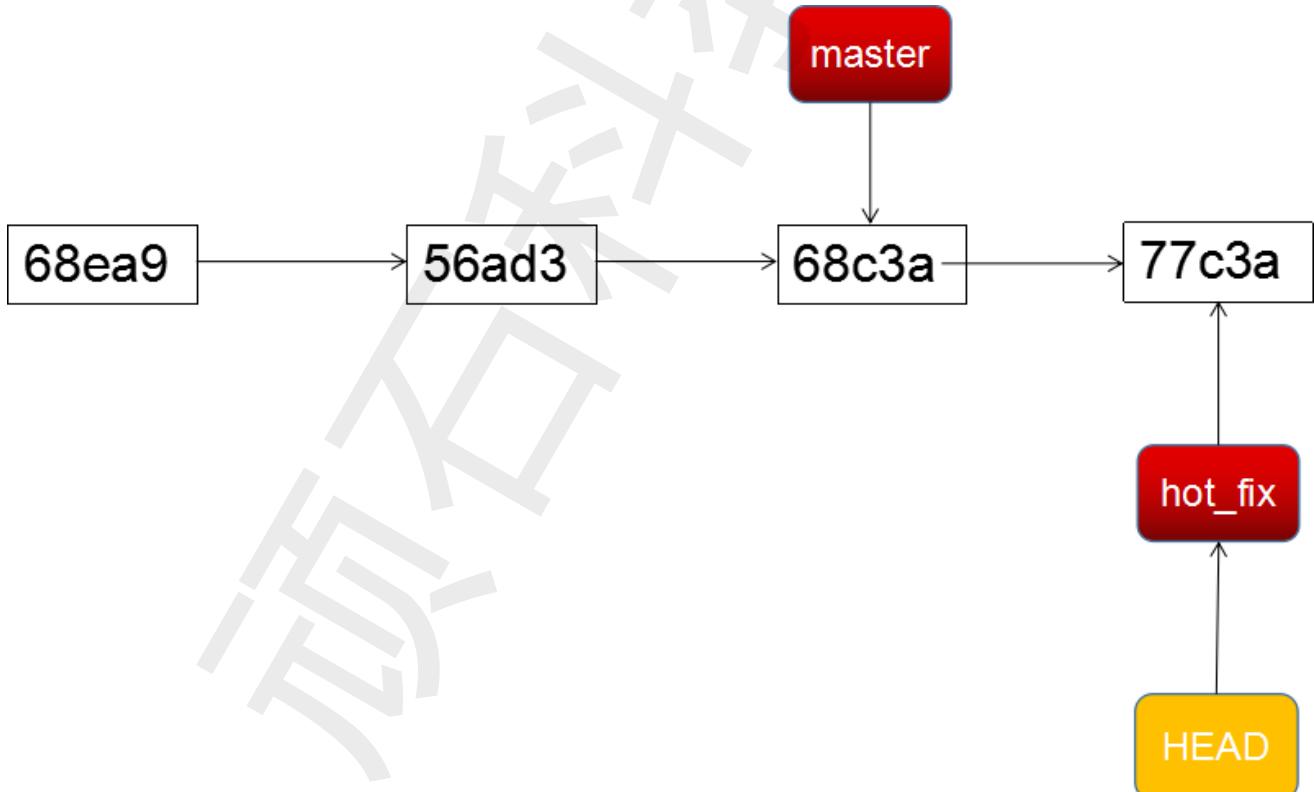


7.2 切换分支

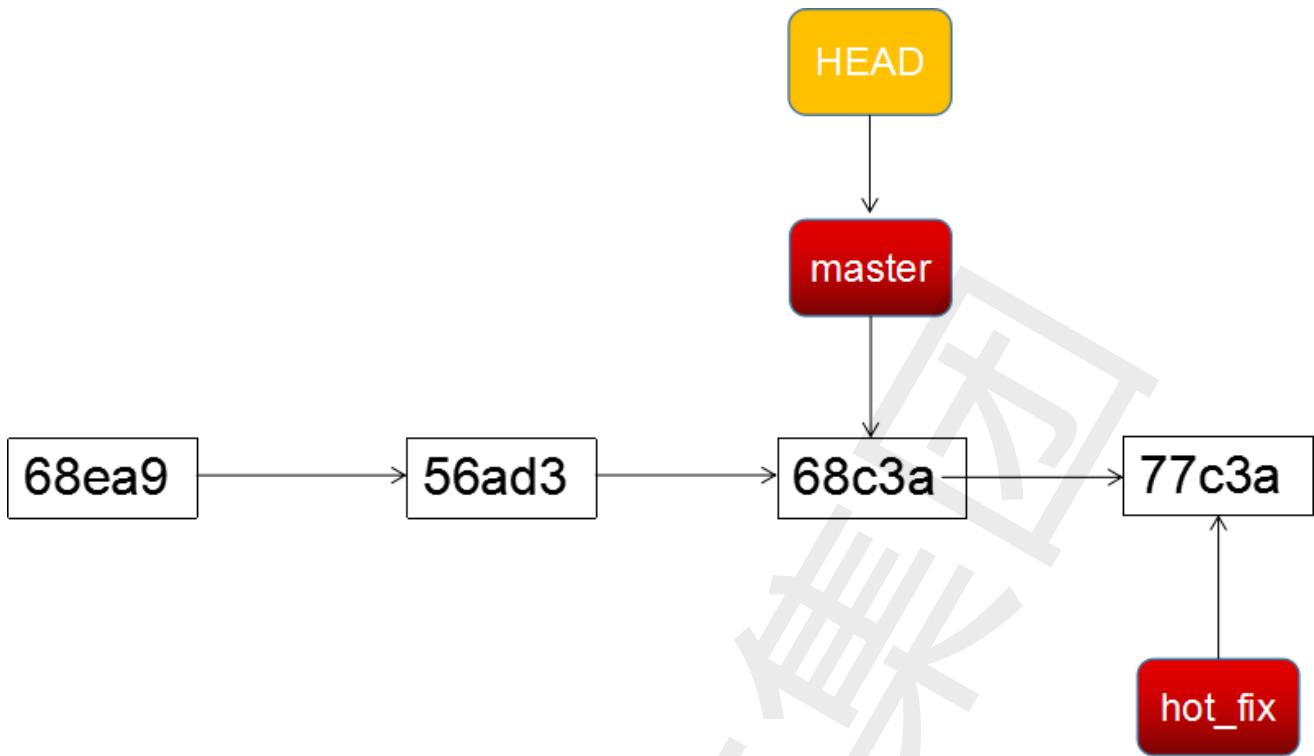
- 创建了两个分支，当前指向hot_fix



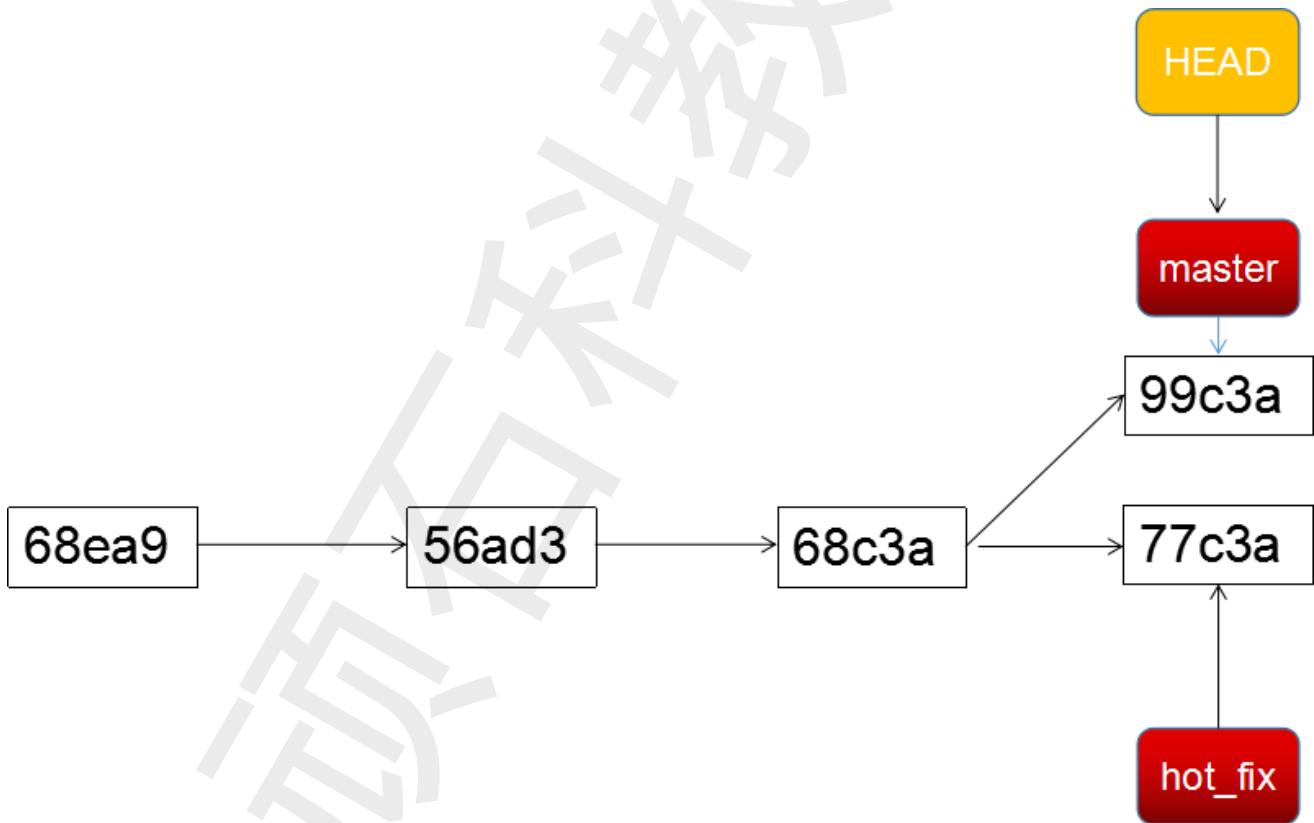
- 在hot_fix更新了新的版本，



- 由hot_fix分支切换回master分支



- 在master分支上做了更新



8. 码云代码托管中心

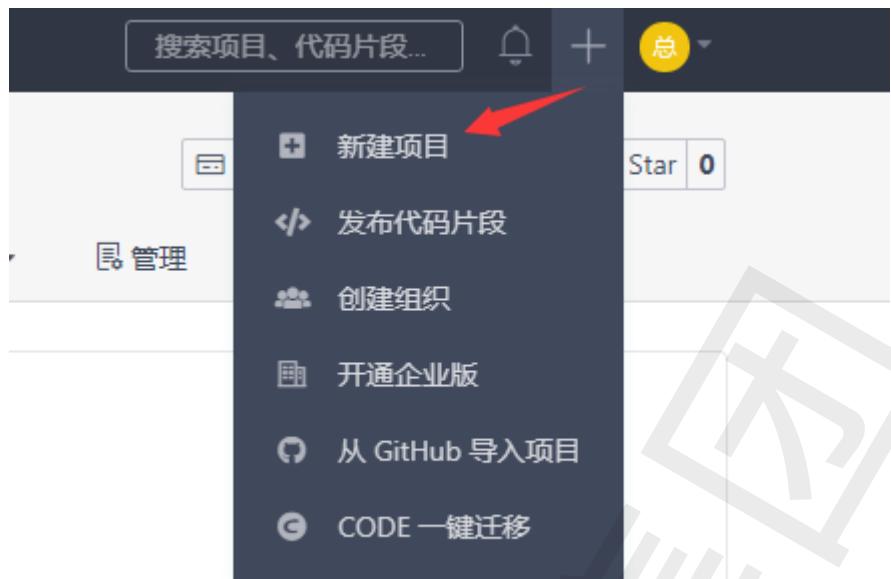
8.1 注册帐号

- 码云首页：<https://gitee.com/>
- 码云注册页面：<https://gitee.com/signup>
- 下面提供了3个帐号：

总	姓名：总监 个性域名：mxg6666 帐号： mengxuegu666@163.com
经	姓名：经理 个性域名：mxg888 帐号： mengxuegu888@163.com
开	姓名：开发人员 个性域名：mxg999 帐号： mengxuegu999@163.com

8.2 创建远程库（项目）

- 点击右上角 **+**，选择 **新建项目**



- 填写项目信息

新建项目

项目名称
远程仓库名
mengxuegu

归属
总监

路径
<https://gitee.com/mxg6666/> mengxuegu

项目介绍 非必填
用简短的语言来描述一下这个项目吧
项目介绍

是否开源 可见性
 公开 私有

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 [①](#)
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 [①](#)
 使用 Pull Request 模板文件初始化这个项目 [①](#)

导入已有项目

创建 点击创建，即可创建成功

- 效果

总监 / mengxuegu

捐赠 0 | Unwatch 1 | ⚙

代码 Issues 0 Pull Requests 0 附件 0 Wiki 0 服务 管理

快速设置—如果你知道该怎么操作，直接使用下面的地址

HTTPS SSH <https://gitee.com/mxg6666/mengxuegu.git>

我们强烈建议所有的git仓库都有一个 README, LICENSE, .gitignore 文件

Git入门 ? 查看 帮助 , Visual Studio / TortoiseGit / Eclipse / Xcode 下如何连接本站. 如何导入项目

简易的命令行入门教程:

Git 全局设置:

```
git config --global user.name "总监"
git config --global user.email "mengxuegu666@163.com"
```

创建 git 仓库:

```
mkdir mengxuegu
cd mengxuegu
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitee.com/mxg6666/mengxuegu.git
git push -u origin master
```

已有项目?

```
cd existing_git_repo
git remote add origin https://gitee.com/mxg6666/mengxuegu.git
git push -u origin master
```

8.3 创建本地仓库

```
1 mkdir mengxuegu
2 cd mengxuegu
3 git init # 初始化仓库
4 vim demo1.txt # 新增文件，按 i 开始编辑，ctrl+c退出编辑，:wq保存并退出，:q!不保存退出
5 git add demo1.txt # 添加到暂存区
6 git commit -m "fisrt commmit" demo1.txt # 提交到本地仓库
```

8.4 创建远程库地址别名

- 查看当前所在本地仓库下的所有远程地址别名

```
1 git remote -v
```

- 创建远程库地址别名

```
1 git remote add 别名 远程地址
```

- 演示：

```
1 git remote add origin https://gitee.com/mxg6666/mengxuegu.git
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git remote add origin https://gitee.com/mxg6666/mengxuegu.git

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git remote -v
origin https://gitee.com/mxg6666/mengxuegu.git (fetch) 取回
origin https://gitee.com/mxg6666/mengxuegu.git (push) 推送
```

8.5 push 推送本地库到远程库

- 命令：

```
1 git push [远程库别名] [分支名]
```

- 演示：将master分支推送到 origin 远程库

```
1 git push origin master
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git push origin master
remote: You do not have permission to push to the repository
fatal: Authentication failed for 'https://gitee.com/mxg6666/mengxuegu.git/'

如果出现上面错误，再次命令
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git push origin master
```

- 如果上面报没权限错误，则重新输入 `git push origin master`，就会弹出以下窗口，输入 码云平台 的用户名和密码就行。



- 推送成功

```
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 228 bytes | 228.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Powered by Gitee.com
To https://gitee.com/mxg6666/mengxuegu.git
 * [new branch]      master -> master
新建远程master分支 本地mater 到 远程master 分支
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
```

8.6 克隆远端库

经理也要同时参与 `mengxuegu` 项目的开发，就需要把远端库中克隆下来

- 经理先创建一个本地放目录

```
1 cd ..
2 mkdir mengxuegu_jl
3 cd mengxuegu_jl
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ cd ..

Administrator@PC201802211917 MINGW64 /d/gitStudy
$ pwd
/d/gitStudy

Administrator@PC201802211917 MINGW64 /d/gitStudy
$ mkdir mengxuegu_jl

Administrator@PC201802211917 MINGW64 /d/gitStudy
$ cd mengxuegu_jl

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_jl
$ |
```

- 克隆命令

```
1 git clone 远程地址
```

- 演示

```
1 git clone https://gitee.com/mxg6666/mengxuegu.git
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ cd .. ←

Administrator@PC201802211917 MINGW64 /d/gitStudy
$ mkdir mengxuegu_jl

Administrator@PC201802211917 MINGW64 /d/gitStudy
$ cd mengxuegu_jl/

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_jl
$ git clone https://gitee.com/mxg6666/mengxuegu.git
Cloning into 'mengxuegu'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
```

- 克隆效果

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

8.7 修改克隆文件再提交

- 经理修改 `demo1.txt` 文件进行提交本地库

```
1 vim demo1.txt
2 git add demo1.txt
3 git commit -m "jl commit" demo1.txt
```

- 先把本地缓存的git帐号清除掉，不然会以原来登录的帐户提交（总监），而不是经理帐号

- 进入控制面板：按 `Ctrl + R` 打开运行窗口，输入 `control`

- 找到 凭据管理器



- 找到 gitee.com , 展开 , 然后删除



- 经理再将本地库提到远程库，但是没有权限提交

```
1 git push origin master
```

```
Administrator@PC201802211917:~ /d/gitStudy/mengxuegu_jl/mengxuegu (master)
$ git push origin master
remote: You do not have permission to push to the repository
fatal: Authentication failed for 'https://gitee.com/mxg6666/mengxuegu.git/'
```

8.8 添加项目成员

总览 / mengxuegu

代码 Issues Pull Requests 附件 Wiki 统计 服务 管理

项目成员(1)

管理员(1) 1

所有 1

总监 项目拥有者 我自己 mxg6666 (mengxuegu666@163.com)

管理员 0

开发者 0

添加项目成员 邀请用户 邀请其他项目成员

邀请用户

链接邀请 直接添加 通过项目邀请成员

直接添加码云用户

权限

经理 mxg888

mengxuegu888@163.com

添加

• 点击提交

邀请详情

邀请用户后，受邀方接受邀请后将直接加入项目
若邀请的用户为非码云站内用户，则系统将发送邀请链接至目标邮箱，链接有效期为 3 天

序号	用户邮箱	用户昵称	个性域名	项目权限(双击可更改)	用户状态	操作
1	****	经理	mxg888	开发者	码云用户	移除

取消 提交

- 已将邀请链接以私信的形式发送至 经理

邀请详情

序号	用户邮箱	用户昵称	个性域名	项目权限	用户状态	邀请结果
1	****	经理	mxg888	开发者	码云用户	✓ 已将邀请链接以私信的形式发送至 经理

关闭

- 经理 登录自己的 码云，找到 私信，然后访问邀请链接



- 接受邀请后，再提交就可以

```
1 git push origin master
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ git push origin master
remote: You do not have permission to push to the repository
fatal: Authentication failed for 'https://gitee.com/mxg6666/mengxuegu.git/'

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 261 bytes | 261.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Powered by Gitee.com
To https://gitee.com/mxg6666/mengxuegu.git
  b01551e..aaba9ef master -> master
```

8.9 pull 拉取操作

- pull 拉取操作其实是两步：

```
pull = fetch + merge
```

- fetch 操作：只把远程库中的内容下载到本地，但是没有改本地工作区的文件。

```
1 git fetch 远程库地址别名 远程分支名
```

- 演示：

```
1 git fetch origin master
2
3 cat demo1.txt # 查看并没有更改本地工作区的文件
4
5 git checkout origin/master # 切换远程分支对比下内容
6 cat demo1.txt      # 远程分支 内容是不一样的
7
8 git checkout master #切换回本地master分支
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ cd ../../mengxuegu          切换到总监目录下

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ pwd
/d/gitStudy/mengxuegu

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git fetch origin master
From https://gitee.com/mxg6666/mengxuegu
 * branch            master      -> FETCH_HEAD

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ cat demo1.txt
aaaaaaaaaa

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git checkout origin/master    切换，预览远程库中的内容
Note: checking out 'origin/master'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at f081428 jl commit

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu ((f081428...))
$ cat demo1.txt
aaaaaaaaaa

jl add content.......

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu ((f081428...))
$ git checkout master           切换回本地master分支上
Previous HEAD position was f081428... jl commit
Switched to branch 'master'

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ cat demo1.txt
aaaaaaaaaa

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ |
```

- `merge` 操作：把远程代码合并到本地代码中

```
1 git merge 远程库地址别名/远程分支名
```

- 演示

```
1 git merge origin/master
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git merge origin/master
Updating 731f148..f081428
Fast-forward
 demo1.txt | 2 ++
 1 file changed, 2 insertions(+)

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ cat demo1.txt
aaaaaaaaaa

j1 add content.......

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ |
```

- `pull` 操作：针对没有冲突的情况，不分上面两个步骤，可以直接使用 `pull` 提取操作，

```
1 git pull 远程库地址别名 远程分支名
```

◦ 演示

```
1 $ pwd
2 /d/gitstudy/mengxuegu_j1/mengxuegu
3 $ vim demo1.txt
4 $ git commit -m "j1 cmmmit" demo1.txt
5 $ git push origin master
6
7 $ cd ../../mengxuegu
8 Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
9 $ git pull origin master # 直接拉取，不分两步 fetch,merge
10 $ cat demo1.txt
11 hello, 我是总监
12 hi, 我是经理，我加入项目组来啦....
13 haha, 我是经理，我又来了。。。
```

```
Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ cd ../mengxuegu_j1/mengxuegu/

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ ls
demo1.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ vim demo1.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ git add demo1.txt

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ git commit -m "j1 second commit" demo1.txt
[master 339fe76] j1 second commit
 1 file changed, 2 insertions(+)

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 282 bytes | 141.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Powered by Gitee.com
To https://gitee.com/mxg6666/mengxuegu.git
  f081428..339fe76 master -> master

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu_j1/mengxuegu (master)
$ cd ../../mengxuegu

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
```

```
$ pwd
/d/gitStudy/mengxuegu

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ git pull origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://gitee.com/mxg6666/mengxuegu
 * branch            master      -> FETCH_HEAD
   f081428..339fe76  master      -> origin/master
Updating f081428..339fe76
Fast-forward
 demo1.txt | 2 ++
 1 file changed, 2 insertions(+)

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ cat demo1.txt
aaaaaaaaaa

j1 add content.....
j1 second write bbbbb

Administrator@PC201802211917 MINGW64 /d/gitStudy/mengxuegu (master)
$ |
```

8.10 解决冲突

- 什么情况有冲突？

在企业中团队协作开发时，当多个人同时修改同一个文件，同一行代码时，就会产生冲突。

只有先推送的那个人才可以正常推送，后面那个人与它出现冲突的代码，是没有办法推送的，必须先拉取下来，然后自己手动解决冲突后才可进行推送。

- 总监 修改第2行的内容，然后提交到远程库

```
1 Administrator@PC201802211917 MINGW64 /d/gitstudy/mengxuegu (master)
2 vim demo1.txt
3
4 git commit -m 'zj update1' demo1.txt
5
6 git push origin master
```

- 经理 不知道总监修改了文件，所以没有拉取，直接修改了自己本地库的第2行的内容，

然后直接提交到远程库，发现提交报错，

```
1 cd ../mengxuegu_jl/mengxuegu/
2 vim demo1.txt
3 git commit -m "jl update111" demo1.txt
4
5 $ git push origin master
6 To https://gitee.com/mxg6666/mengxuegu.git
7 ! [rejected]          master -> master (fetch first)
8 error: failed to push some refs to 'https://gitee.com/mxg6666/mengxuegu.git'
9 hint: Updates were rejected because the remote contains work that you do
10 hint: not have locally. This is usually caused by another repository pushing
11 hint: to the same ref. You may want to first integrate the remote changes
12 hint: (e.g., 'git pull ...') before pushing again.
13 hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- 解决冲突：

- 上面有冲突时，需要先 `git pull` 拉取远程代码，然后 `vim` 查看修改文件，再手动解决冲突，保留需要的

```
1 git pull origin master
2 vim demo1.txt
```

```
MINGW64:/d/gitStudy/mengxuegu_jl/mengxuegu
hello, 我是总监
<<<<< HEAD
hi, 我是经理, 我加入项目组来啦.... 开始敲代码helloworld
=====
hi, 我是经理, 我加入项目组来啦.... zzzzzjjjjjj HEAD 是你修改的文件内容
>>>> 52bd5f05e371e07228c075a030bb72769232c486 这是远水库拉取下来的冲突代码
haha, 我是经理, 我又来了... |
~
```

- 添加到本地库和远端库

```

1 git add demo1.txt
2 git commit -m "jl resolve conflict"
3 git status
4 On branch master
5 Your branch is ahead of 'origin/master' by 2 commits.
6   (use "git push" to publish your local commits)
7
8 nothing to commit, working tree clean
9
10 git push origin master

```

```

$ git status
on branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Administrator@PC201802211917 MINGW64 /d/gitstudy/mengxuegu_jl/mengxuegu (master)
$ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 543 bytes | 271.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Powered by Gitee.com
To https://gitee.com/mxg6666/mengxuegu.git
  52bd5f0..9a92283 master -> master

```

8.11 fork 跨团队协作

- 开发人员入场，访问 <https://gitee.com/mxg6666/mengxuegu>，点击右上角的 Fork



The screenshot shows the Gitee project page for 'mengxuegu'. At the top right, there is a 'Fork' button with a red border and a red arrow pointing to it. Below the header, there's a summary bar with metrics: 8 次提交, 1 个分支, 0 个标签, 0 个发行版, and 1 位贡献者. Underneath is a navigation bar with tabs for '代码', 'Issues 0', 'Pull Requests 0', '附件 0', 'Wiki 0', '统计', and '服务'. The main content area shows a commit history with one entry from 'alan_glb' 12 minutes ago: 'jl resolve conflict' on file 'demo1.txt'.

- fork之后，发现域名变了，和项目名下面多了一行描述

https://gitee.com/mxg999/mengxuegu

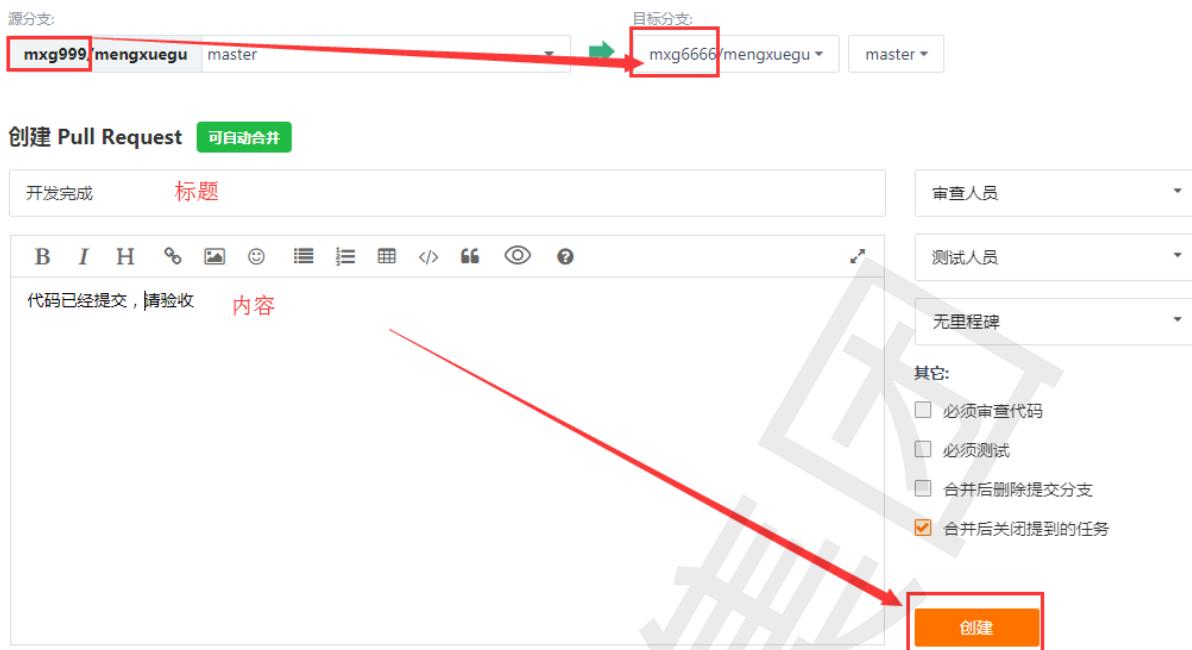
The screenshot shows a Gitee project page for 'mengxuegu'. At the top, there's a navigation bar with links like 'xiaomi', '前端', 'SpringBoot', '遵循 semver 标准', 'Android', 'Markdown生成左边', 'CODING | 代码托管', '计算机免费书籍', '电子', and '培训参考'. Below the navigation is the Gitee logo and a search bar. The main content area shows the repository name 'mengxuegu' and a note that it was forked from '总监 / mengxuegu'. It lists 8 commits, 1 branch, 0 tags, 0 releases, and 1 contributor ('alan_glb'). The commit history shows a recent update by 'alan_glb' resolving a conflict 16 minutes ago.

- 开发人员 克隆项目，然后修改内容提交

```
1 $ cd ../..
2 $ mkdir mengxuegu_kf
3 $ cd mengxuegu_kf/
4 $ git clone https://gitee.com/mxg999/mengxuegu.git
5 Cloning into 'mengxuegu'...
6 remote: Enumerating objects: 24, done.
7 remote: Counting objects: 100% (24/24), done.
8 remote: Compressing objects: 100% (8/8), done.
9 remote: Total 24 (delta 7), reused 24 (delta 7)
10 Unpacking objects: 100% (24/24), done.
11
12
13 $ cd mengxuegu/
14 $ vim demo1.txt
15 $ git add demo1.txt
16 $ git commit -m 'kf commit' demo1.txt
17 $ git push origin master
```

- 在码云上点击 Pull Request 标签，+新建 Pull Request

The screenshot shows the same Gitee project page for 'mengxuegu'. The 'Pull Requests' tab is highlighted with a red arrow pointing to the '+新建 Pull Request' button. The page displays a table of pull requests, with the first one being '开启的' (Open) with a status of '0'. There are also filters for '搜索 pull requests' (Search pull requests) and '重置过滤条件' (Reset filter conditions).



- 切换 **总监**，点击 **Pull Request** 标签，点击下方开启的

总监 / mxg999

已开启 | Issues 0 | Pull Requests 1 | 附件 0 | Wiki 0 | 统计 | 服务 | 管理

+新建 Pull Request

所有 | **已开启** | 已合并 0 | 已关闭 0 | 创建者 | 测试人员 | 审阅人员 | 里程碑 | 排序

2分钟前

1 by mxg999 | 开发人员:master → 总监:master

- 预览审核代码

已开启 | 1 开发完成

开发人员:master → 总监:master

开发人员 创建于：6分钟前

代码已经提交, 请验收

共 0 条评论, 1 人参与

源分支与目标分支没有冲突
可自动合并 (如果你仍然想手动合并, 点击这里 查看怎样操作)

合并

评论 0 提交 1 预览

1 文件发生了变化, 影响行数: +1 -0

全屏查看 双栏对比

demo1.txt

... 2 hi, 我是经理, 我加入项目组来啦....开始敲代码helloworld
3 hi, 我是经理, 我加入项目组来啦....zzzzzjjjjjj
4 haha, 我是经理, 我又来了....
5 我是外派来的小三

代码已经提交，请验收

共 0 条评论, 1 人参与

Merge pull request !1 from 开发人员/master

!1 开发完成

取消 接受 Pull Request

评论 0 提交 1 文件 1

1 文件发生了变化，影响行数: +1 -0

demo1.txt

... | ... @@ -2,3 +2,4 @@ hello, 我是总监
2 | 2 hi, 我是经理, 我加入项目组来啦...。开始敲代码helloworld

查看文件 @ 2ff37bb

全屏查看 双栏对比

8.12 SSH登录

windows系统自带保存密码功能，如果 mac 没有自带保存密码功能怎么办？

https 没有记住就帐号，每次登录时都要输入帐号密码，就很浪费时间。

ssh 可以避免每次添加用户名密码。

- 生成密钥

```
1 进入当前用户的家目录 ~  
2 $ cd ~  
3  
4 如果有.ssh 目录，则删除.ssh 目录  
5 $ rm -rf .ssh  
6  
7 运行命令生成.ssh 密钥目录  
8 $ ssh-keygen -t rsa -C mengxuegu666@163.com  
9 [注意：这里-c 这个参数是大写的 C ， 出现光标停留的地方直接按 Enter 键]  
10  
11 进入.ssh 目录查看文件列表  
12 $ cd .ssh  
13  
14 查看所有目录 与 文件  
15 $ ls -lf  
16 ./ ../.id_rsa id_rsa.pub  
17  
18 查看 id_rsa.pub 文件内容，复制 id_rsa.pub 文件内容(公钥)  
19 $ cat id_rsa.pub
```

- 登录 码云，点击用户头像→设置



- 点击 SSH公钥,输入复制的密钥信息

SSH公钥

使用SSH公钥可以让你在你的电脑和码云通讯的时候使用安全连接 (Git的Remote要使用SSH地址)

您当前的SSH公钥数: 0

你还没有添加任何SSH公钥

添加公钥

标题
mykey

公钥
把你的公钥粘贴到这里, 查看 [怎样生成公钥](#)

```
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQAB...mc1h4w0KJOGDj3/71JEA  
nZZ6Bt0zJmTNR1WZ7KOzGiZVQVgTVTlfw1f3z/08rvPMWaLx++PHJVjwpAgXs30wnw871XmgqlnHrc32jnDFya2Q  
9oafutHDblF4b7NS4iQLKs+kwSWzf8gxQBE5xk0/r0m0Hcvj6//NW898i4azp14C0VqyirNdy2ytjdoZtEhKojFvsLTyga  
R+b6ODcko4a38OGHRWAR/YIMn6JvHREnUT5...DTPjuXZGHq03TZh0g  
nbq1DW+LzkeYK6EWYh3 mengxuegu666@163.com
```

基本设置

- 个人资料
- 修改密码
- 个性域名
- 通知设置

安全设置

- SSH公钥

数据管理

- 升级为组织
- 升级为企业版
- 私有项目成员
- 代码风格

- 权限验证

权限验证

当前账户密码

确定

- 回到 `Git Bash` 创建远程ssh地址别名

```
1 cd /d/gitStudy/mengxuegu
2 vim demo1.txt
3 git commit -m "test ssh update" demo1.txt
4 git remote add origin_ssh git@gitee.com:mxg6666/mengxuegu.git
5
6 git remote -v
7 origin https://gitee.com/mxg6666/mengxuegu.git (fetch)
8 origin https://gitee.com/mxg6666/mengxuegu.git (push)
9 origin_ssh git@gitee.com:mxg6666/mengxuegu.git (fetch)
10 origin_ssh git@gitee.com:mxg6666/mengxuegu.git (push)
```



- 推送文件进行测试

```
1 git push origin_ssh master
```

```
$ git push origin_ssh master
The authenticity of host 'gitee.com (218.11.0.86)' can't be established.
ECDSA key fingerprint is SHA256:FQGC9Kn/eye1W8icdBgrQp+KkGYoFgbVr17bmjey0Wc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'gitee.com,218.11.0.86' (ECDSA) to the list of known
hosts.
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 278 bytes | 278.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Powered by Gitee.com
To gitee.com:mxg6666/mengxuegu.git
 883a126..6f337c5  master -> master
```

9. TortoiseGit 图形化工具

9.1 什么是TortoiseGit

- TortoiseGit 是基于图形化操作git的工具，来代替前面的命令行



9.2 下载TortoiseGit

- 官网：<https://tortoisegit.org/download/>
- 根据自己电脑操作系统选择对应版本（32位或64位）：

TortoiseGit.org » Download

Download

The current stable version is: 2.7.0

For detailed info on what's new, read the [release notes](#).

[FAQ: System prerequisites and installation](#) - This version doesn't run on Windows Vista and below, use [2.4.0](#) instead.

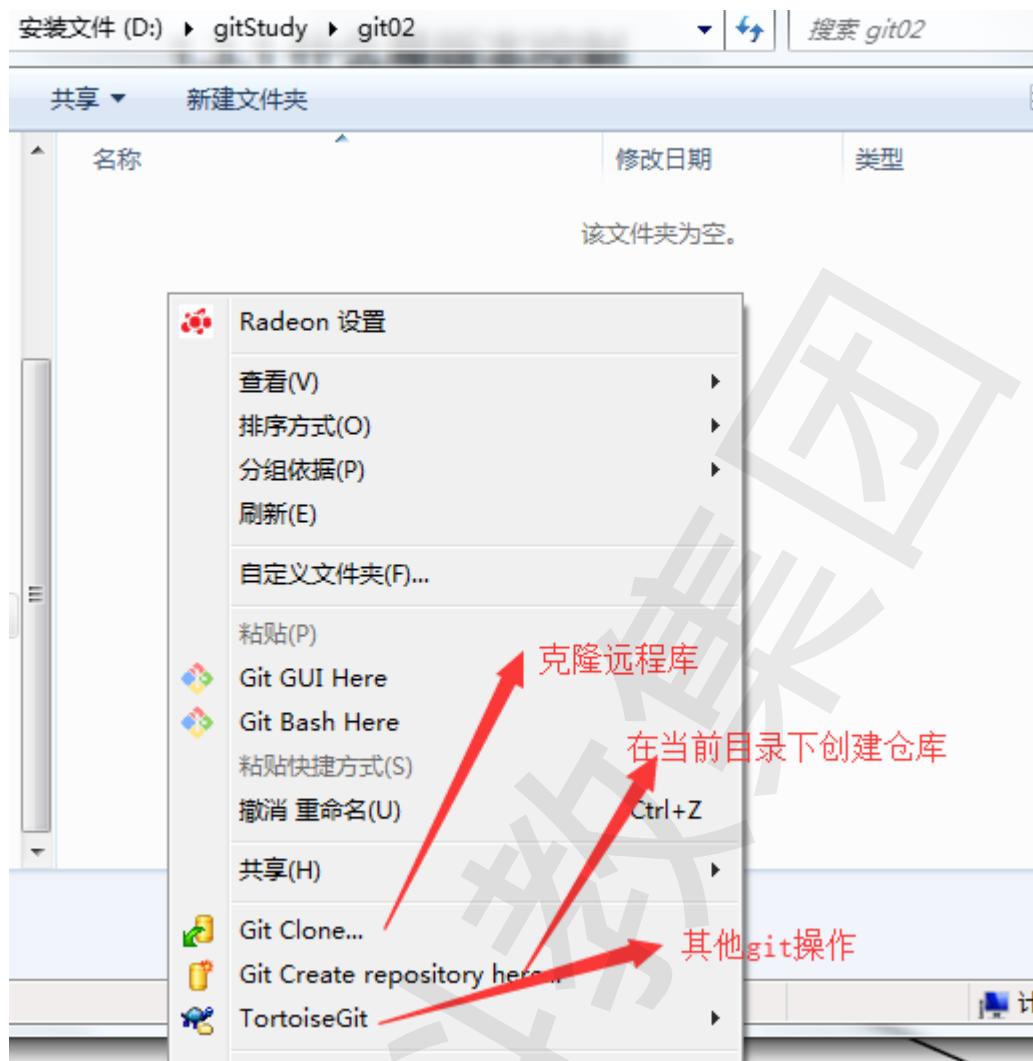
[Donate](#)

Please make sure that you choose the right installer for your PC, otherwise the setup will fail.

for 32-bit Windows	for 64-bit Windows
Download TortoiseGit 2.7.0 - 32-bit (~16.0 MiB)	Download TortoiseGit 2.7.0 - 64-bit (~18.7 MiB)

9.3 安装TortoiseGit

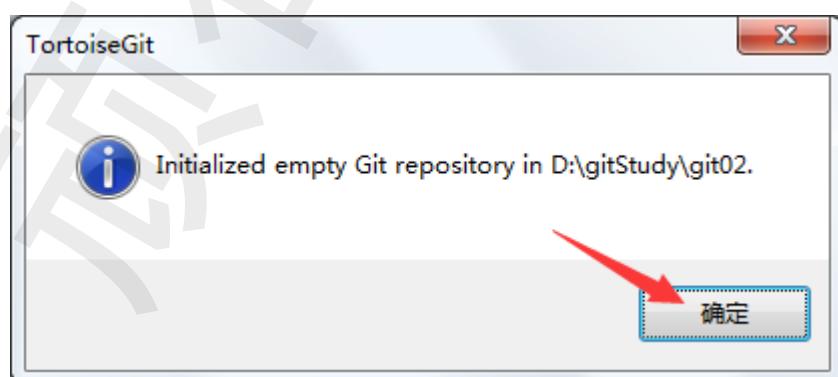
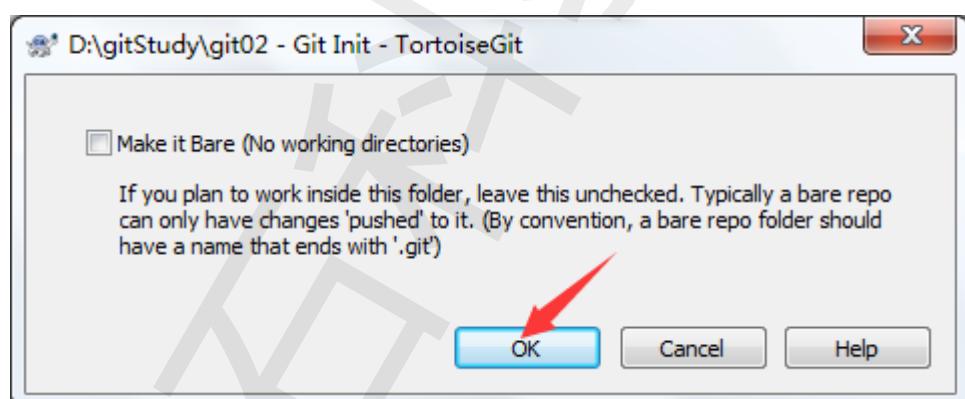
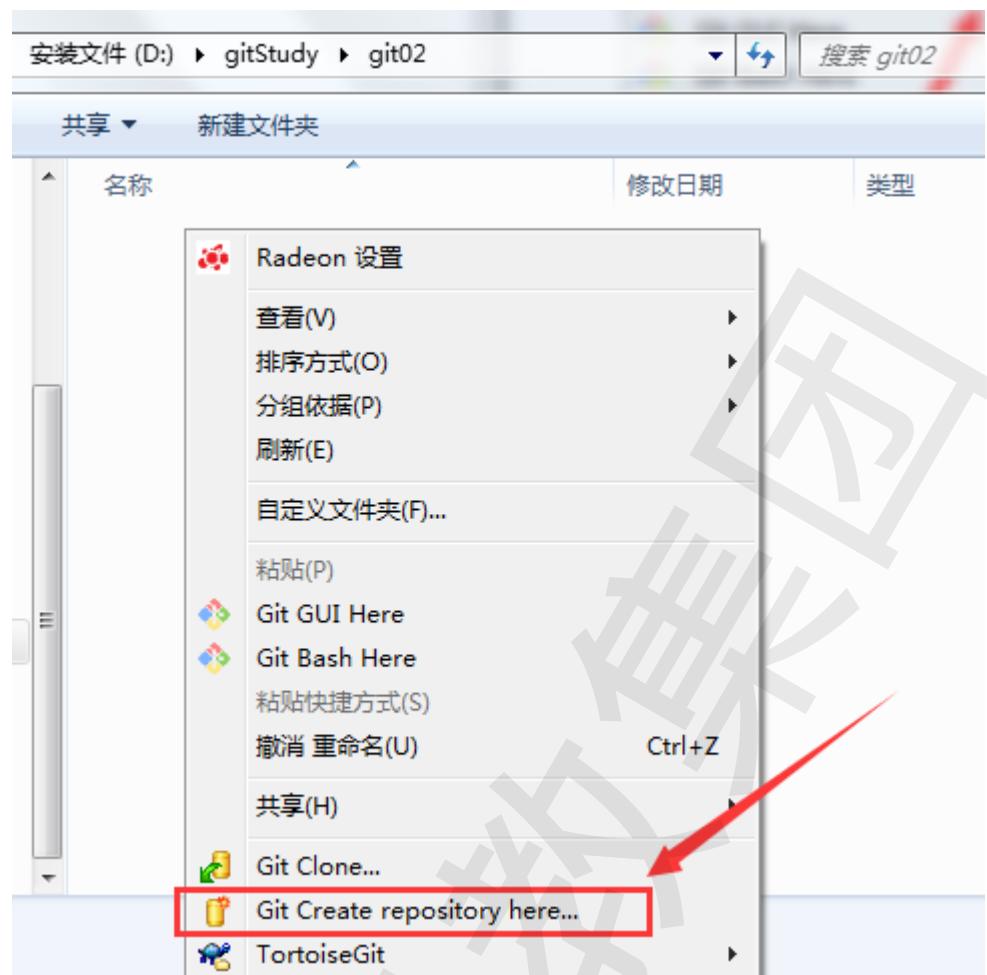
- 傻瓜式安装，此处省略安装步骤
- 安装完成后，任意空白处，单击鼠标右键，如果出现下图，则安装成功：

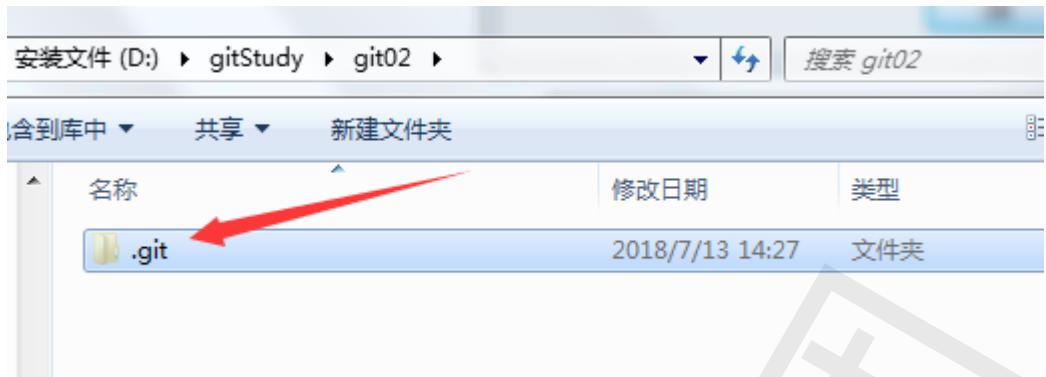


9.4 使用TortoiseGit

9.4.1 初始化本地库

- 进入git02目录，将git02作为本地仓库，如下：



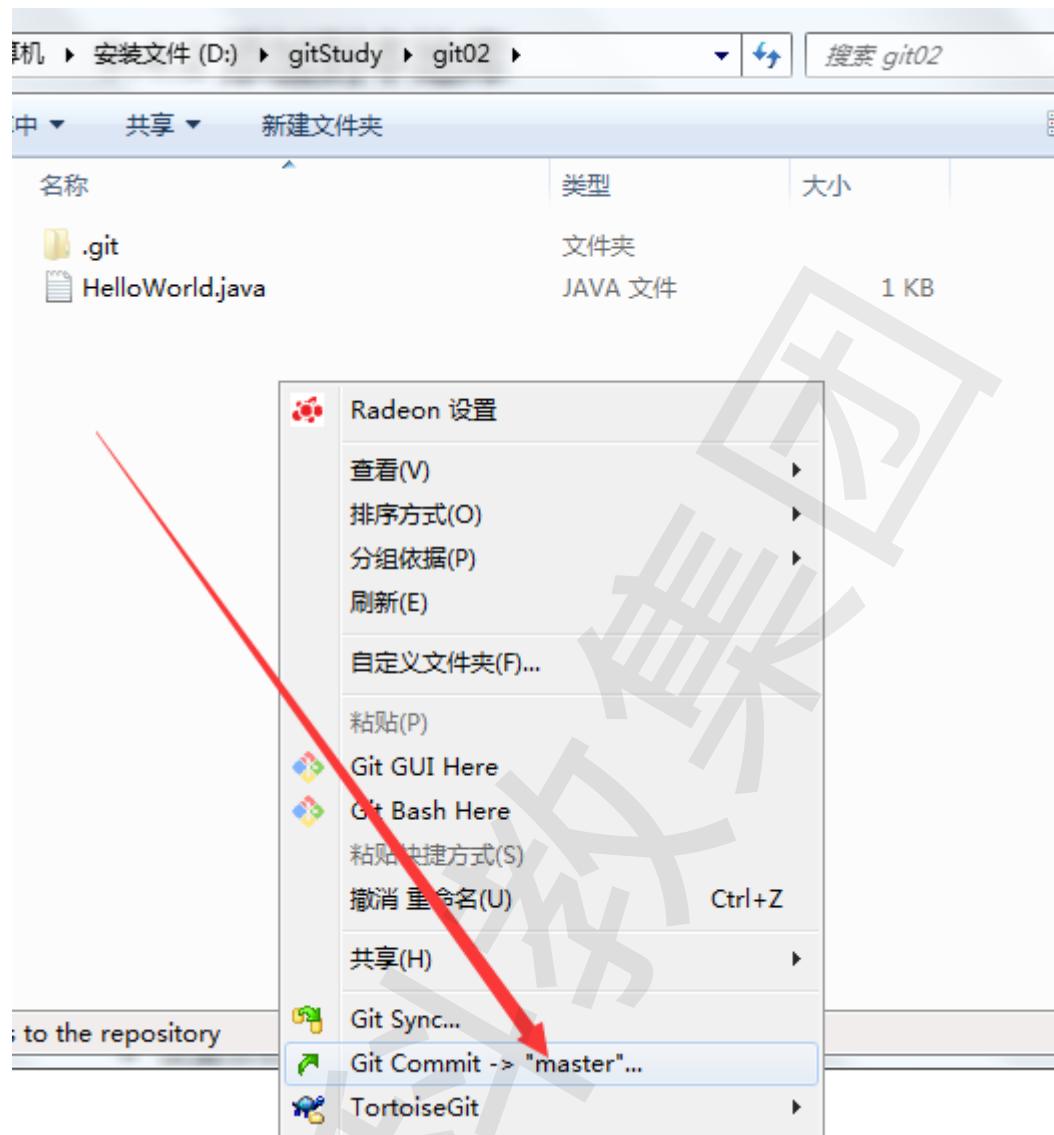


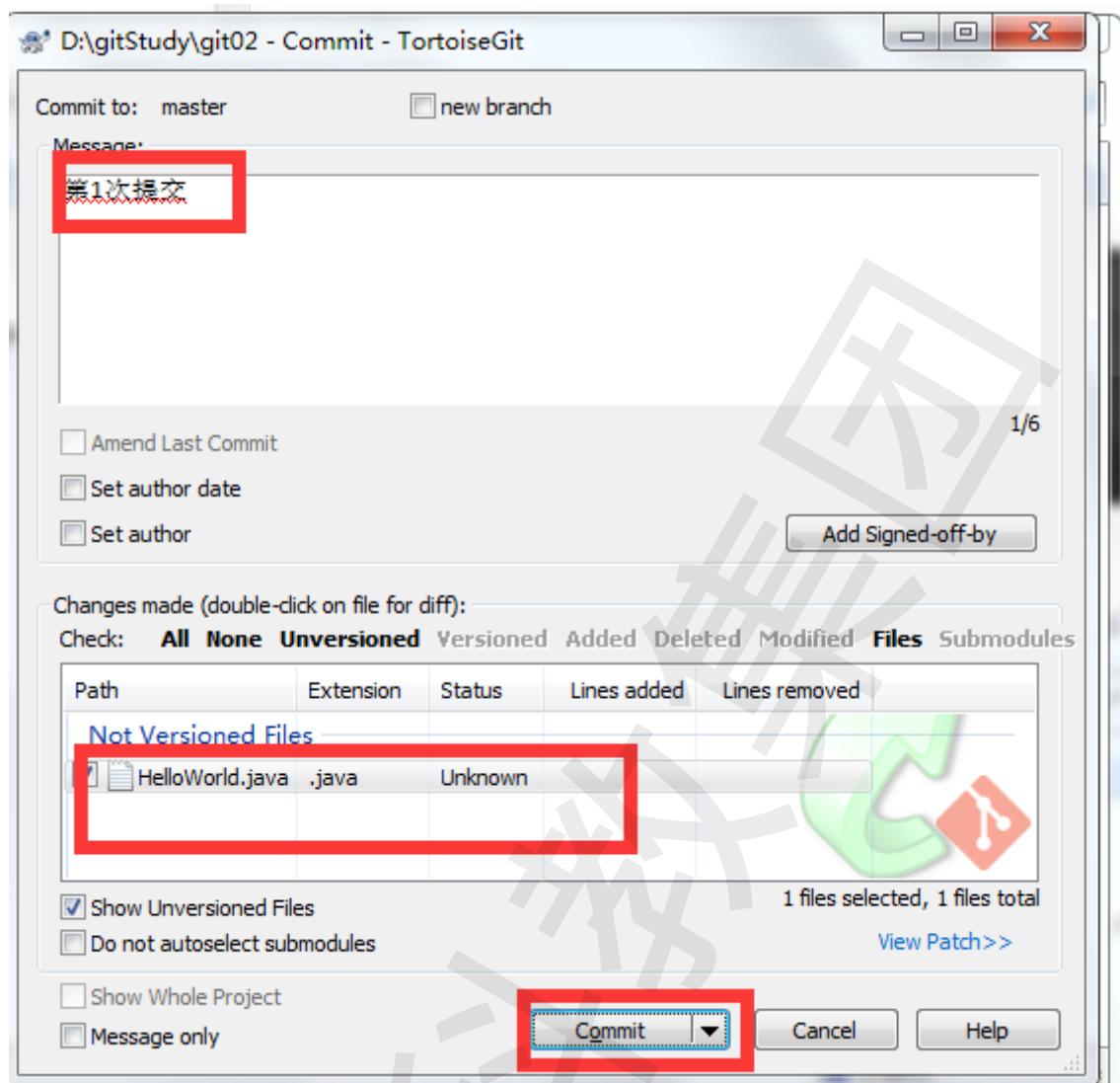
9.4.2 添加到本地库

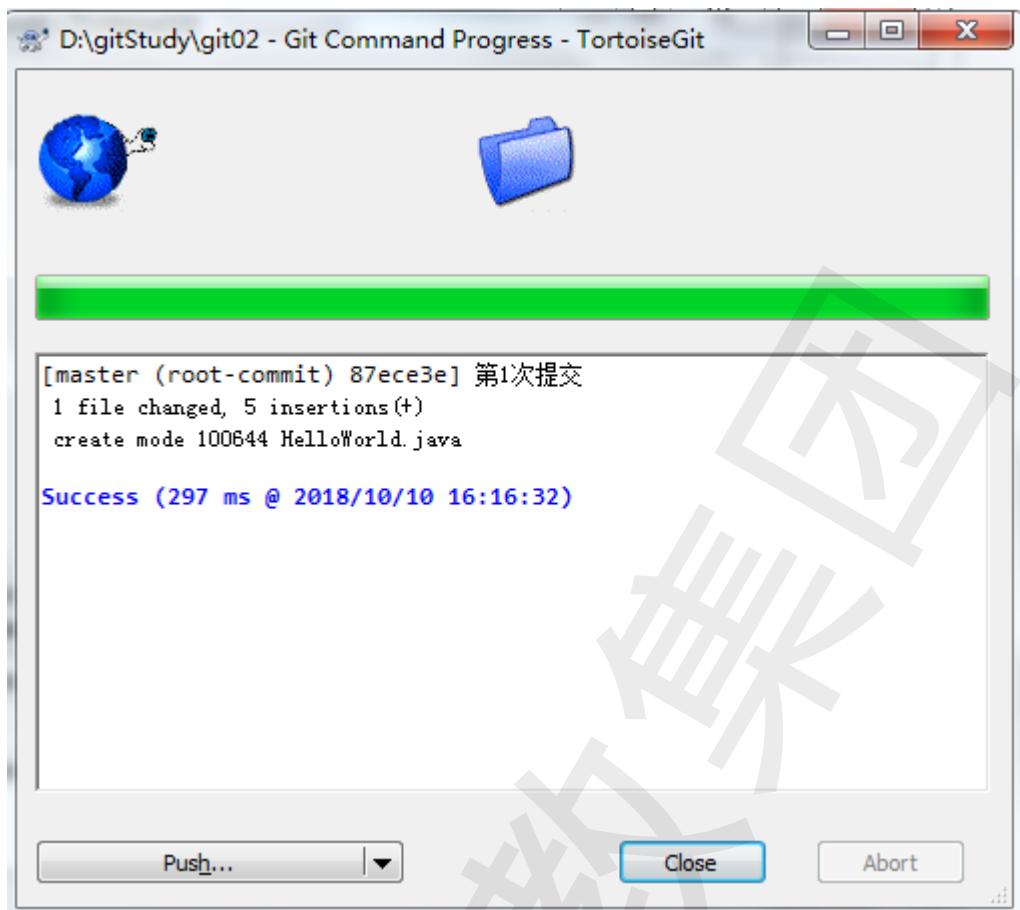
- 新增HelloWorld.java文件并添加内容

```
1 public class HelloWorld {  
2     public void test2() {  
3  
4     }  
5 }
```

- 开始推送到本地库

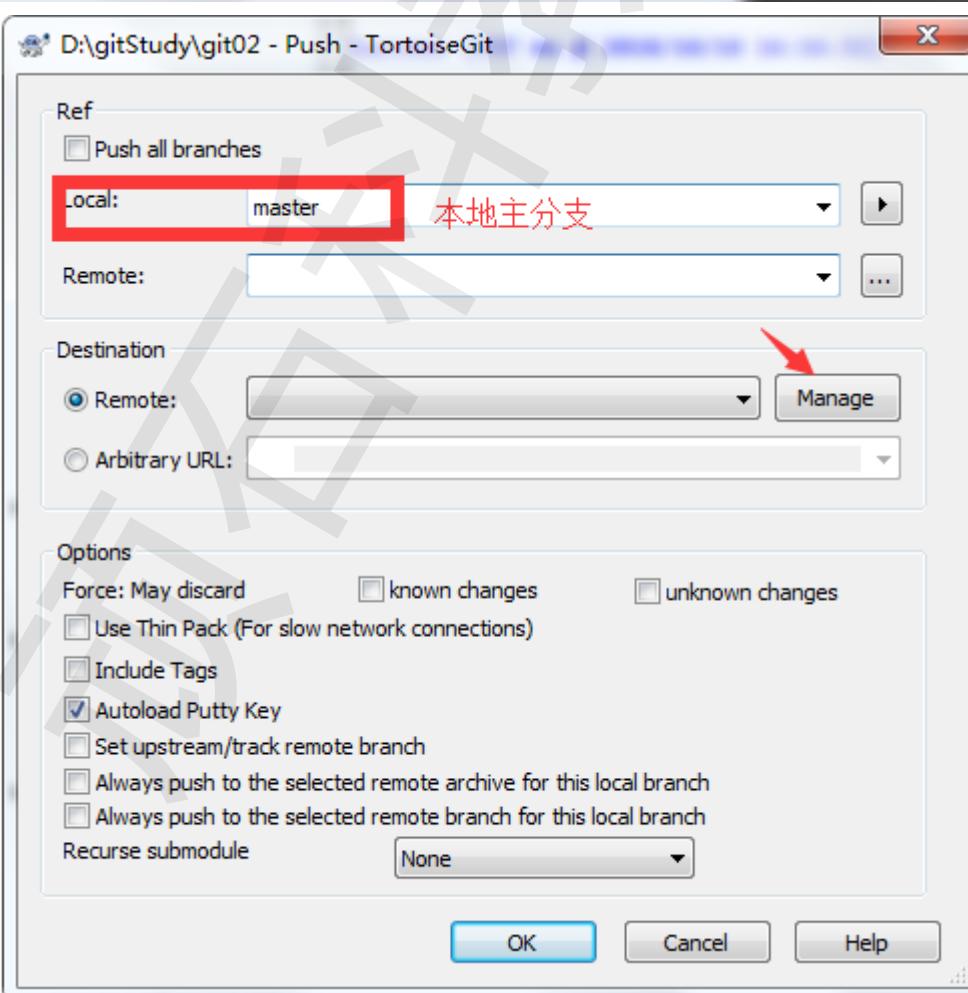
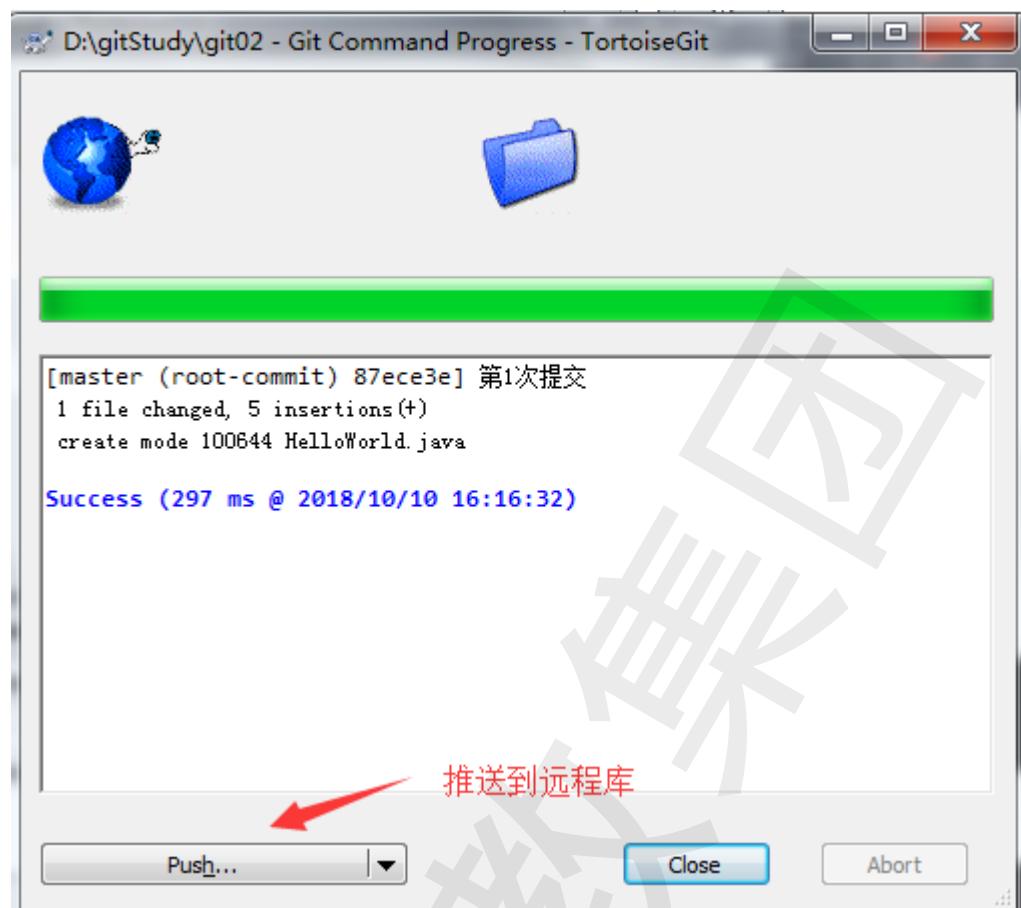




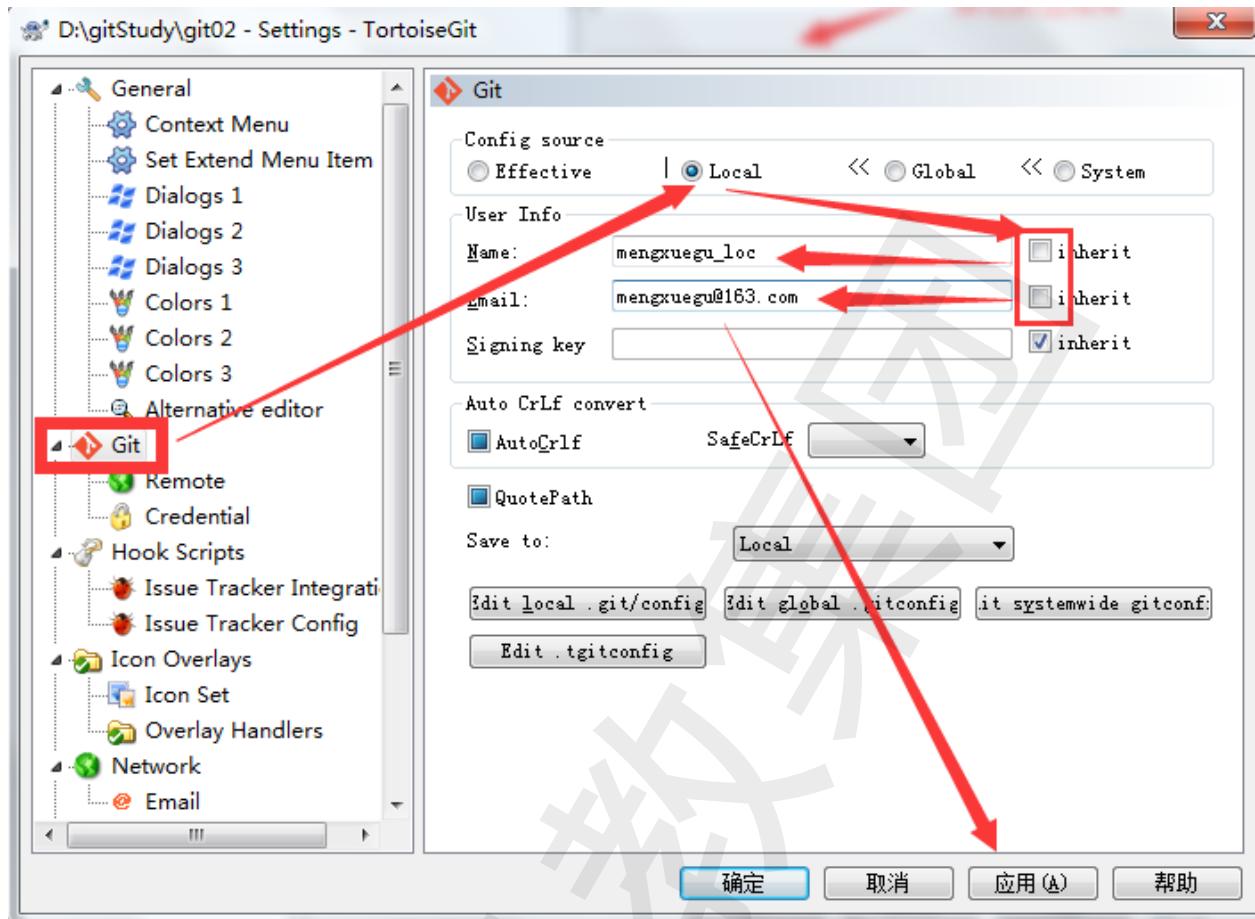


9.4.3 推送到远程库(远程库需要先创建)

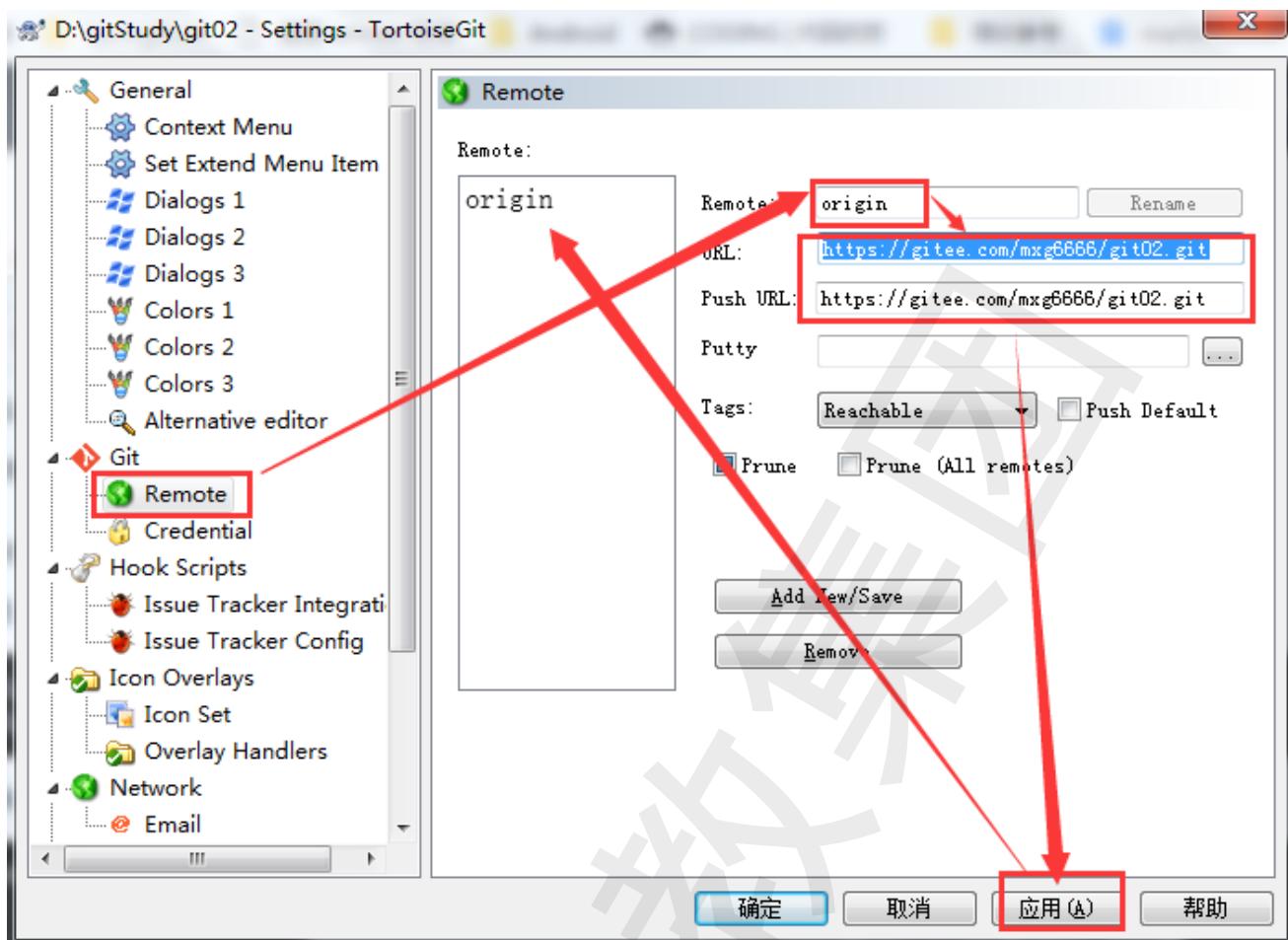
- 上面commit后，会弹出下列窗口，点左下角有个 push 可推送

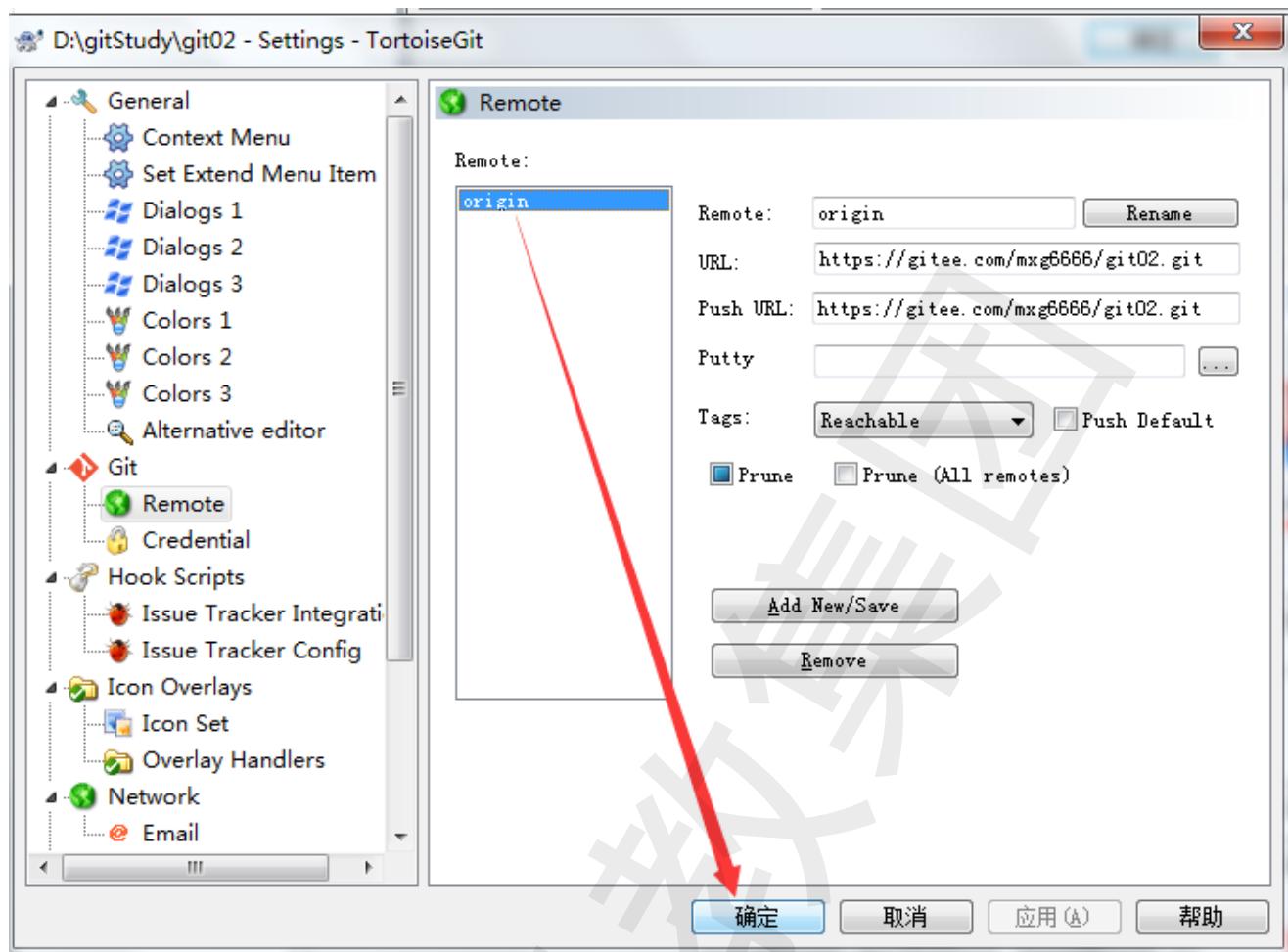


- 设置签名

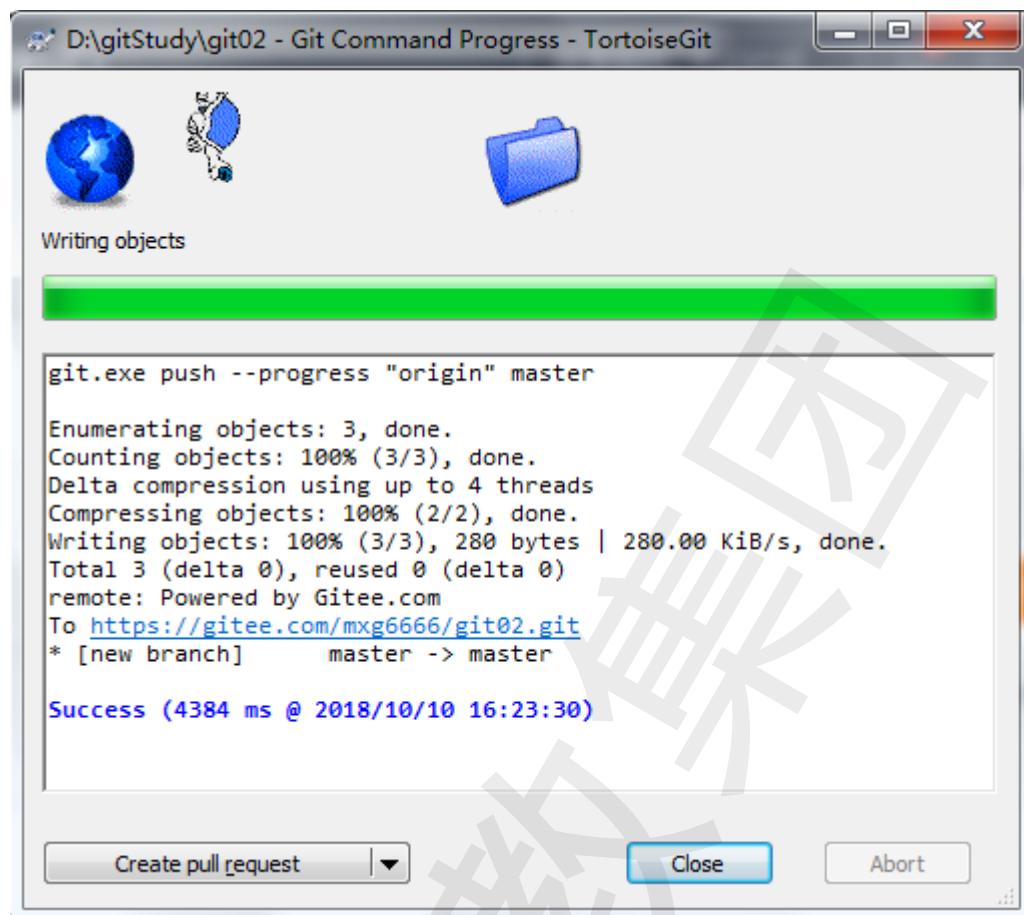


- 设置远程仓库地址（要先在远程服务器上仓库一个仓库）





- 推送成功

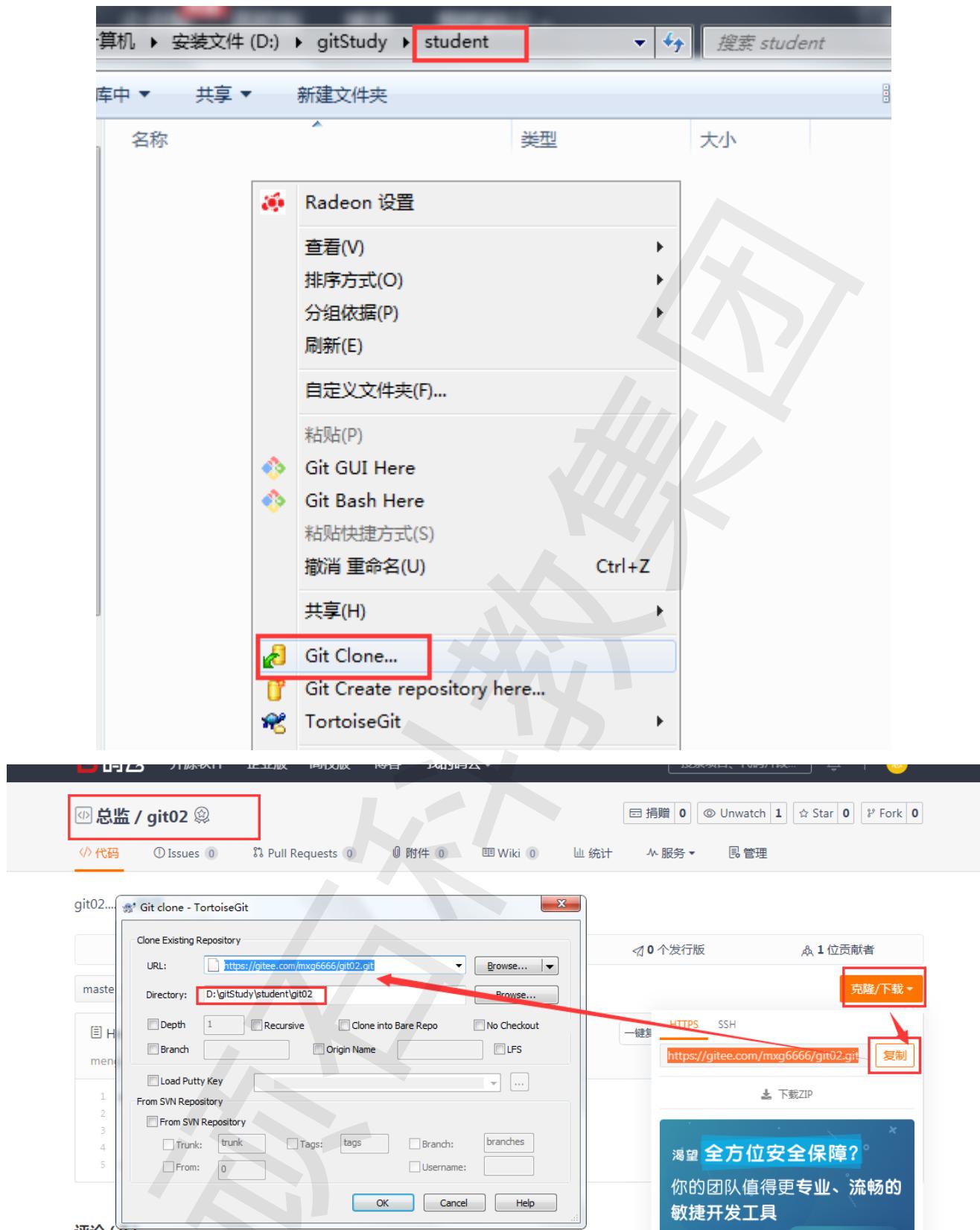


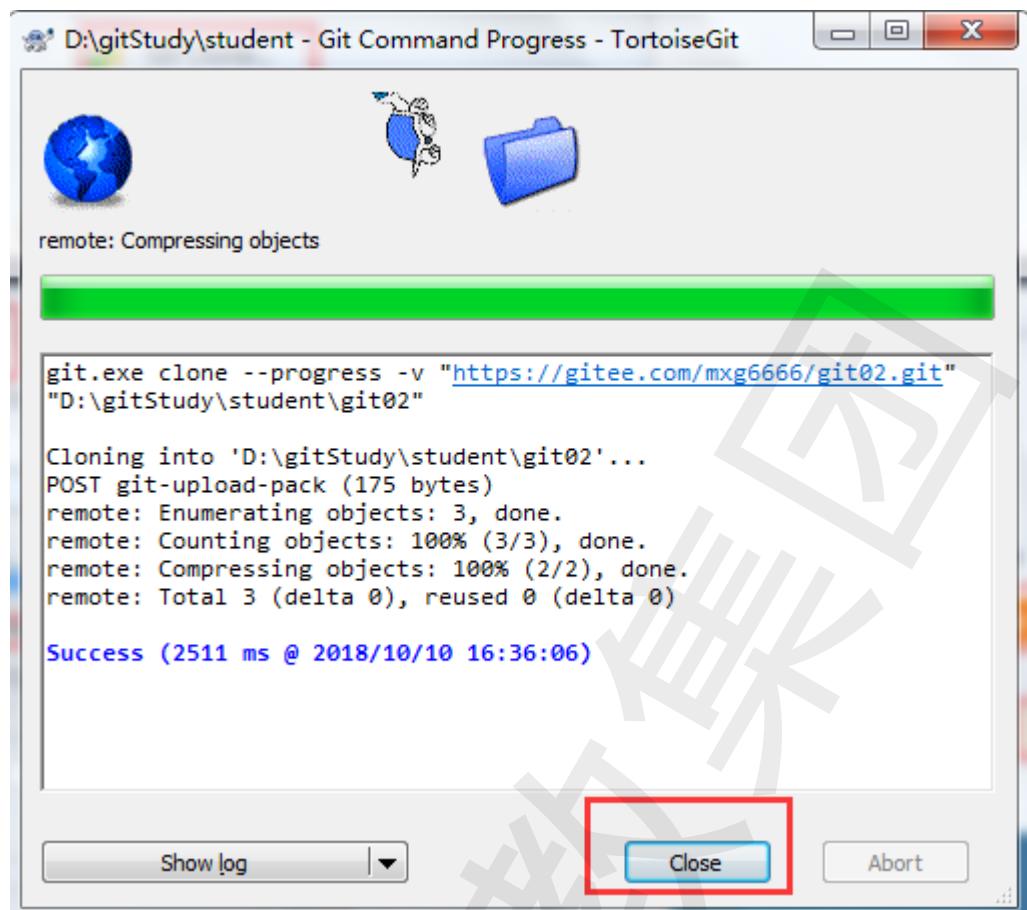
- 远程库效果

The screenshot shows a GitHub repository page for "总监 / git02". The repository has 1 star, 1 fork, and 1 contributor. The "Code" tab is selected, showing 1 commit, 1 branch, 0 tags, 0 releases, and 1 contributor. The commit history shows a single commit from "mengxuegu_glo" titled "提交于 9分钟前, 第1次提交". The commit message contains the following Java code:

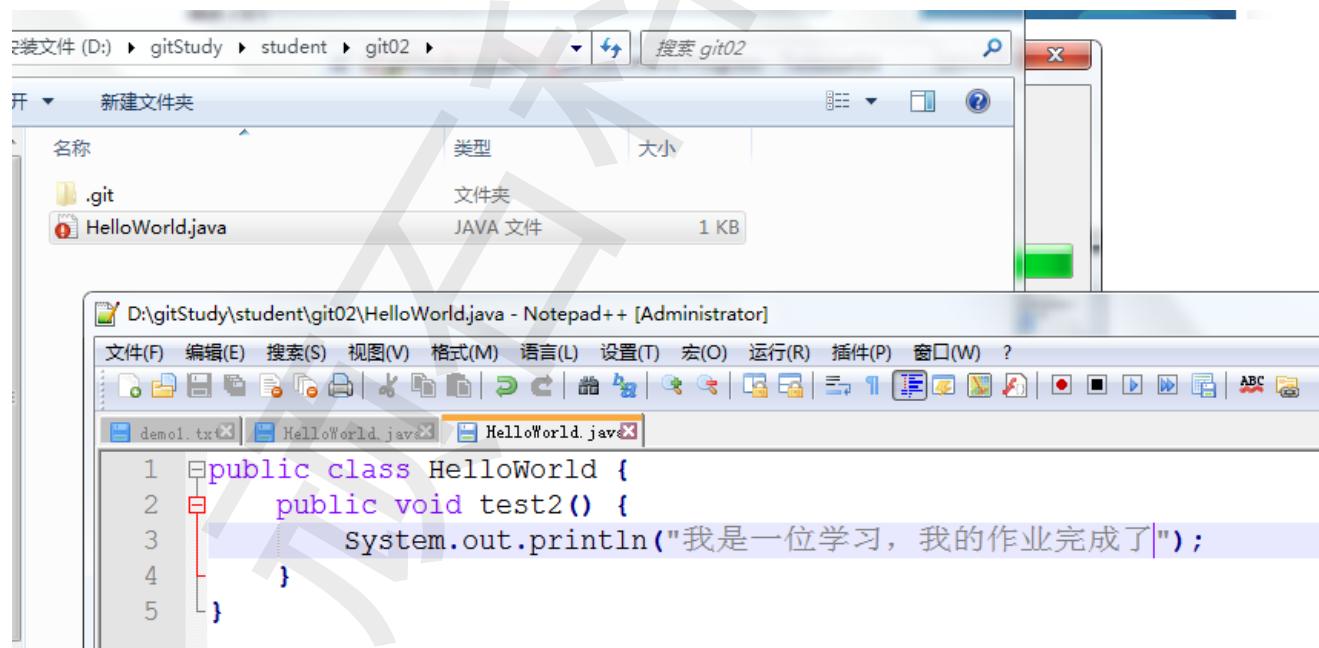
```
1 public class HelloWorld {
2     public void test2() {
3
4     }
5 }
```

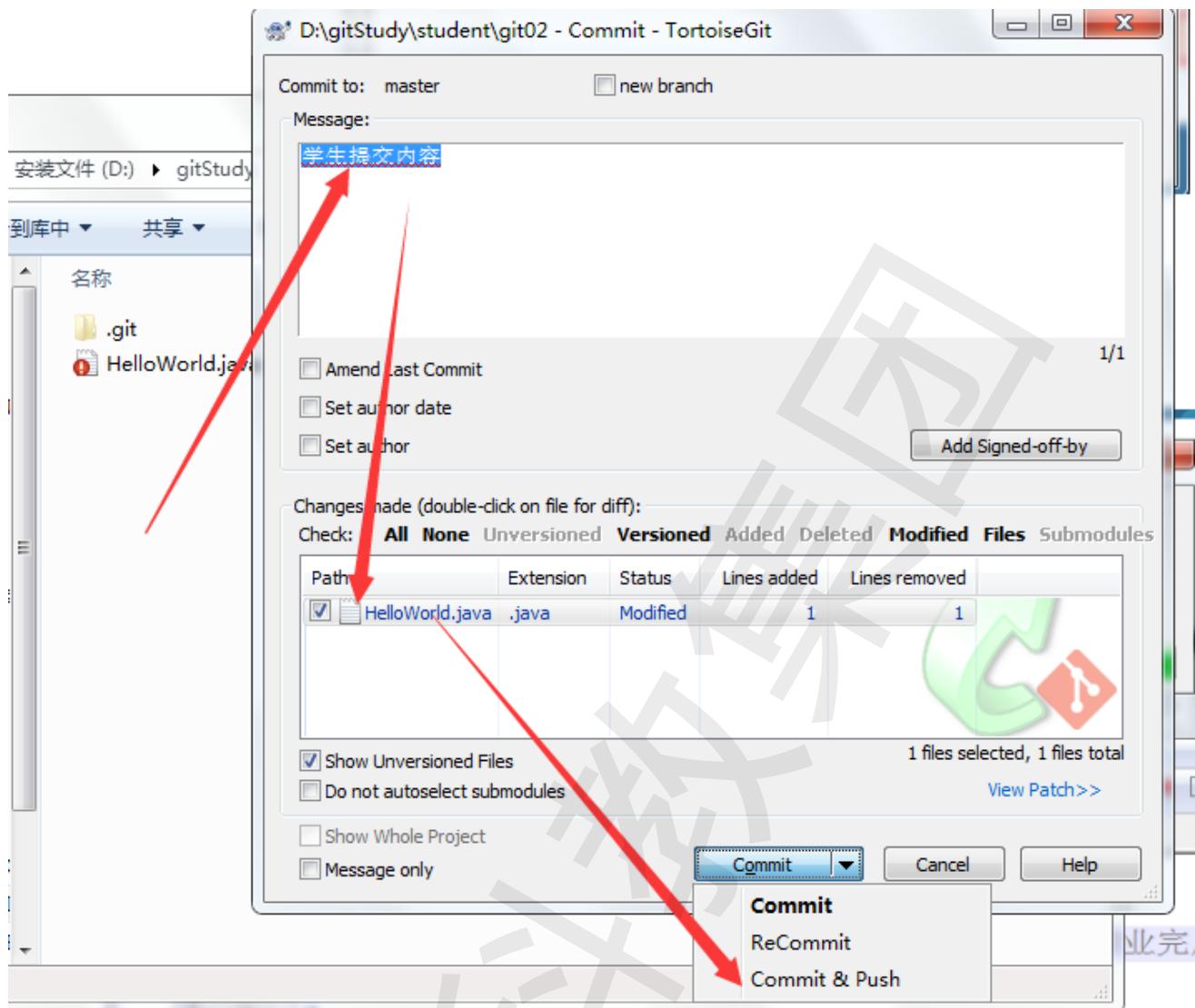
9.4.4 克隆远程库到本地库(student)

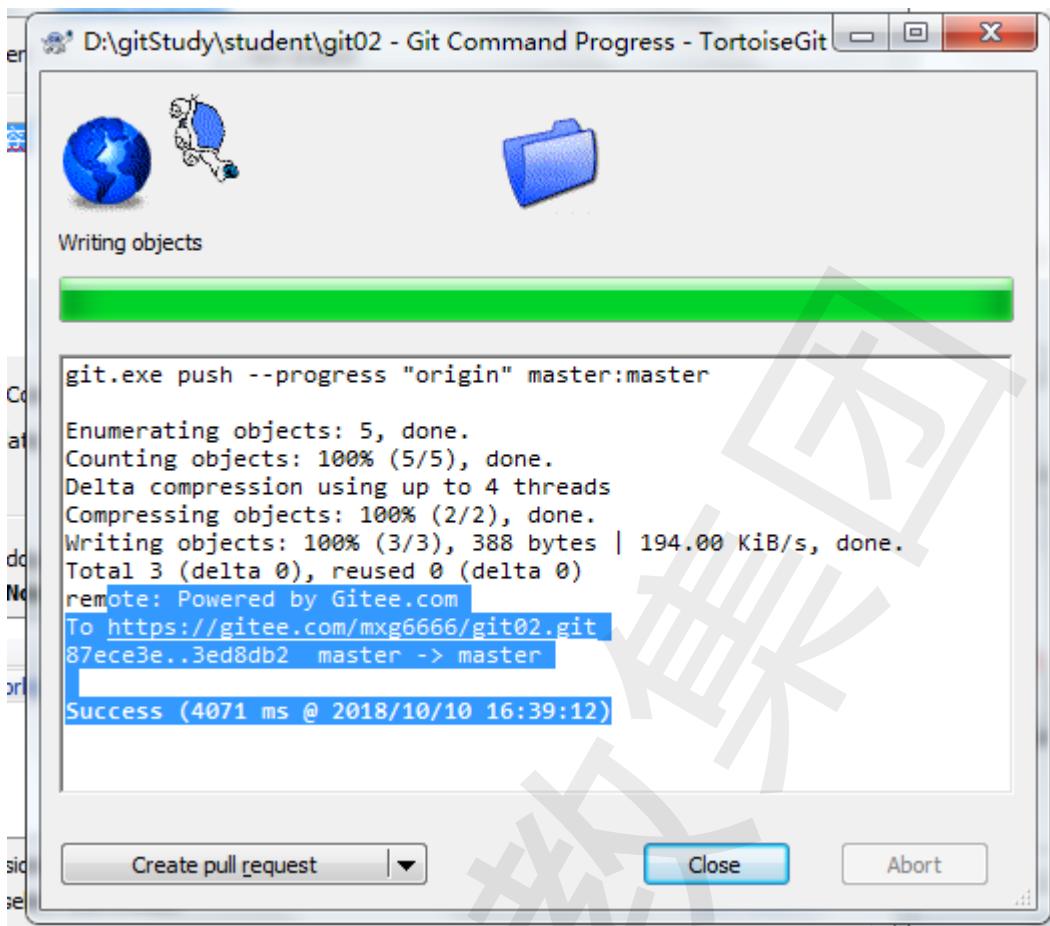




9.4.5 修改代码，推送到远端库(student)





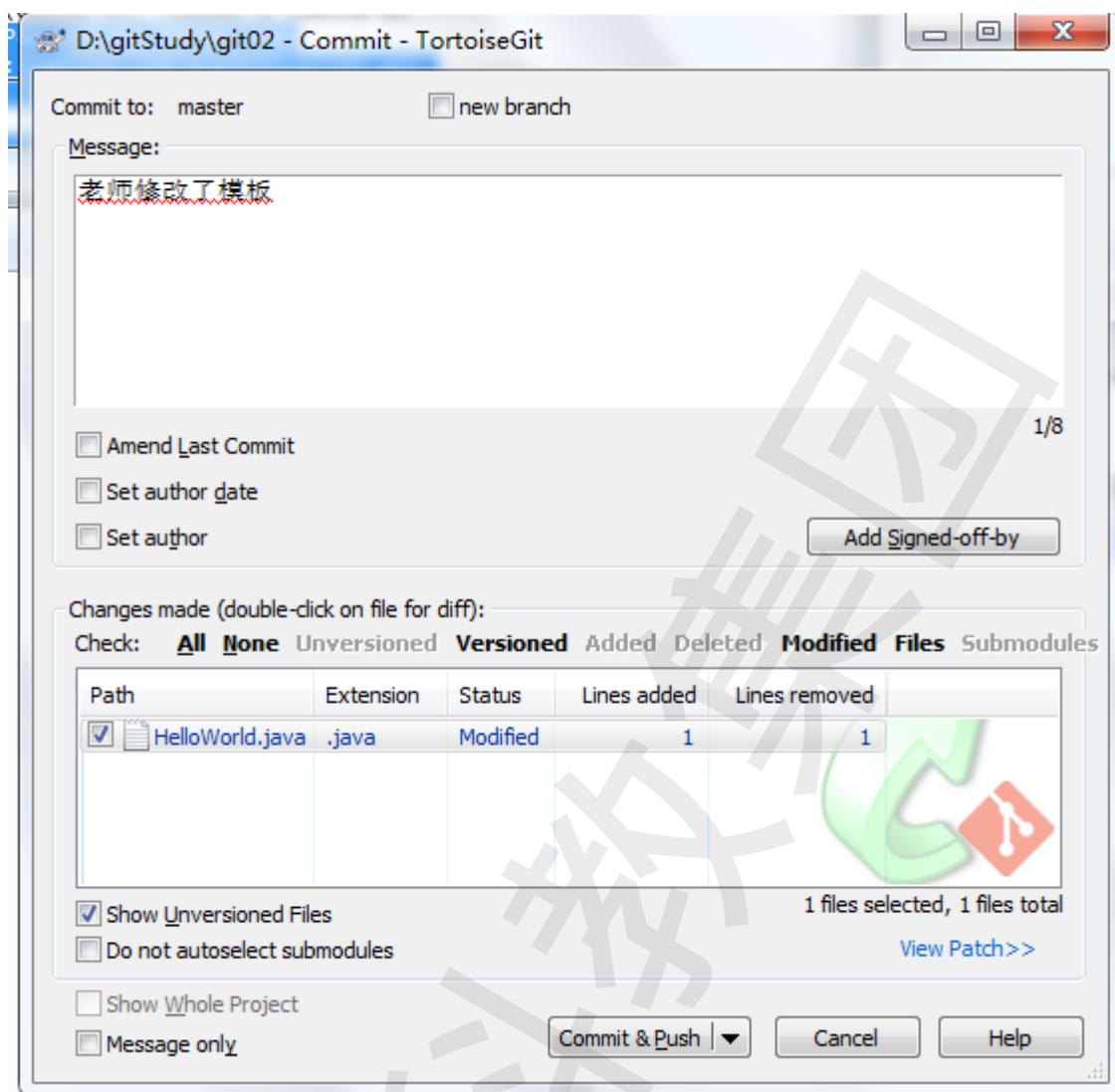


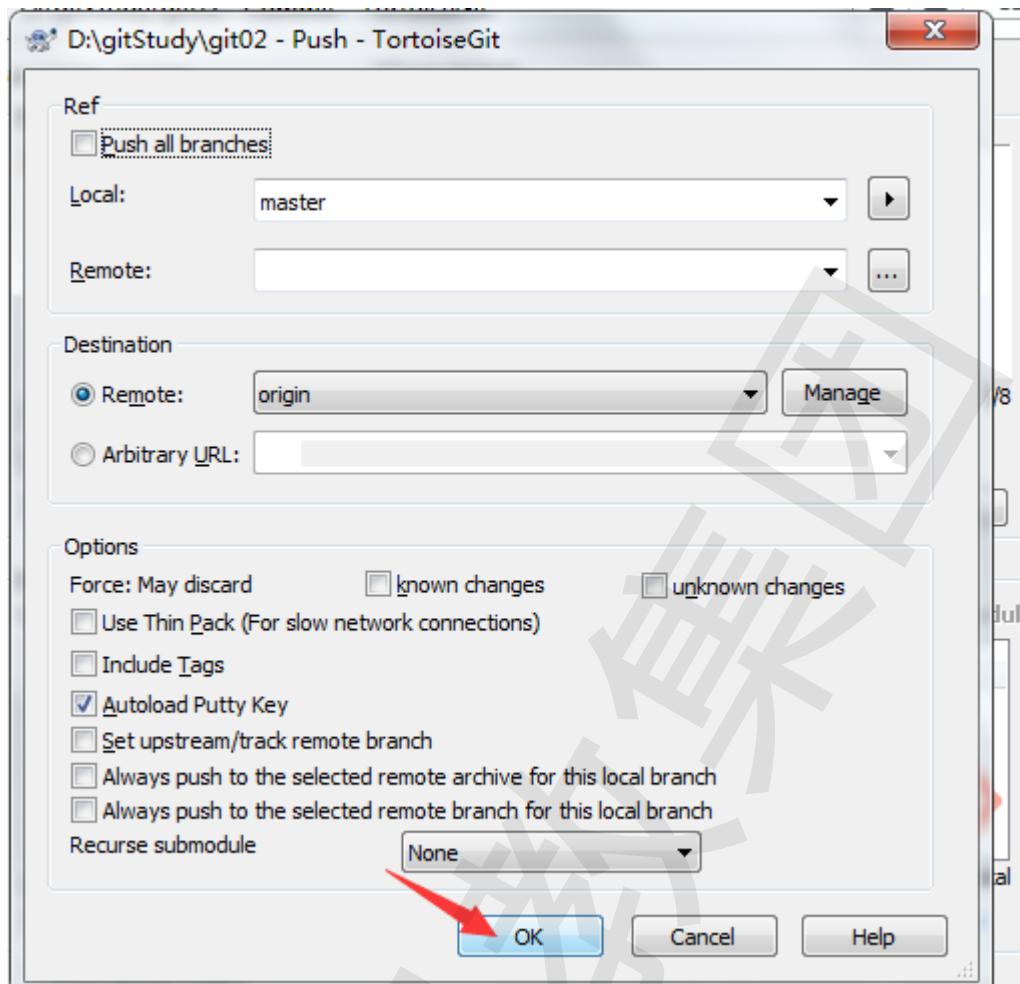
9.4.6 冲突解决

- 老师修改HelloWorld.java

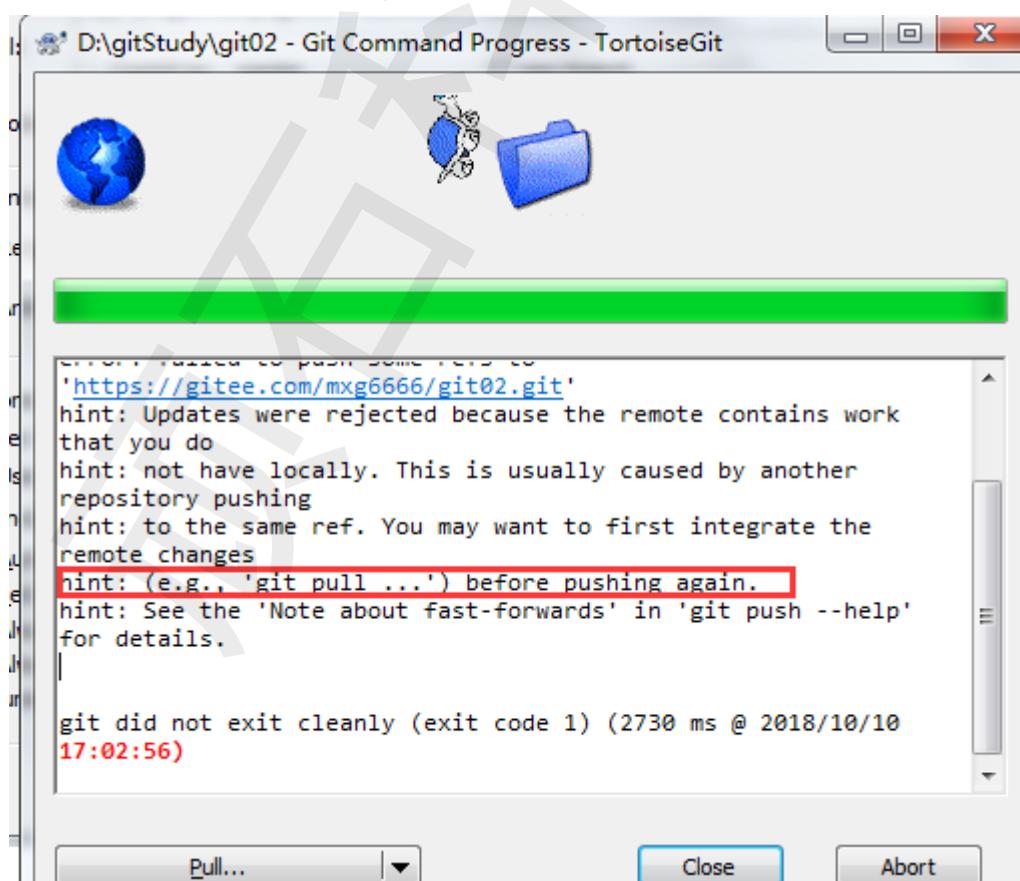
```
1 public class HelloWorld {
2     public void test2() {
3         System.out.println("我是老师，同学们直接在下面实现该方法");
4     }
5 }
```

- 当前远程库中已经被student提交过一次了，3行已经有数据，把当前代码提交上会产生冲突



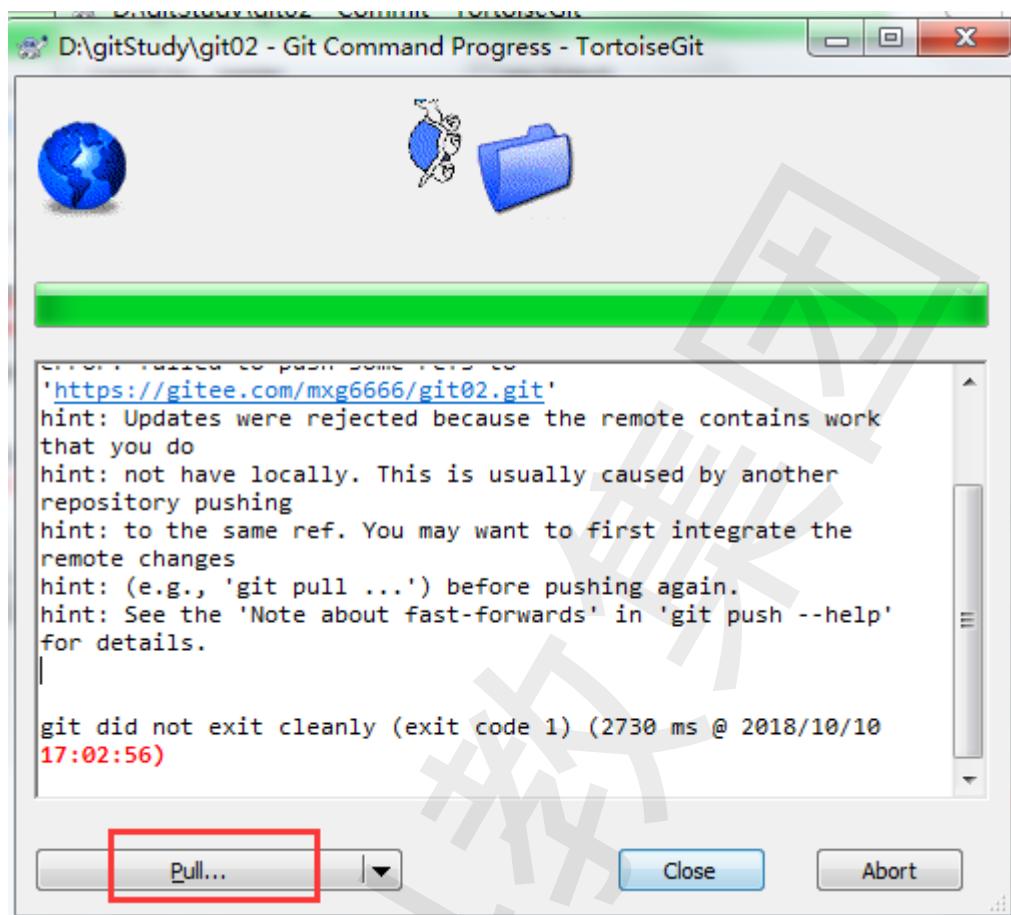


- 出错：

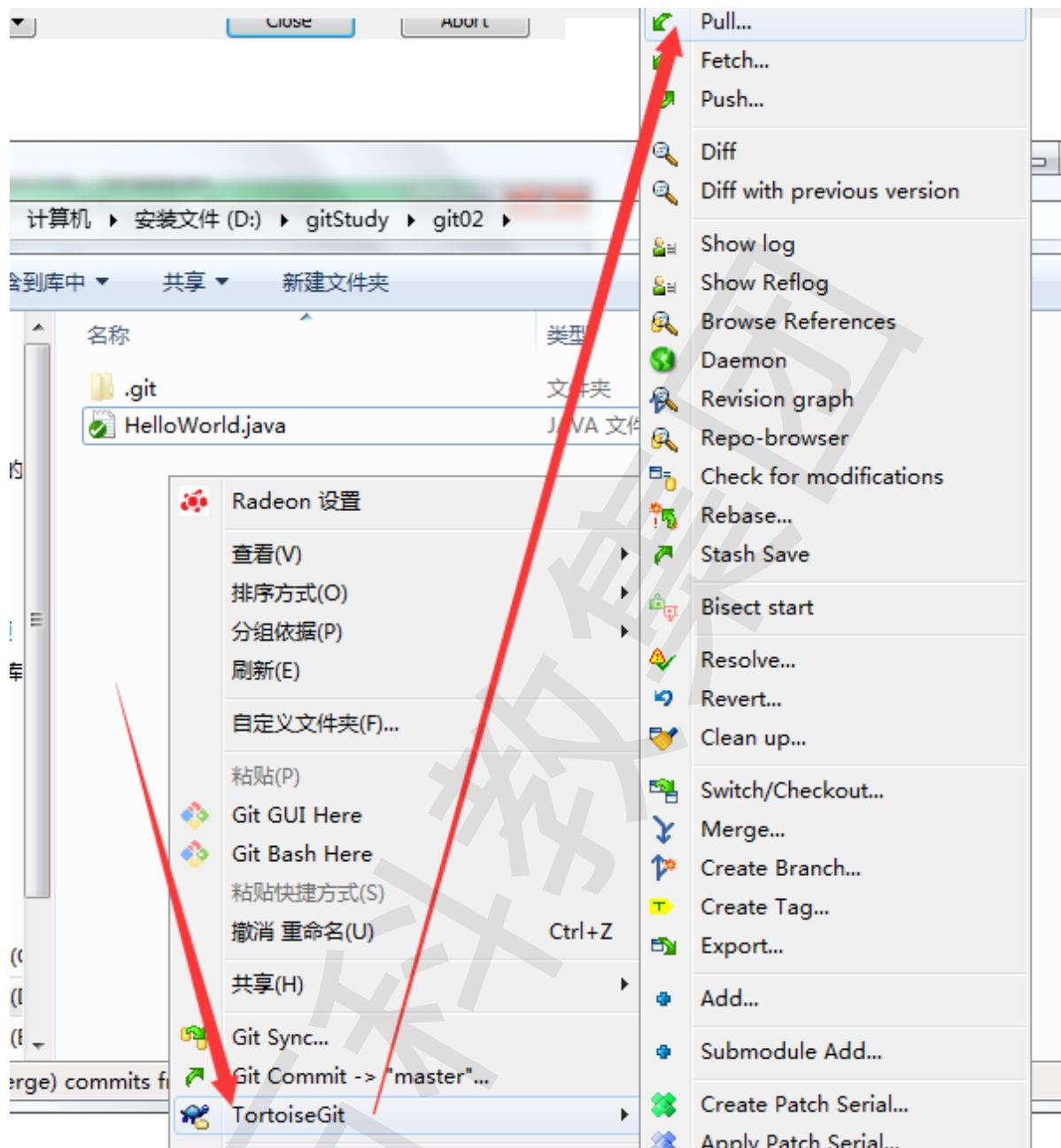


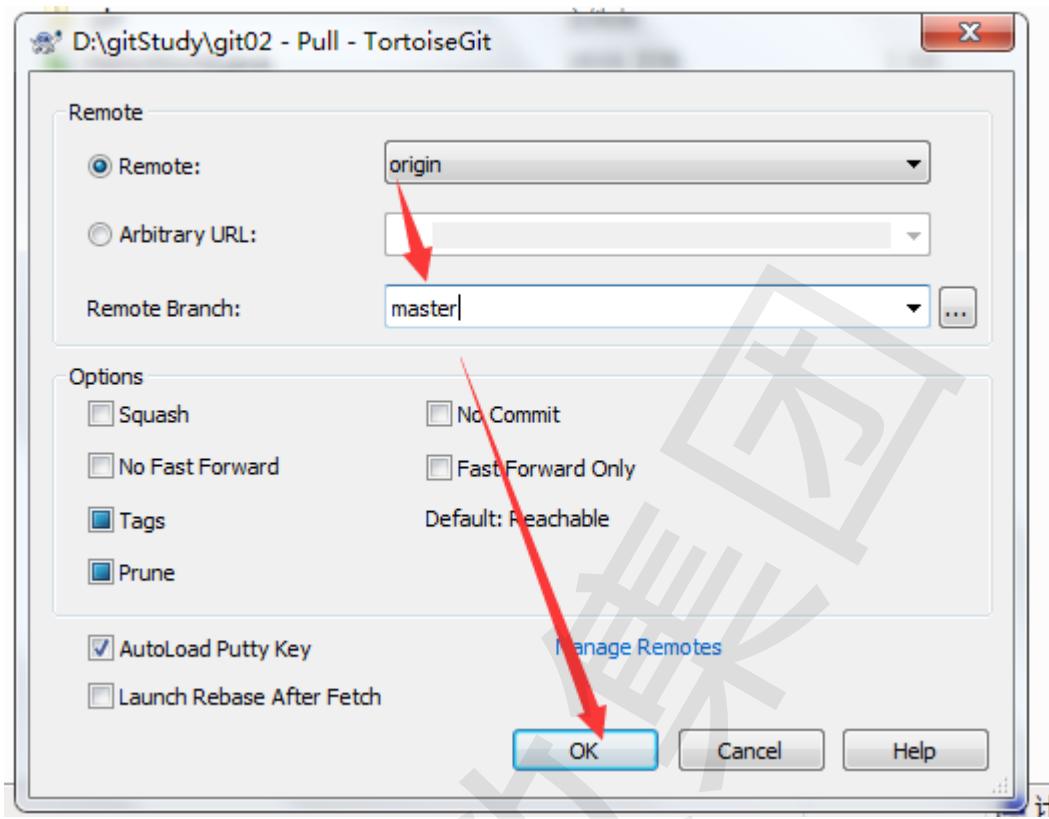
- 要先pull拉取远程

- 方式1：

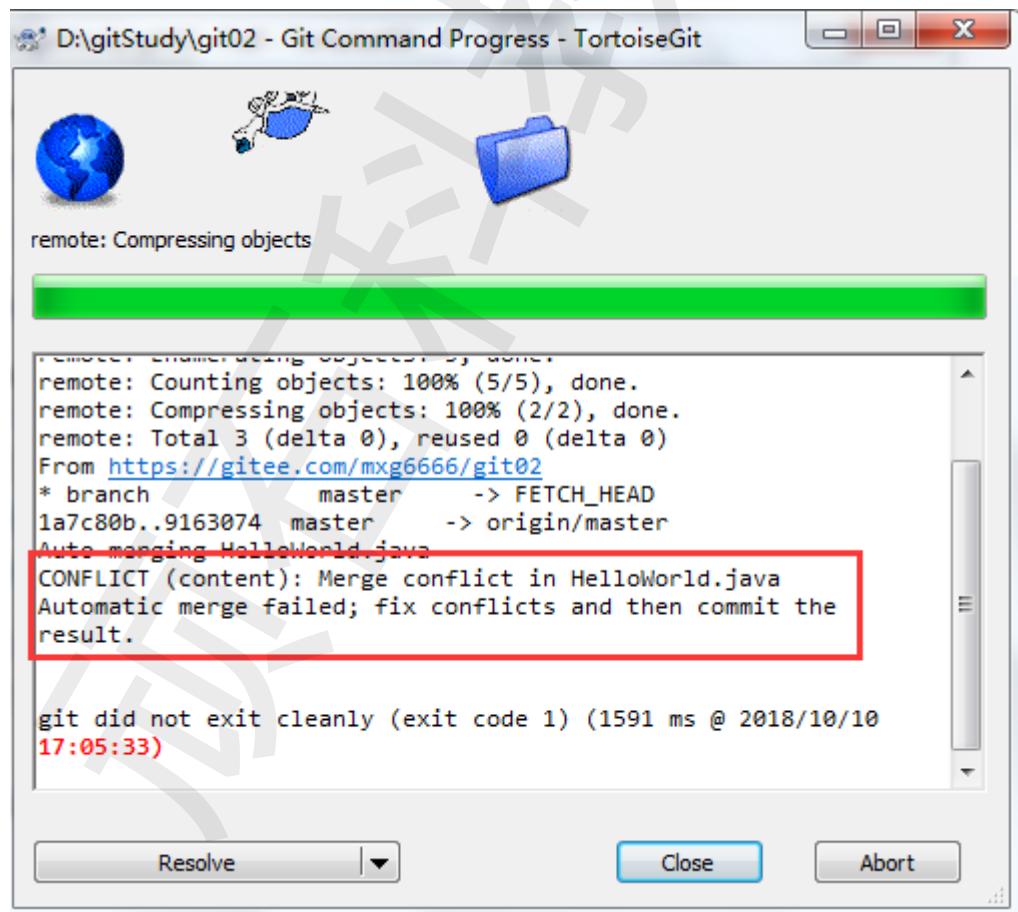


- 方式2：

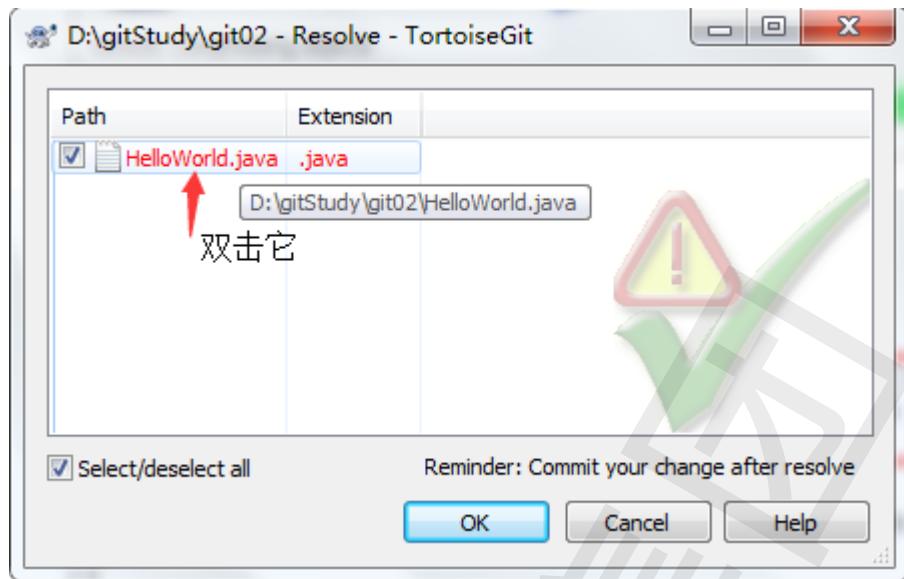




- 提示需要合并



- 点击了上图中resolve按钮弹出的窗口



- 进行合并代码, 最后点 Mark as resolved ,

解决后, 单击它, 表示已经解决冲突

MERGE_HEAD (origin/master)

```

1 public class HelloWorld {  
2     public void test2() {  
3         System.out.println("我是一位学习, 我的作业完成了");  
4     }  
5 }

```

远程库master分支上面的代码

HEAD

```

1 public class HelloWorld {  
2     public void test2() {  
3         System.out.println("我是老师, 同学们直接在下面实现该方法");  
4     }  
5 }

```

这一个是本地当前版本的代码

* Merged - HelloWorld.java

```

1 public class HelloWorld {  
2     public void test2() {  
3         System.out.println("我是老师, 同学们直接在下面实现该方法");  
4         System.out.println("我是一位学习, 我的作业完成了");  
5     }  
6 }

```

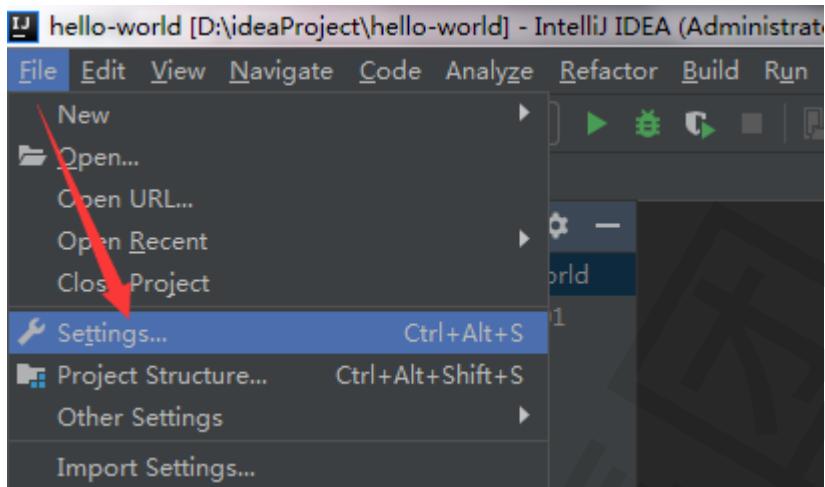
是最终合并的代码

- 再重新提交到本地库, 再提交到远程序

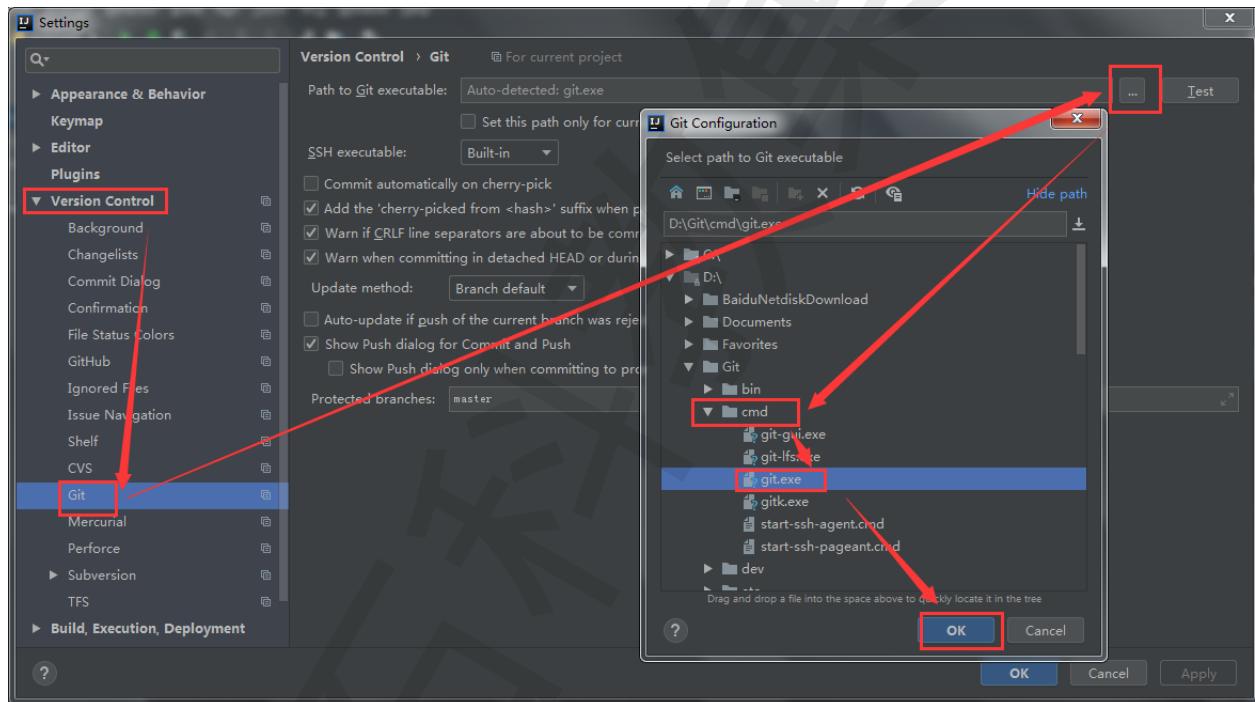
10. 基于IntelliJ IDEA的Git 操作

10.1 IDEA配置Git

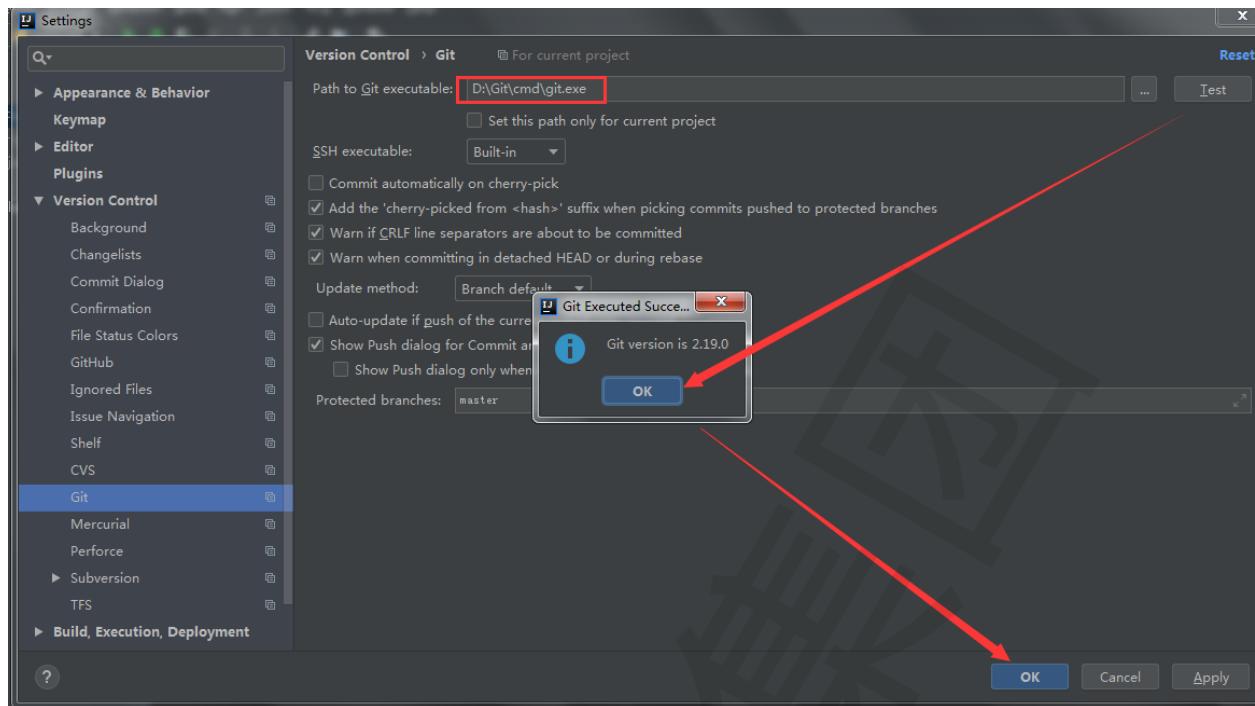
- 打开 `Settings` 窗口



- 选择本地安装的 `git.exe`

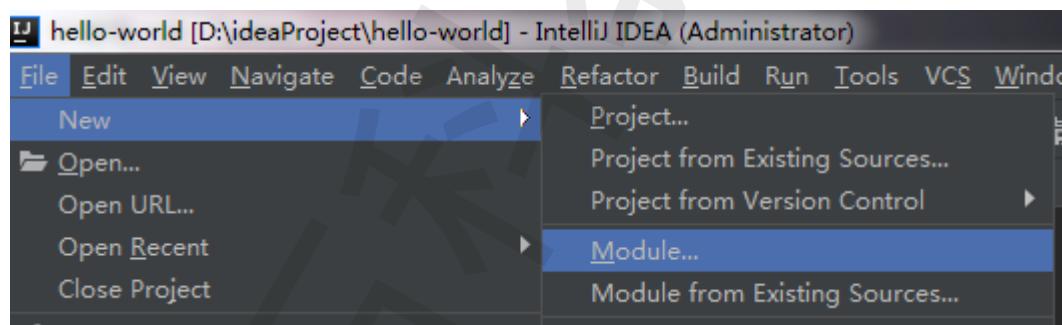


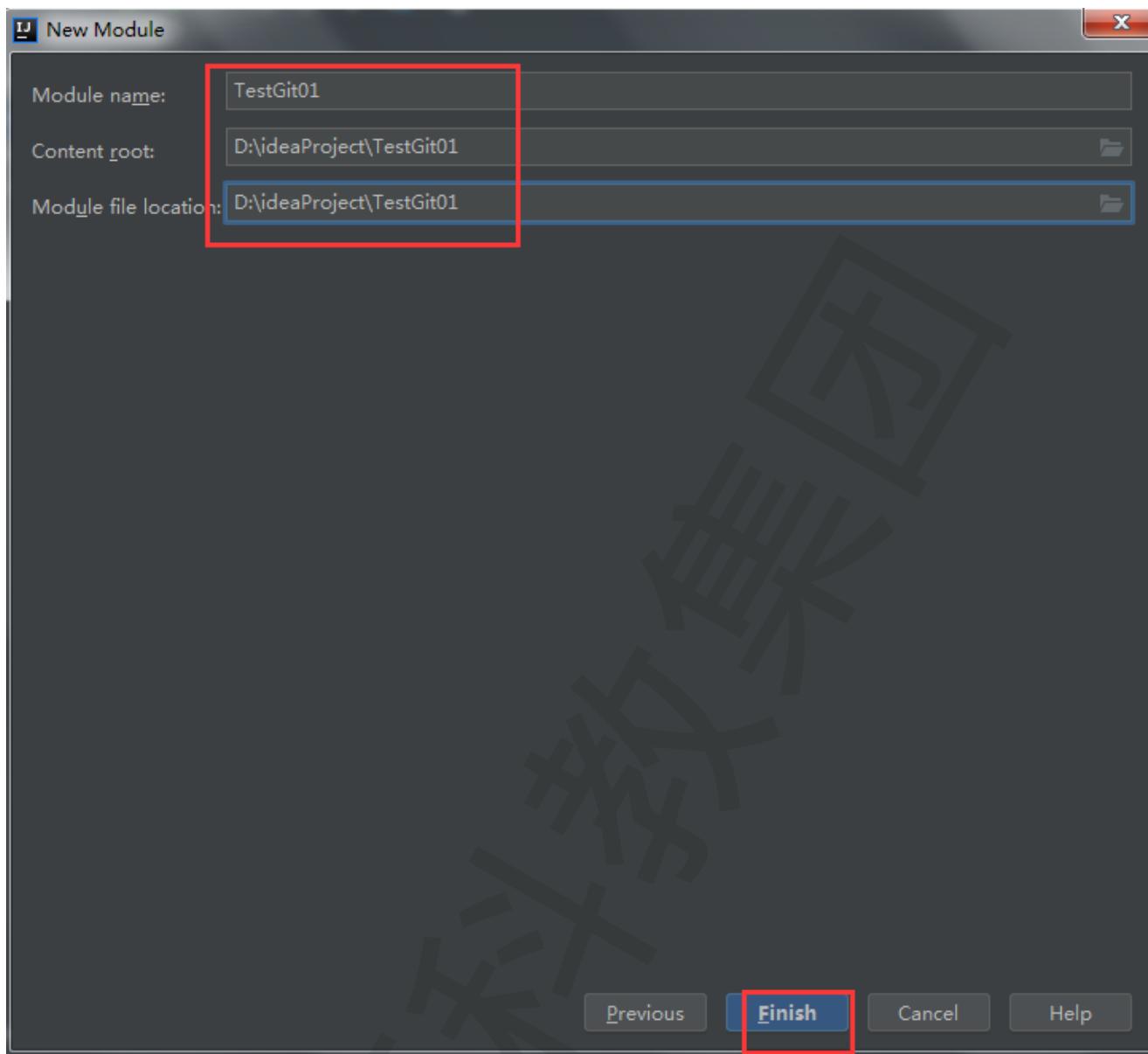
- 点击 `Test` , 如果弹出显示 git版本号 , 则说明配置成功 , 点击“OK”即可



10.2 创建项目并提交到本地库

10.2.1 创建项目



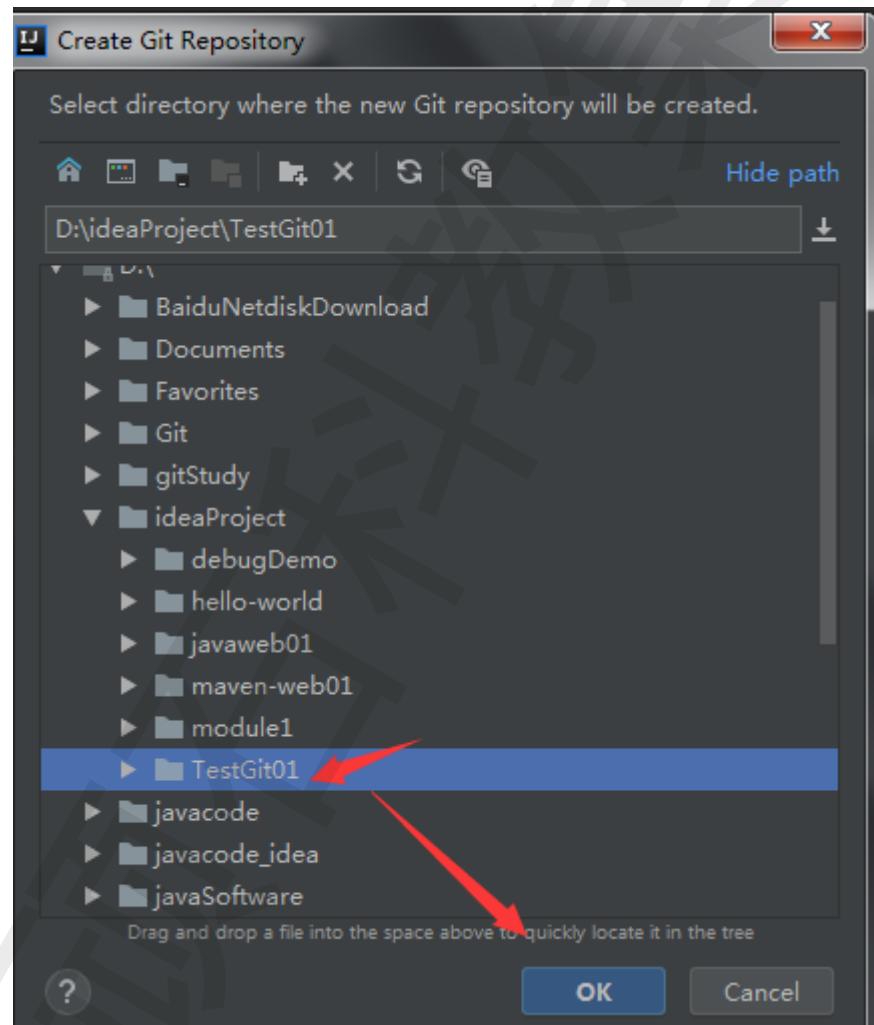
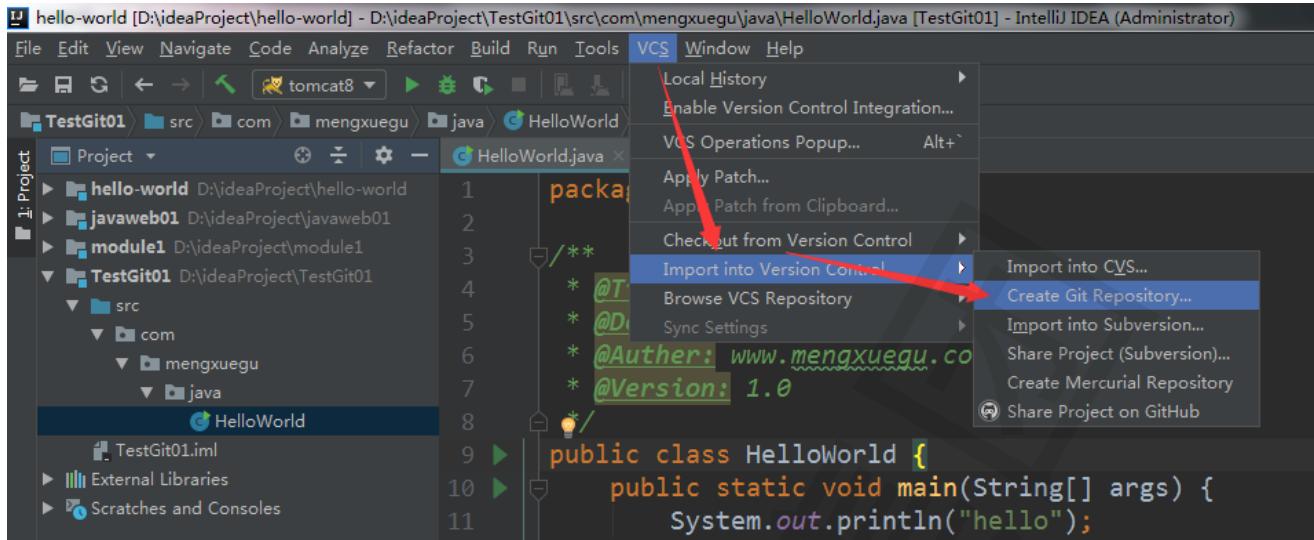


10.2.2 将项目提交到本地库

```
package com.mengxuegu.java;
/**
 * @Title: HelloWorld
 * @Description: com.mengxuegu.java
 * @Author: www.mengxuegu.com
 * @Version: 1.0
 */
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello");
    }
}
```

The screenshot shows the IntelliJ IDEA interface with the 'Project' tool window open. The 'HelloWorld.java' file is selected in the editor. The code defines a class 'HelloWorld' with a single static method 'main' that prints 'hello' to the console. The code includes JavaDoc annotations for Title, Description, Author, and Version. The 'Project' tool window on the left shows other projects like 'hello-world', 'javaweb01', and 'module1'.

- 选择项目名



- 如下图,颜色变为红色,则表示已经初始化好了本地库

```

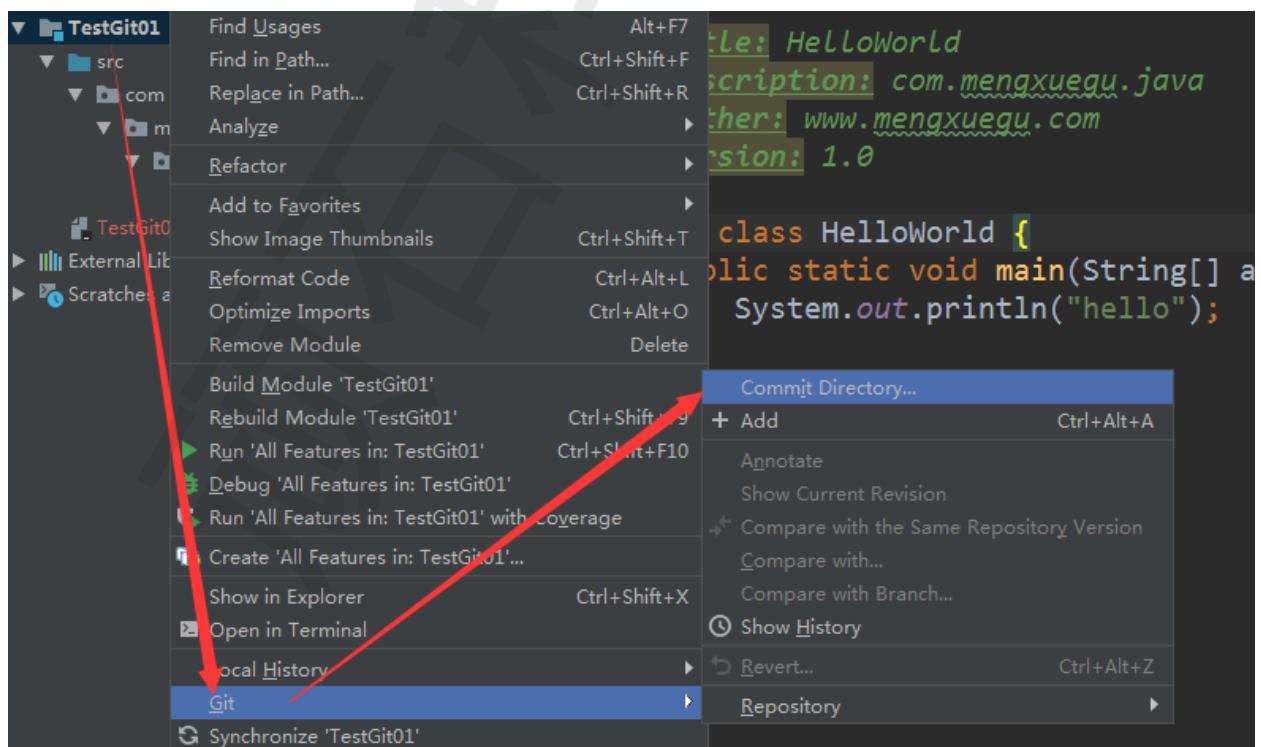
package com.mengxuegu.java;

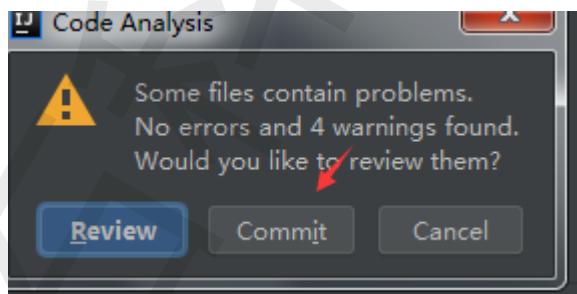
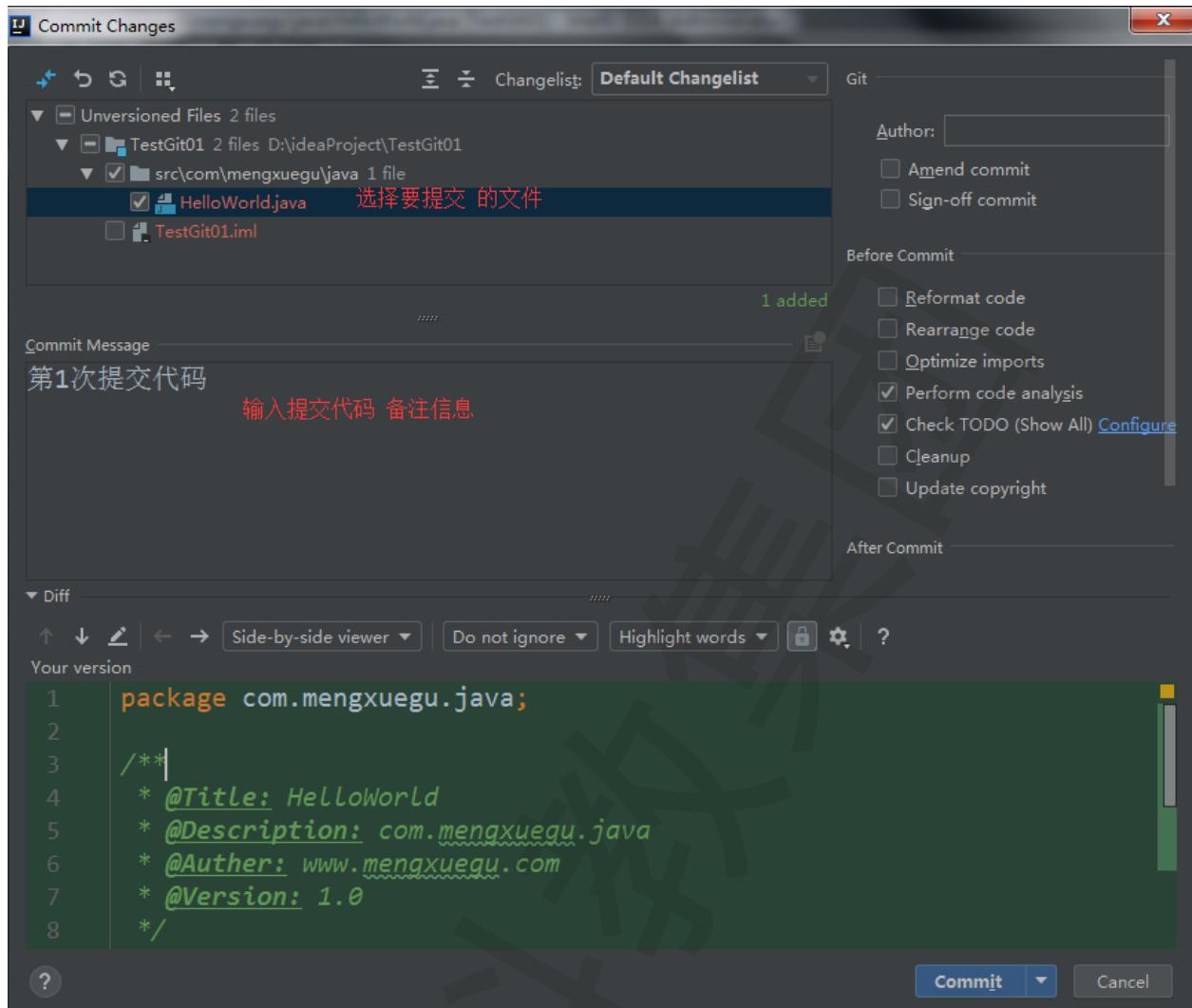
/**
 * @Title: HelloWorld
 * @Description: com.mengxuegu.java
 * @Auther: www.mengxuegu.com
 * @Version: 1.0
 */
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello");
    }
}

```



- 提交代码暂存区和本地库





10.3 IDEA指定忽略文件

- 概念：IDEA构建项目的特殊文件与class文件目录 这些都是 IDEA 为了管理我们创建的工程而维护的文件，和开发的代码没有直接关系。最好不要在 Git 中进行追踪，也就是把它们忽略。

`*.iml` 以iml后缀结尾文件

`classes` 目录下所有文件

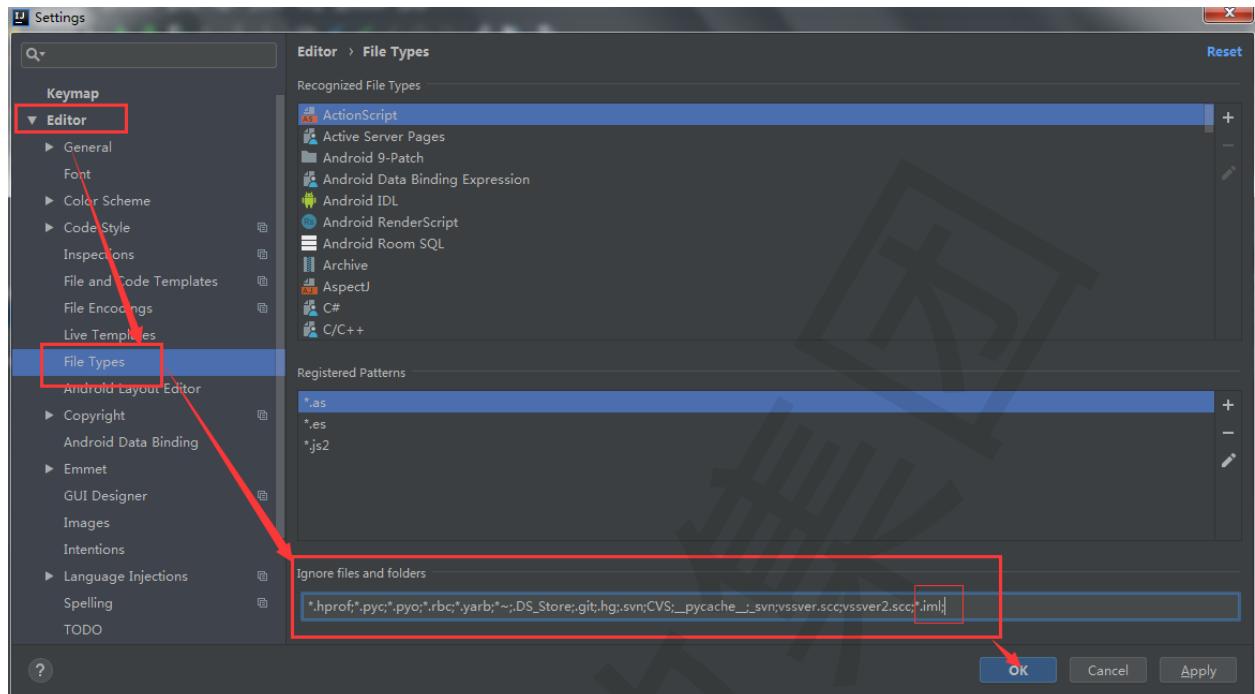
`target` 目录下所有文件

- 为什么要忽略特定文件呢？

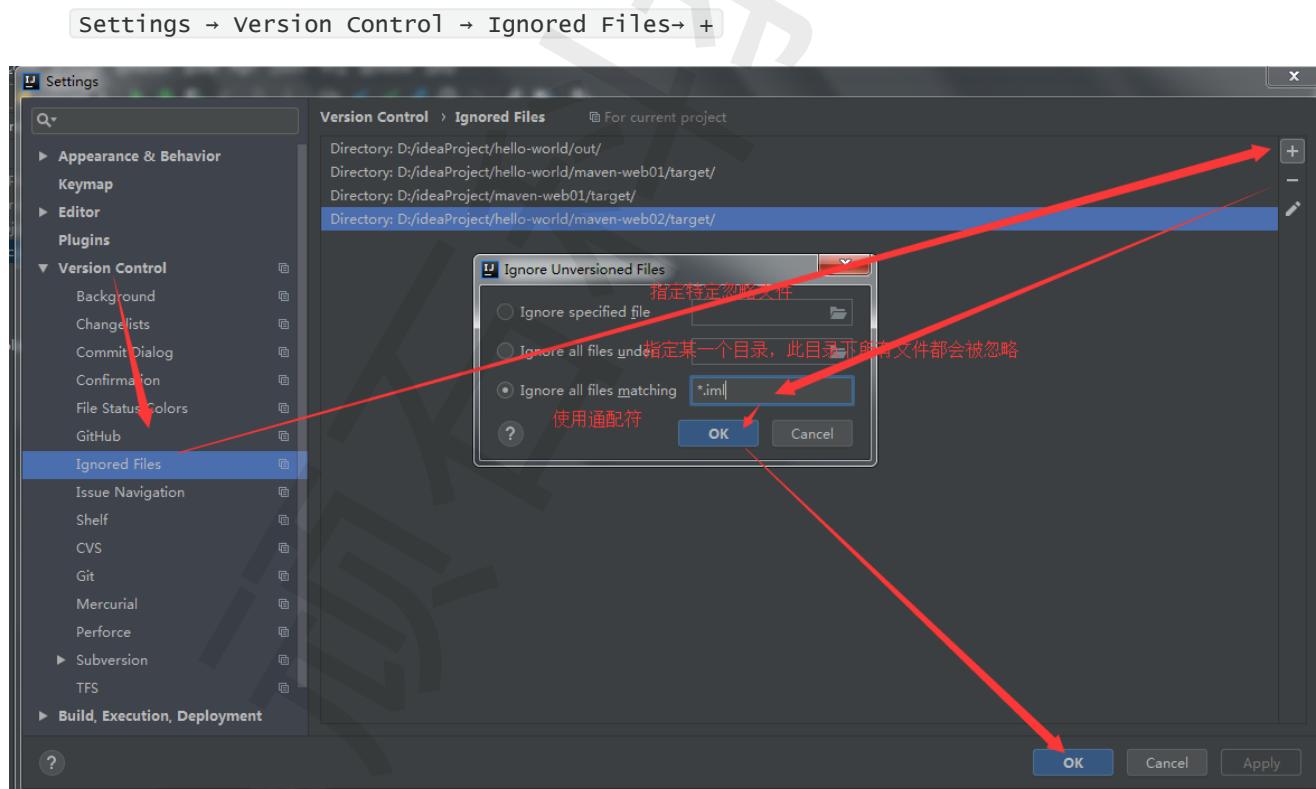
同一个团队中很难保证大家使用相同的 IDEA 工具，而 IDEA 工具不同时，相关工程特定文件就有可能不同。如果这些文件加入版本控制，那么开发时很可能需要为了这些文件解决冲突。

- 指定忽略的置文件或目录：

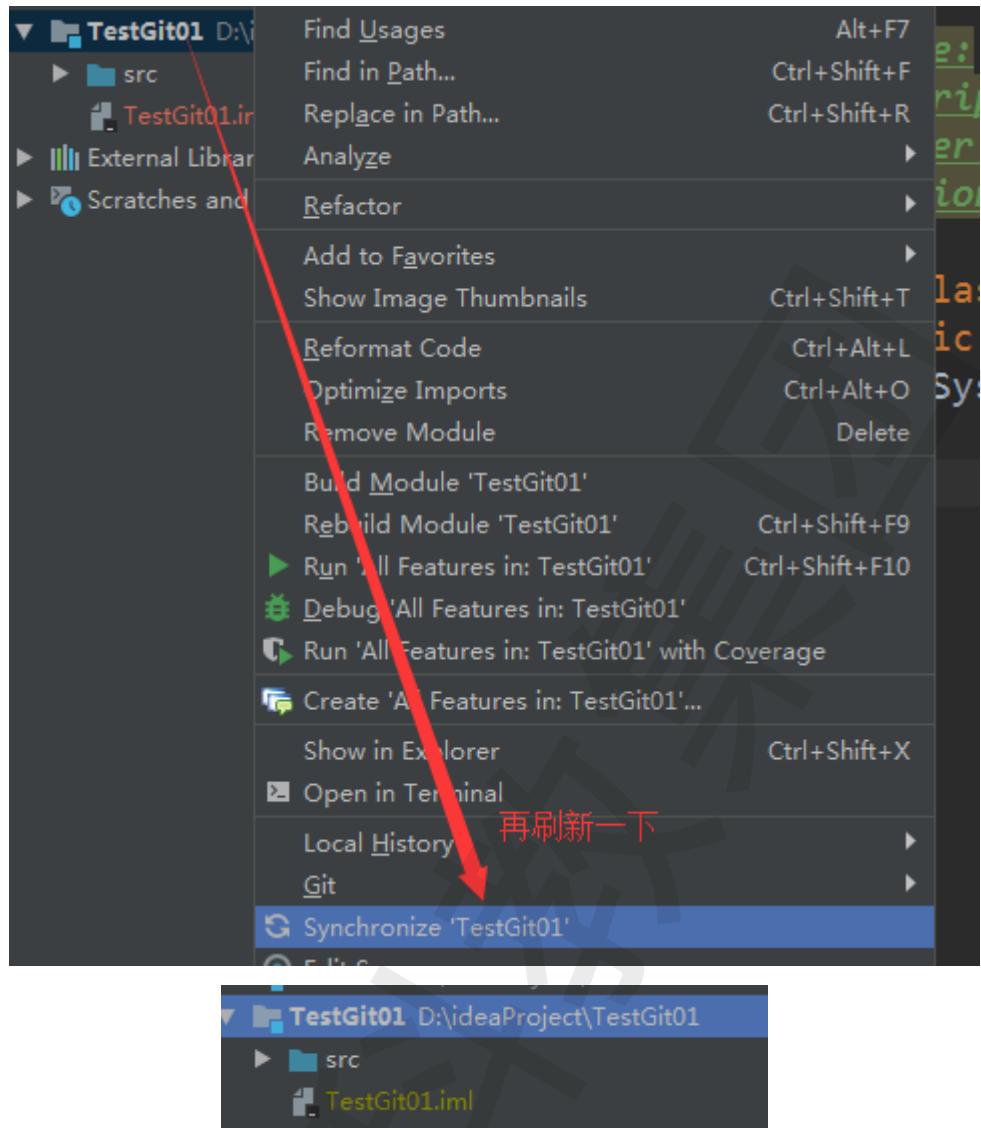
- 方式1：忽略指定文件，在IDEA列表中不显示出来，就不会被选择。
 - 打开Settings → Editor → File Types → Ignore files and folders下输入忽略文件



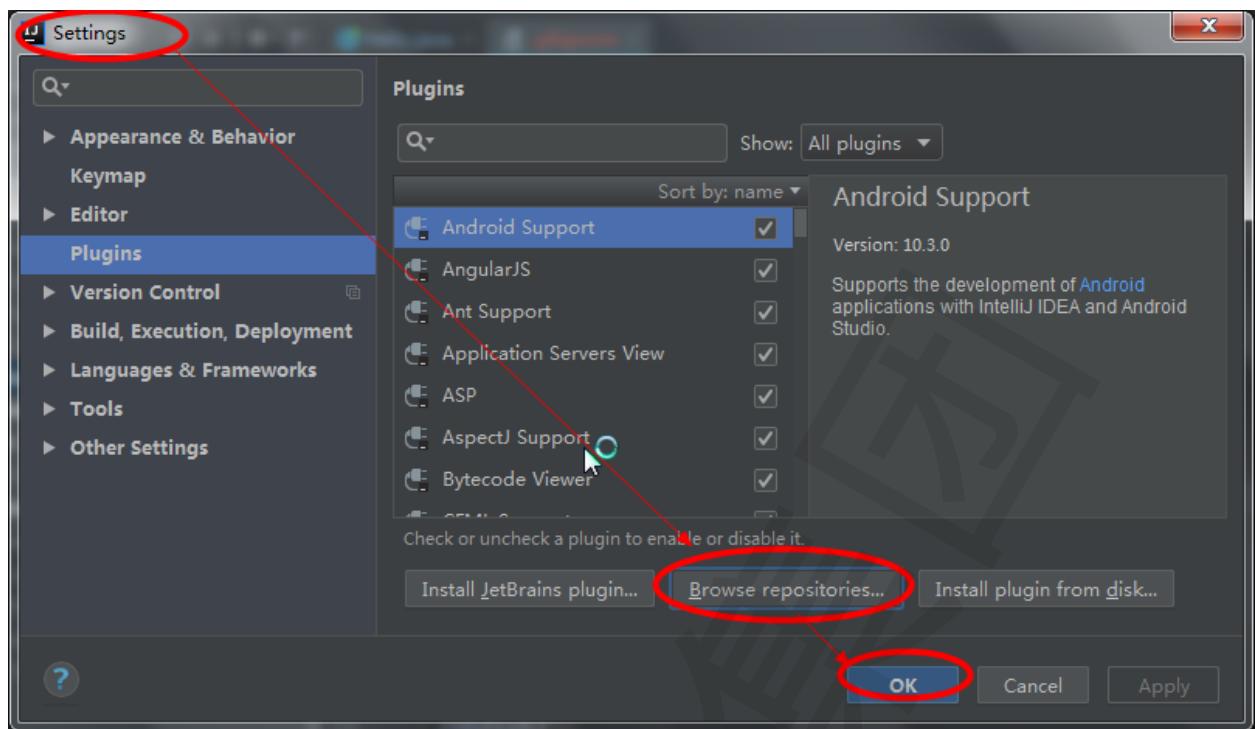
- 方式2：指定忽略文件或目录



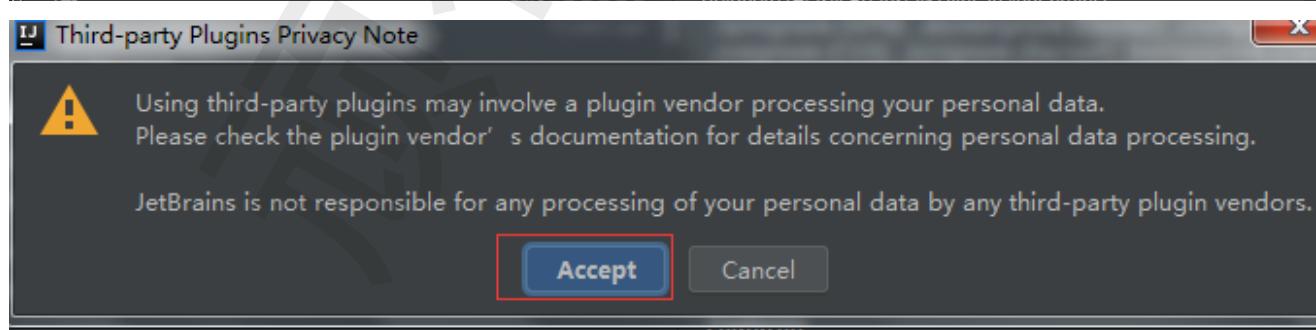
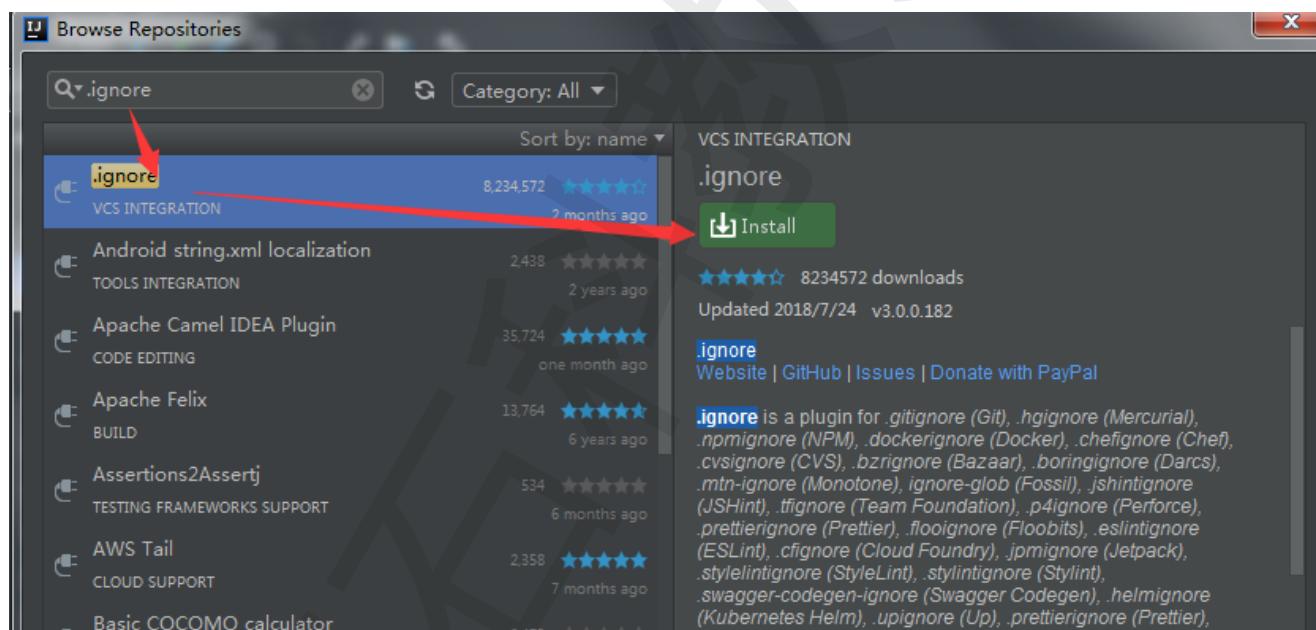
- 再刷新一下，就变成黄绿色



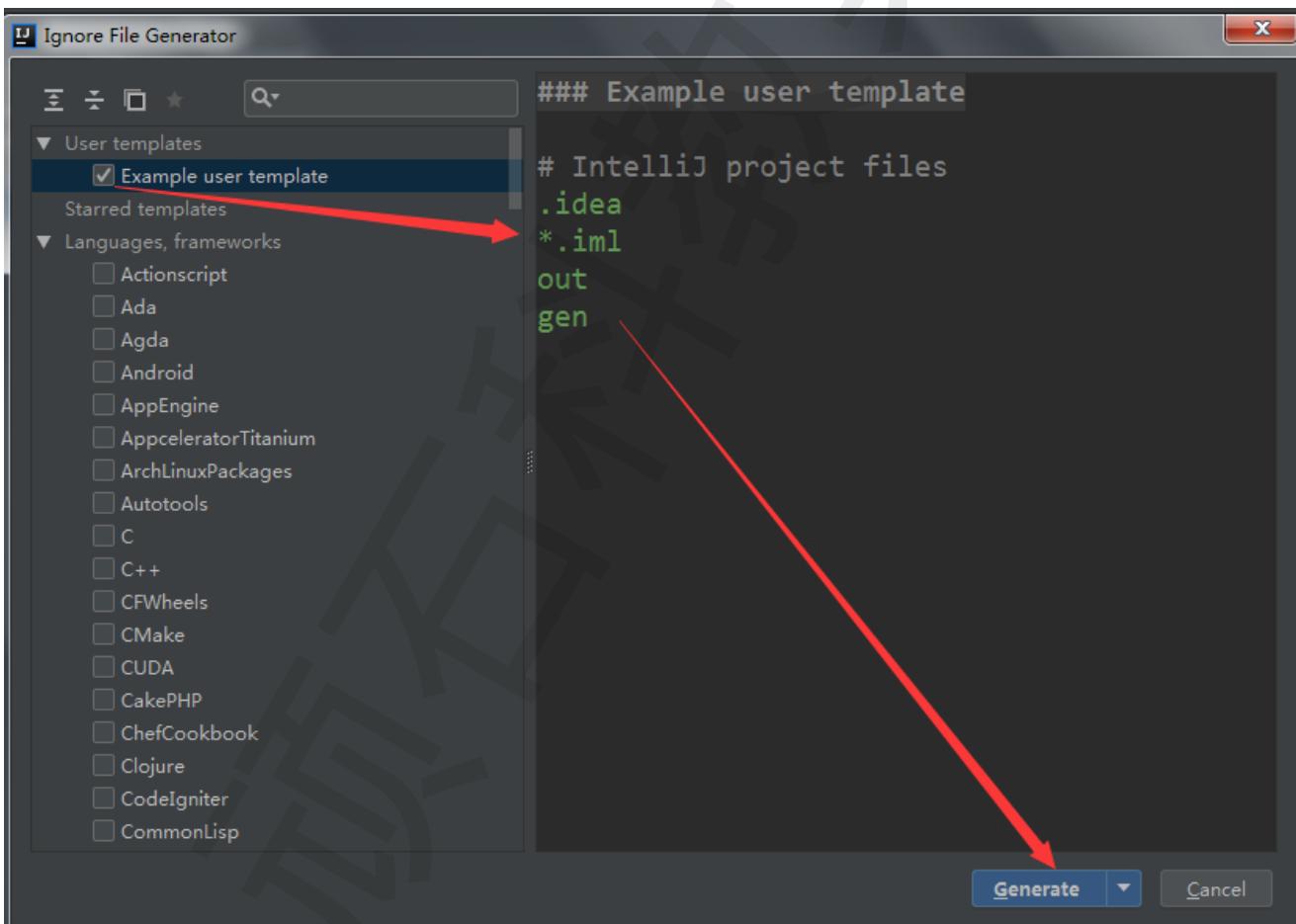
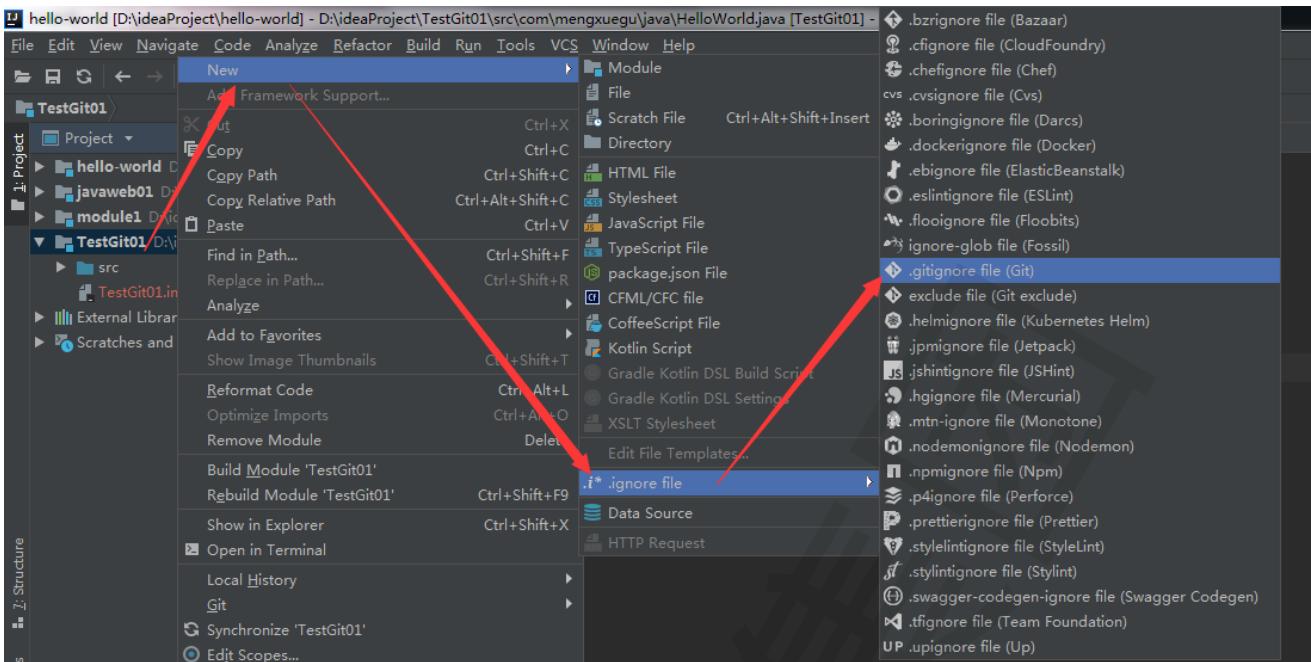
- 方式3：安装 .ignore 插件来忽略文件
 - 在左侧菜单找到Plugins，点击Browse repositories...



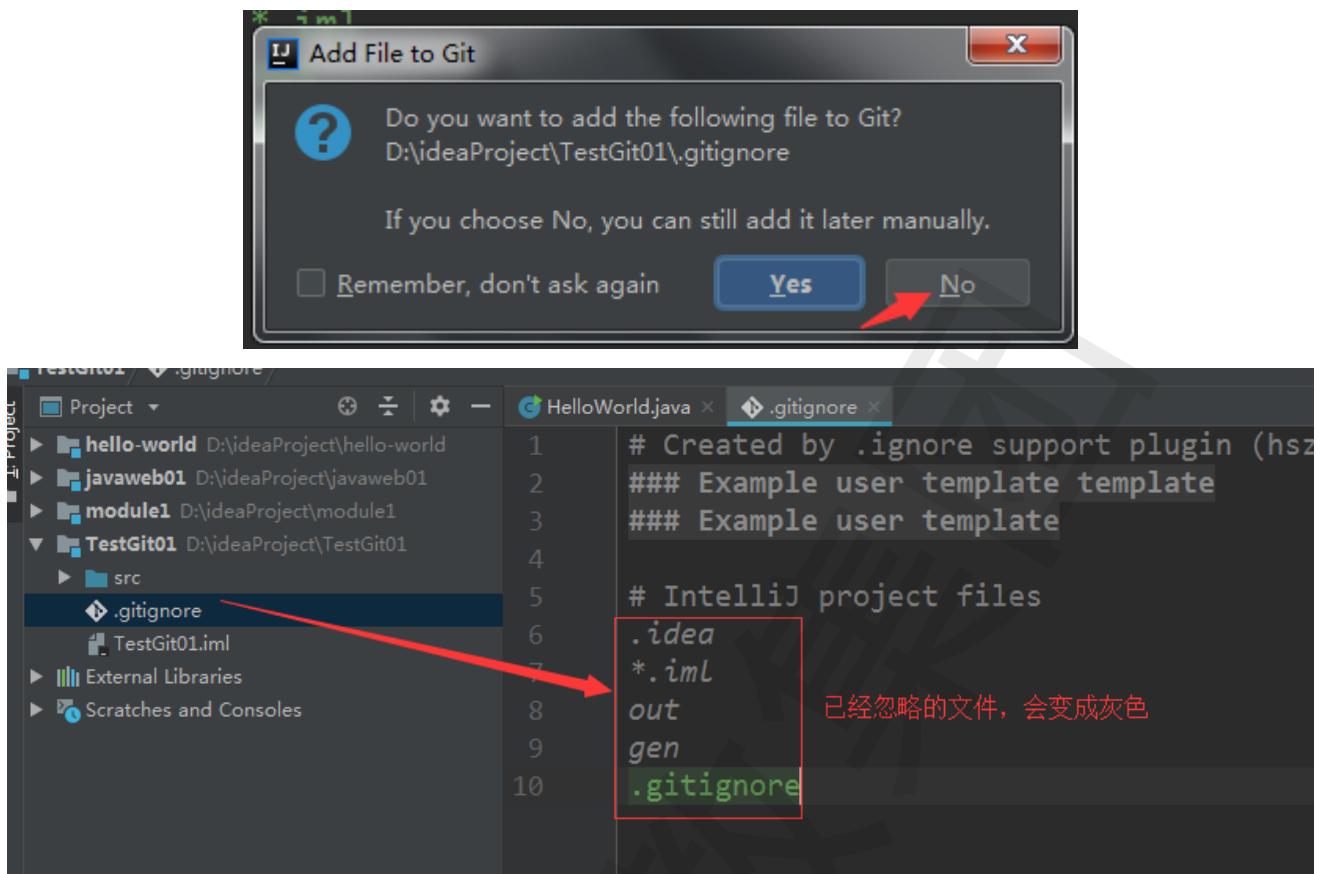
- 搜索 `.ignore` , 点击Install , 安装完成后 , 重启IDEA



重启完，在项目上右键->New ->.ignore file ->.gitignore file(Git)

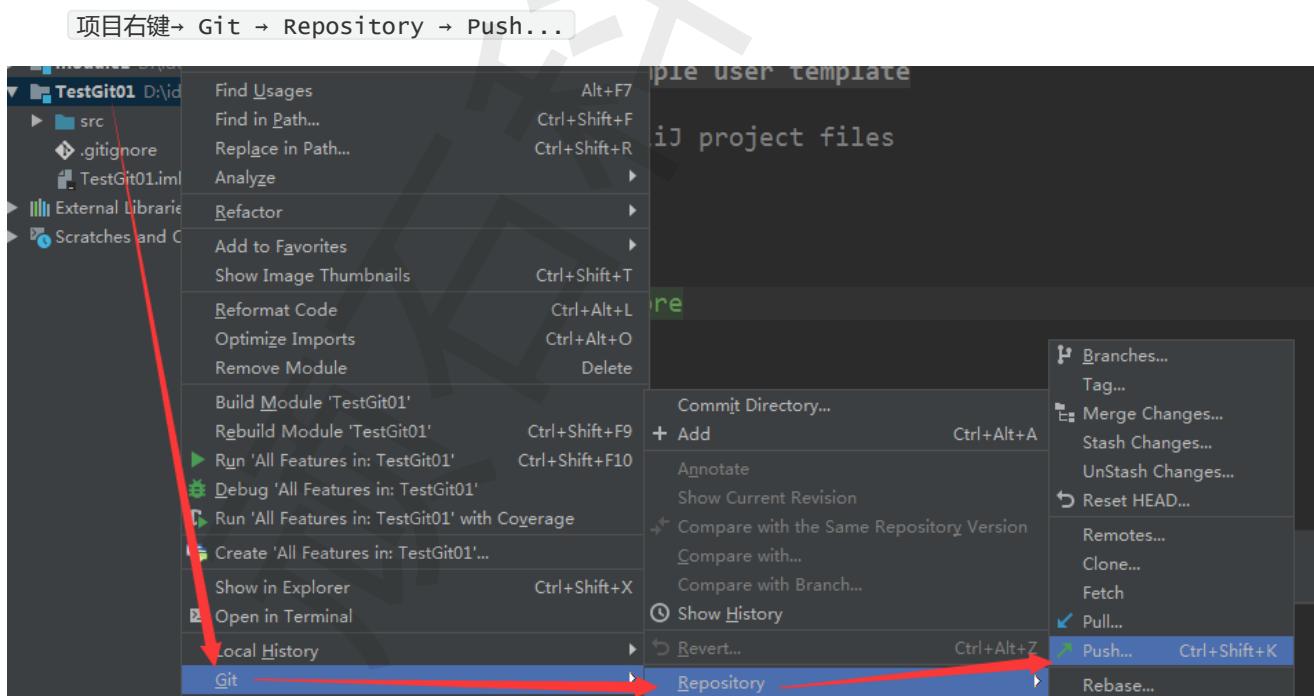


- 选择 NO, 不加入到Git管理中

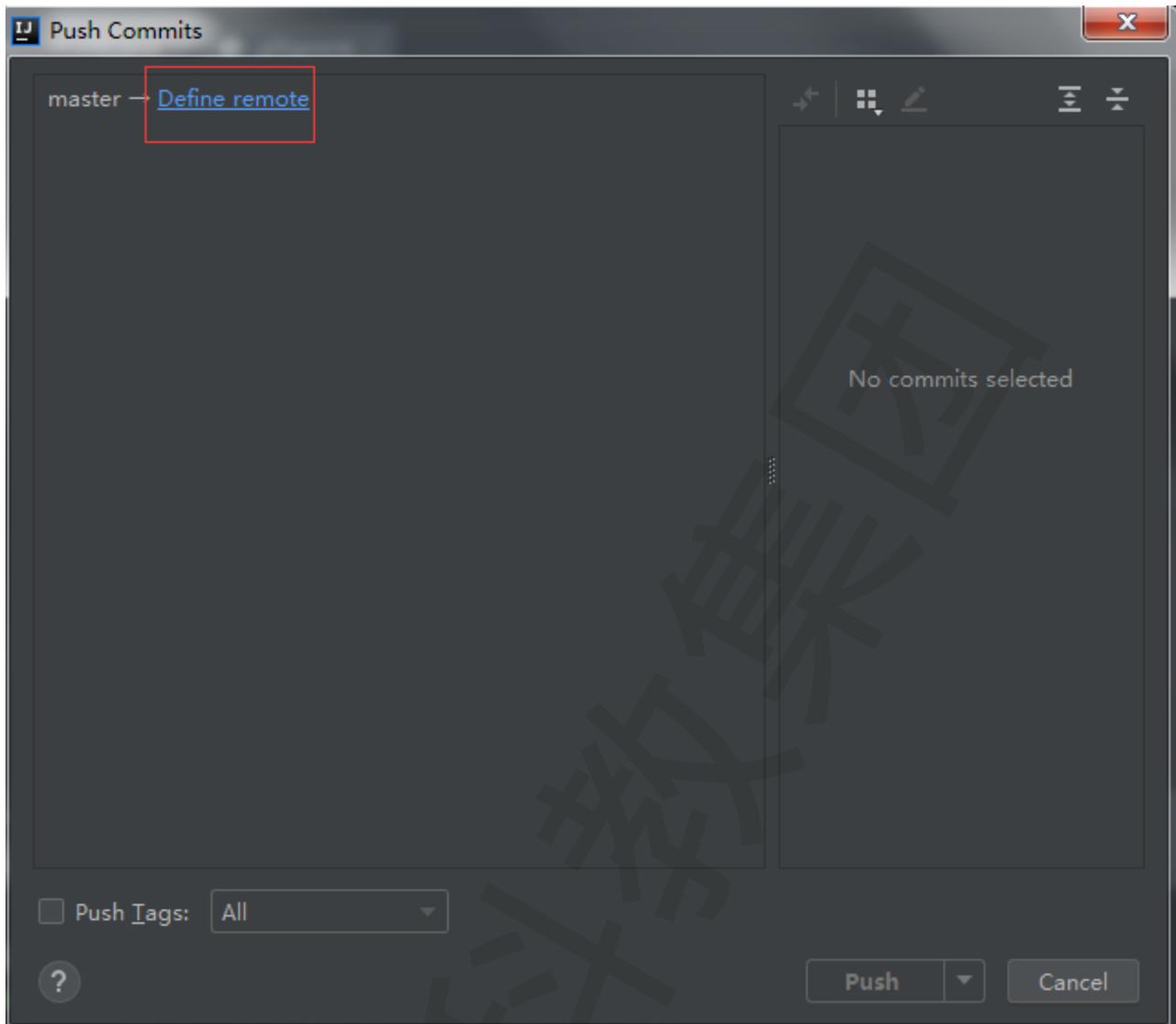


10.4 推送到远程库

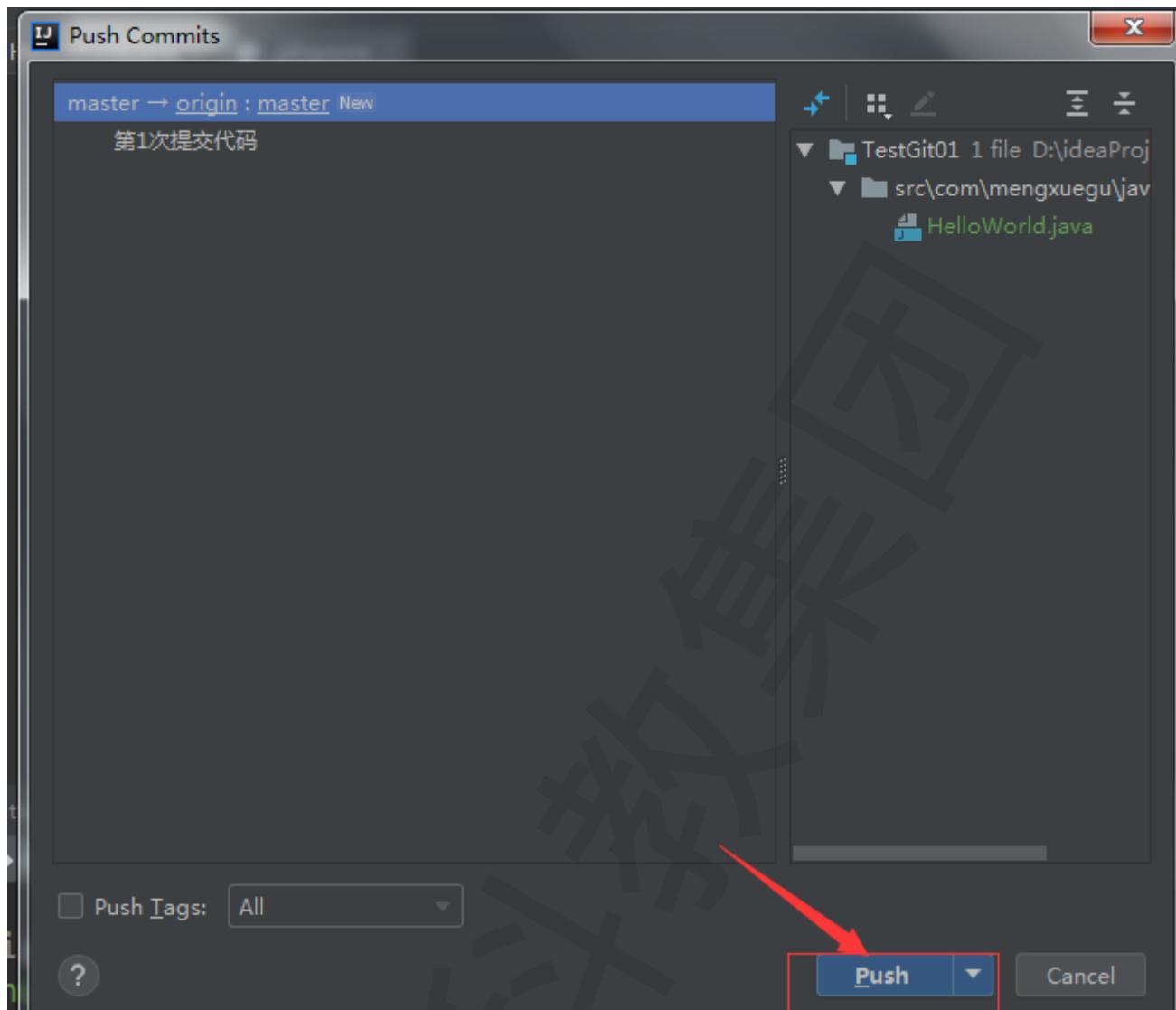
- 推送操作：



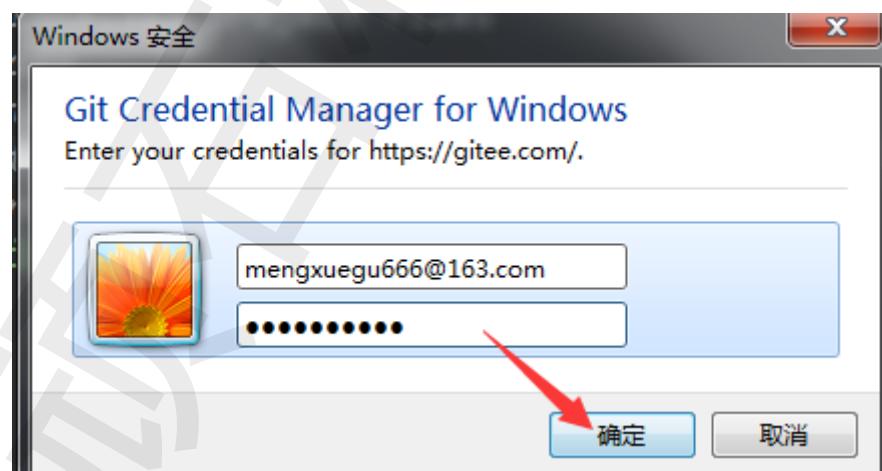
- 定义远端库信息，点击Define remote



- 设置远程库信息：
- 提交到远程库

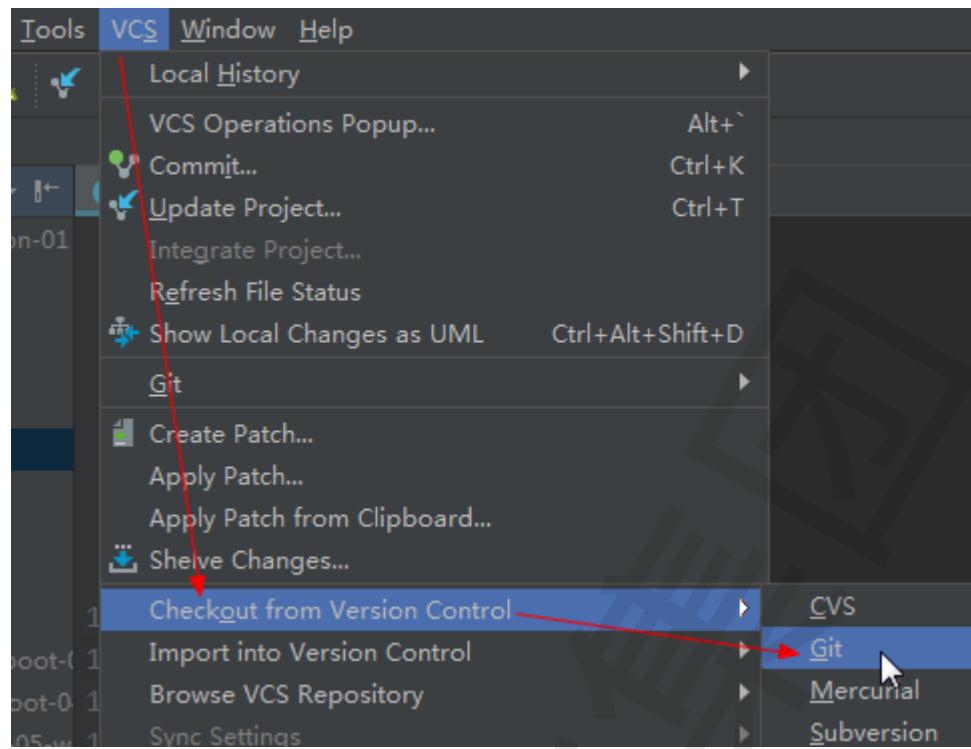


- 会弹出一个输入git远程服务器的用户名与密码的窗口，输入即可。



10.5 克隆远程仓库项目到本地

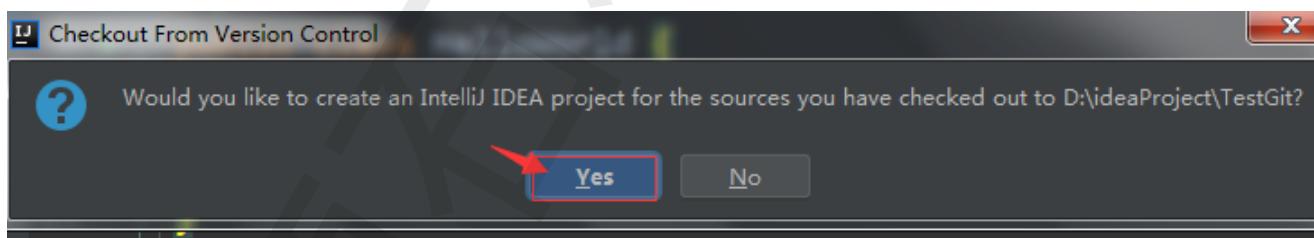
- 在工具栏上：VCS → Checkout from Version Control → Git



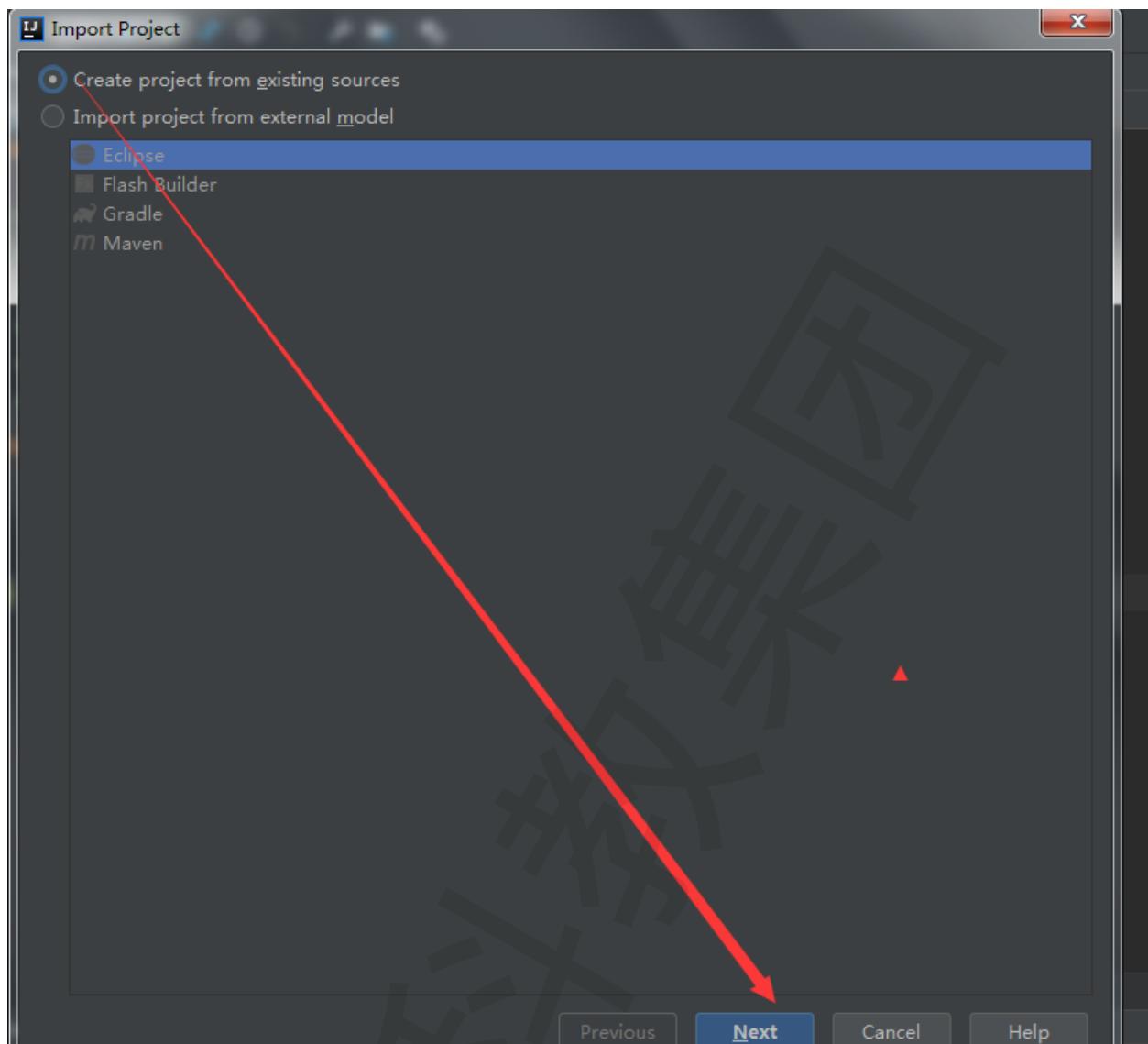
- 克隆，最后点 clone

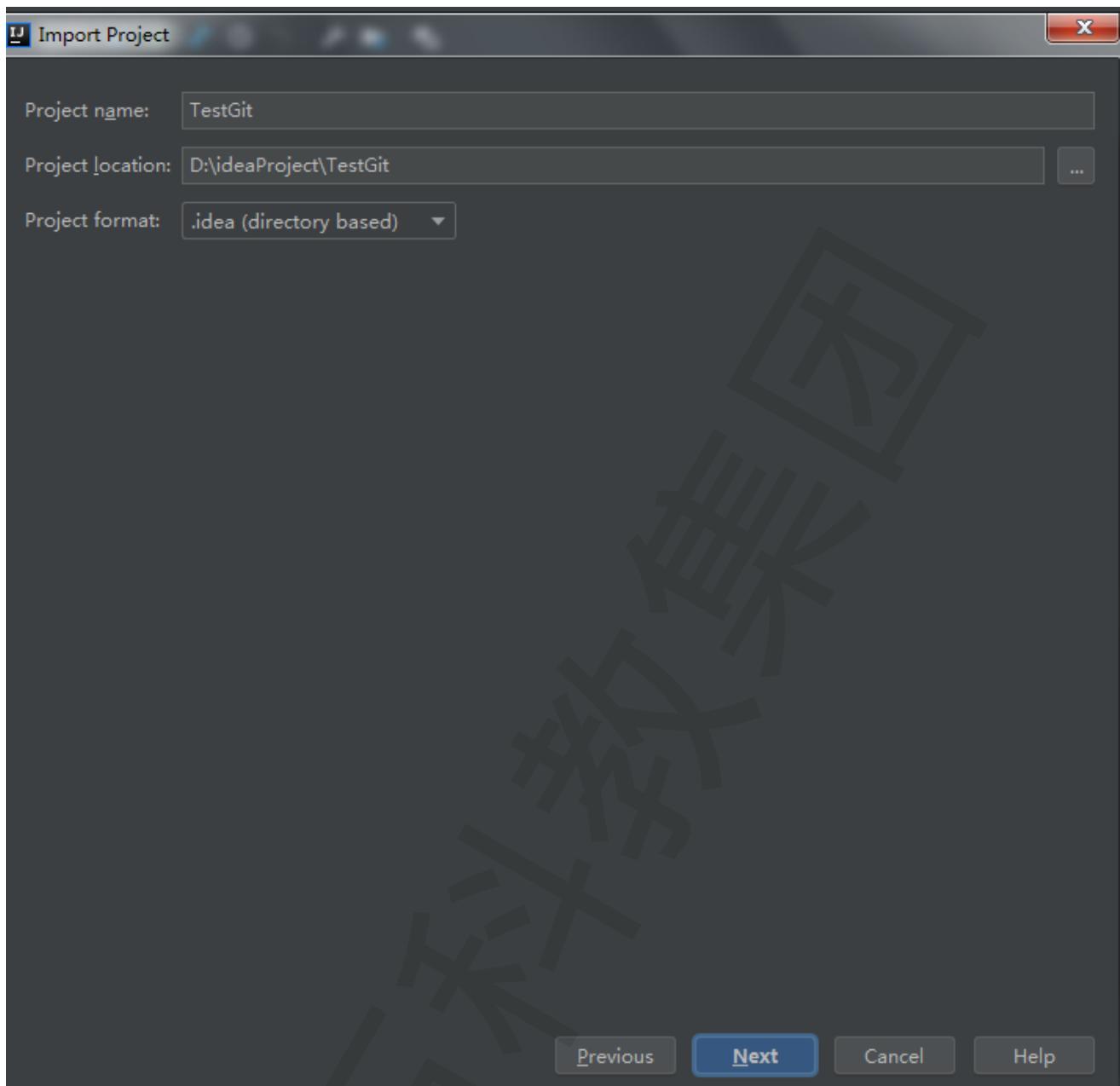


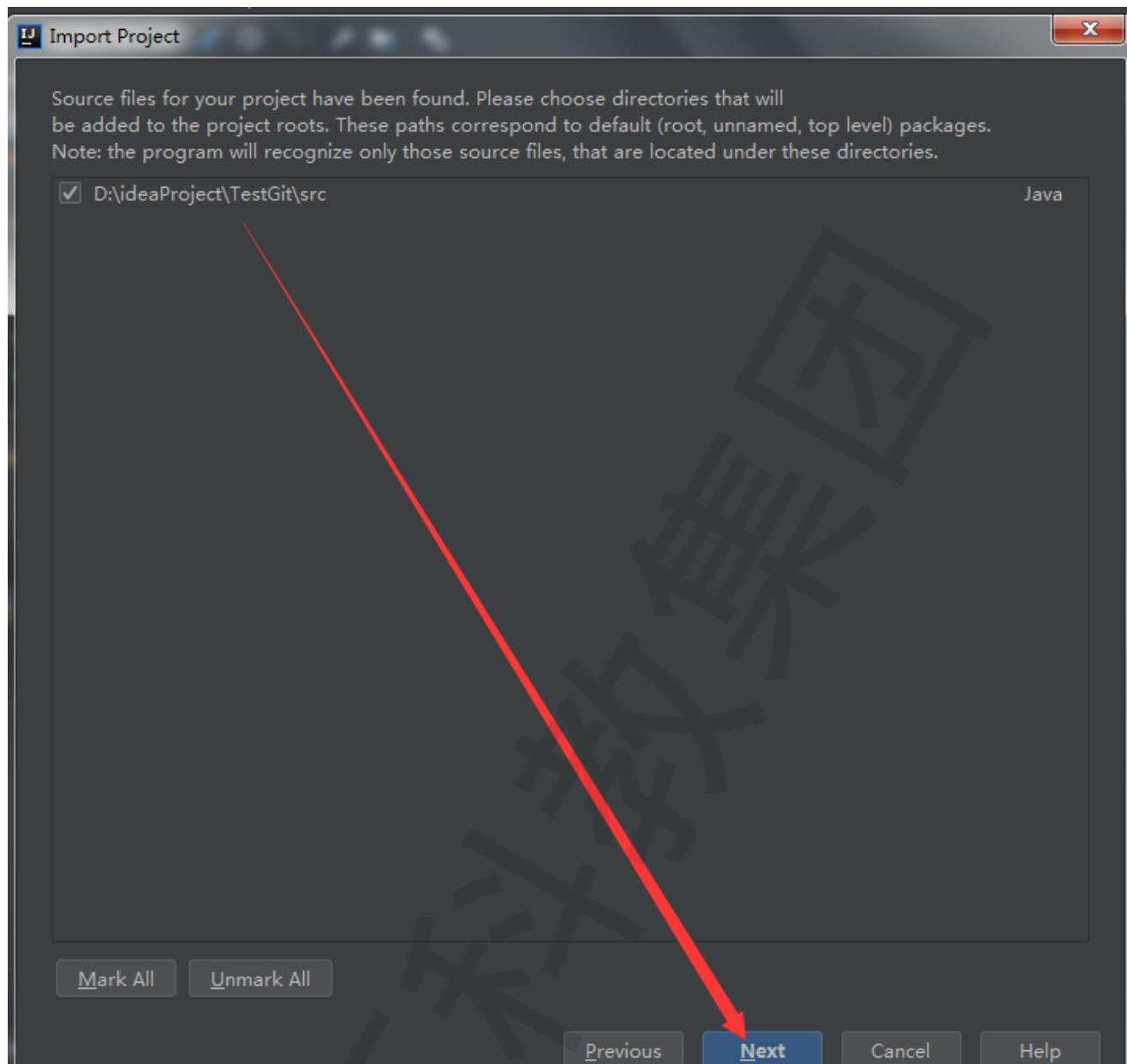
- 把克隆下来的项目创建到IDEA中

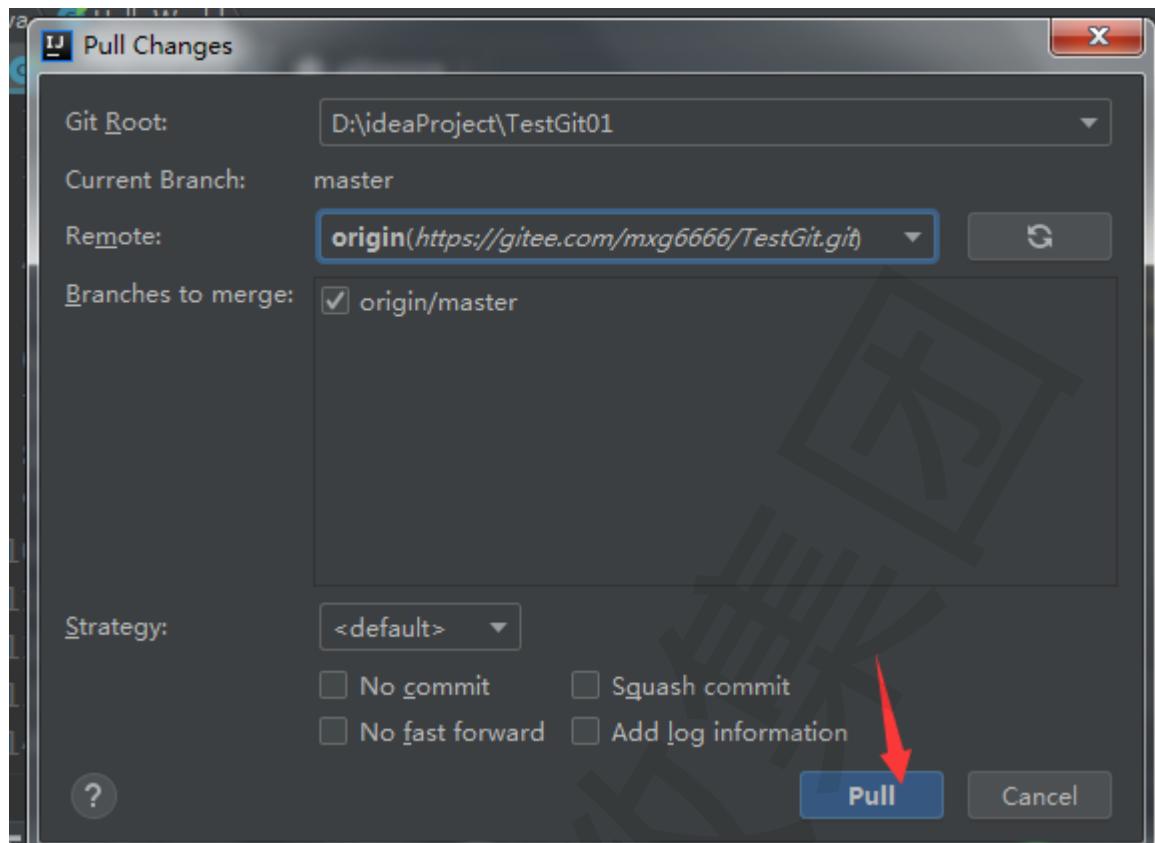


- 导入操作：





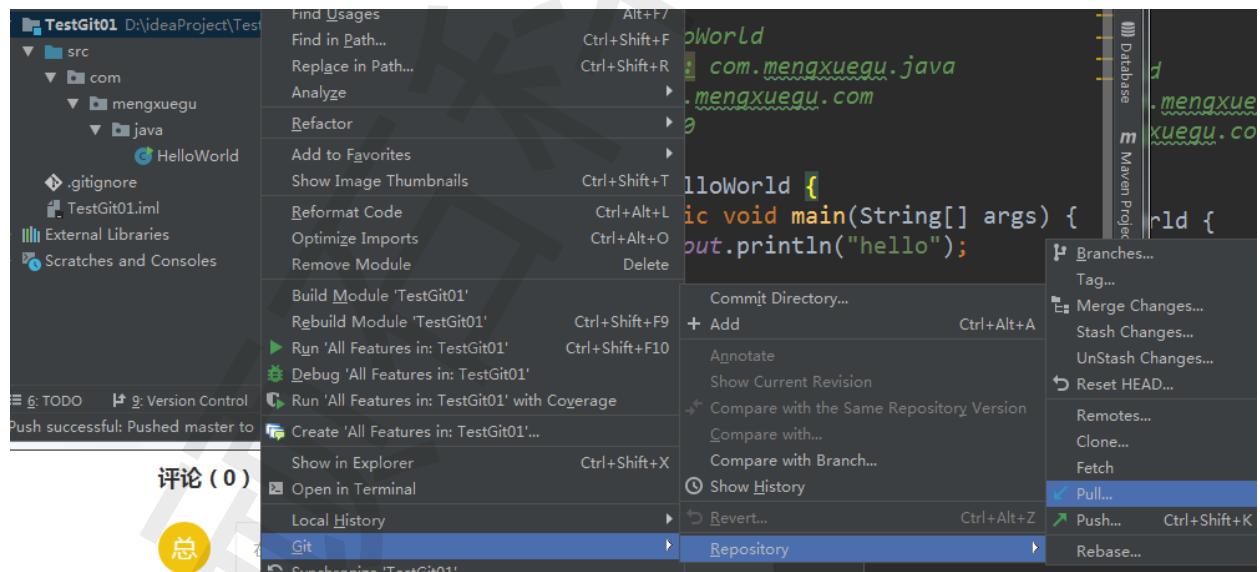




10.6 拉取远端库代码到本地

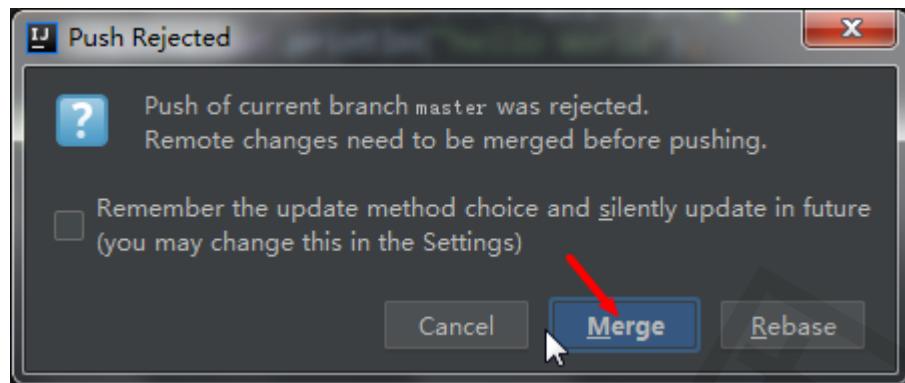
- 操作步骤：

项目右键 → Git → Repository → Pull

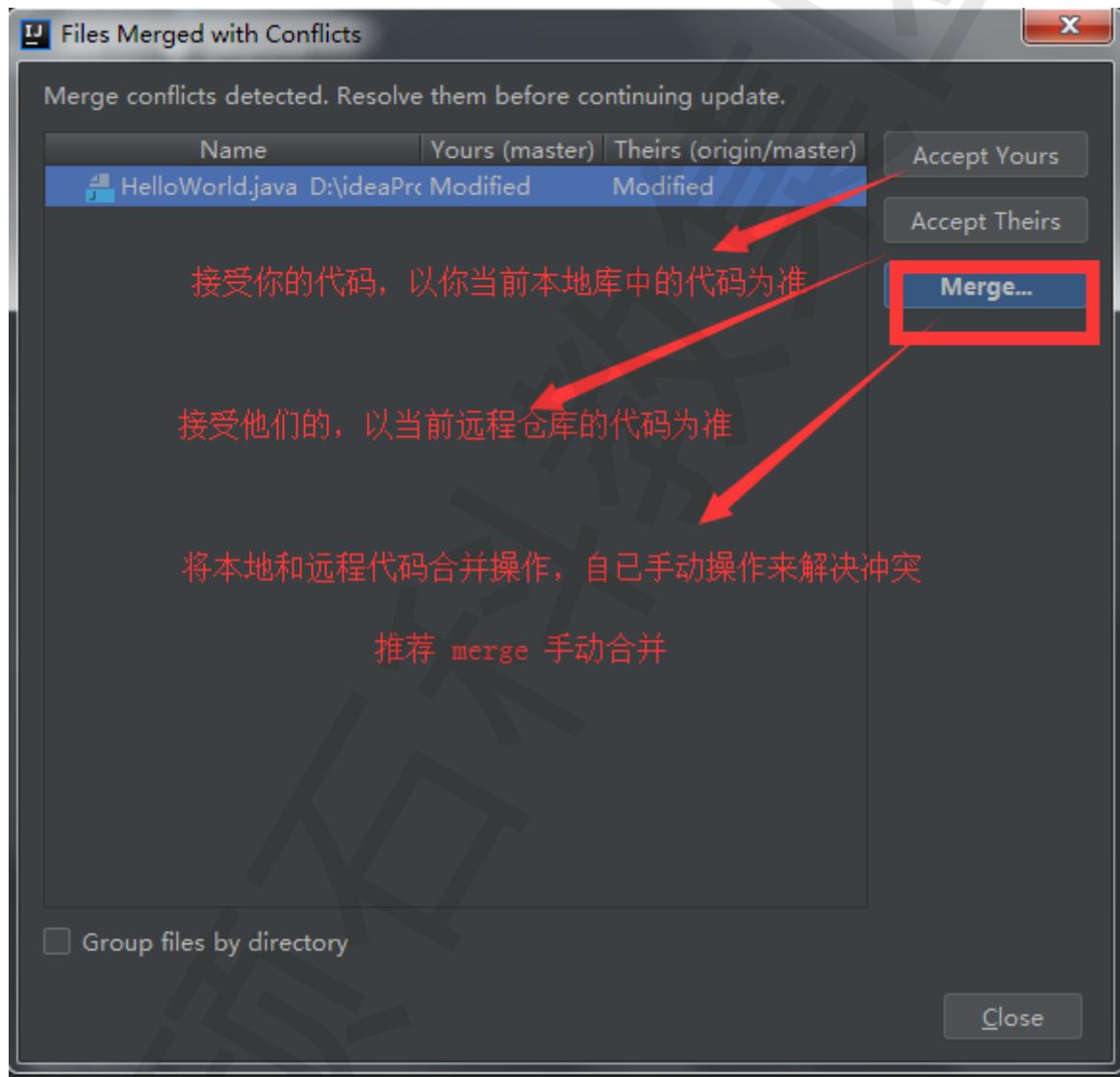


10.7 解决冲突

- 当Push时，出现以下窗口，说明有冲突，需要合并操作，点击merge进行合并



- 选择处理方式, 选择 Merge...



- 如下图, 左边是本地库, 中间是最终合并结果, 右边是远程仓库

Merge Revisions for D:\ideaProject\TestGit01\src\com\mengxuegu\java\HelloWorld.java

Apply non-conflicting changes: Left All Right | Highlight words | Result | 最终合并后的代码 | Changes from branch origin/master, revision 0a77dbc6

Your version, branch master | All conflicts resolved

```

* @Title: HelloWorld
* @Description: com.mengxuegu.java
* @Auther: www.mengxuegu.com
* @Version: 1.0
*/
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello");
        System.out.println("hell");
        System.out.println("hel");
        System.out.println("111");
        System.out.println("333");
    }
    public void add() {
        int i = 0;
        System.out.println(i);
    }
}

```

```

* @Title: HelloWorld
* @Description: com.mengxuegu.java
* @Auther: www.mengxuegu.com
* @Version: 1.0
*/
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hel");
        System.out.println("hel");
        System.out.println("111");
        System.out.println("333");
    }
    public void add() {
        int i = 0;
        System.out.println(i);
    }
    public void test() {
        System.out.println("123");
    }
}

```

Accept Left | Accept Right | Apply | Abort

- 合并完成后，点击Apply

Proj.x | Proj | .gitignore | HelloWorld.java | .gitignore x

hello-world D:\ideaProj
javaweb01 D:\ideaProj
module1 D:\ideaProject
TestGit01 D:\ideaProject

src
com
mengxuegu
java
HelloWorld.java
.gitignore
TestGit01.iml

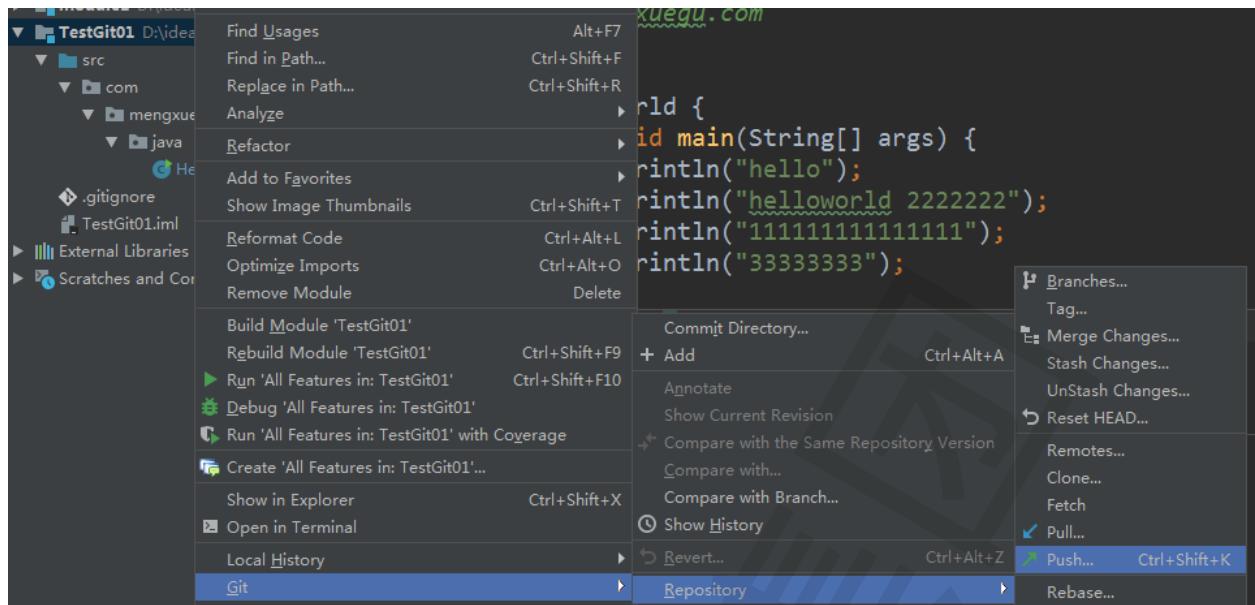
External Libraries
Scratches and Consoles

```

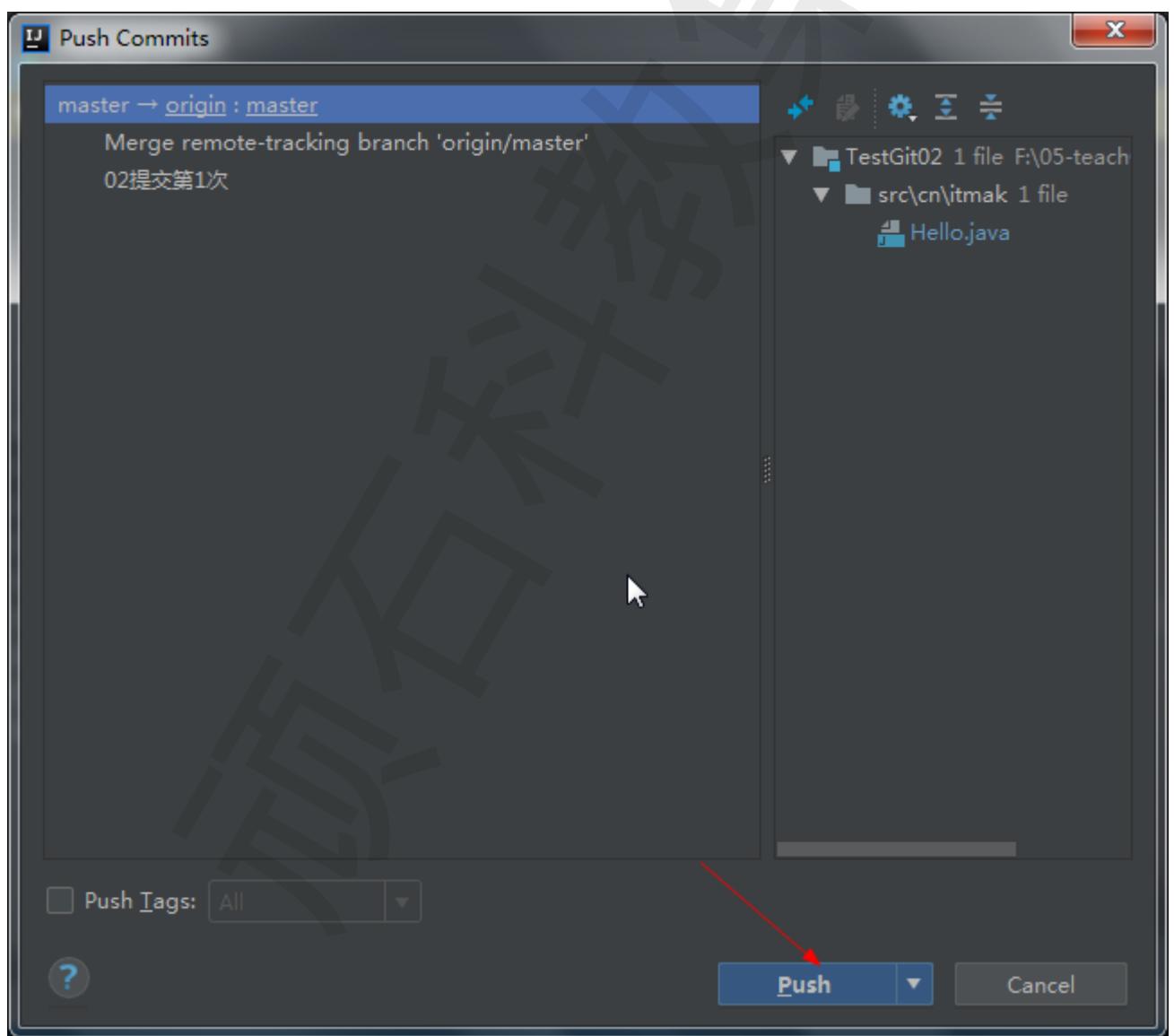
* @Title: HelloWorld
* @Description: com.mengxuegu.java
* @Auther: www.mengxuegu.com
* @Version: 1.0
*/
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello");
        System.out.println("hellworld 2222222");
        System.out.println("1111111111111111");
        System.out.println("33333333");
    }
    public void add() {
        int i = 0;
        System.out.println(i);
    }
    public void test() {
        System.out.println("123345");
    }
}

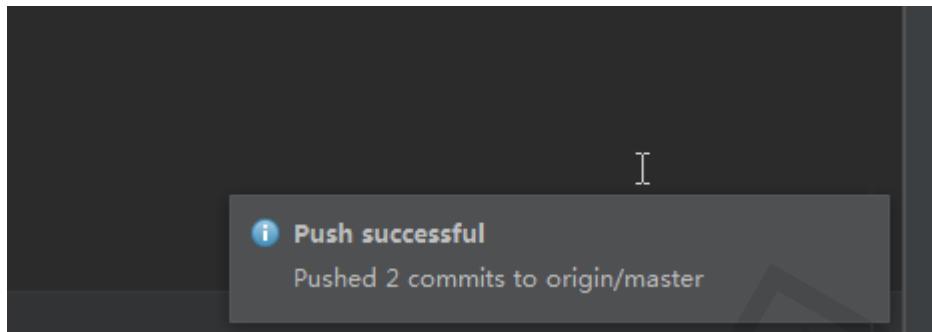
```

- 然后再推送远端库



- 推送到远程库：[项目右键 → Git → Repository → Push]





11. Git 工作流与实战演练

11.1 什么是工作流

- 因为项目开发中，多人协作，分支很多，虽然各自在分支上互不干扰，但是我们总归需要把分支合并到一起，而且真实项目中涉及到很多问题，例如版本迭代，版本发布，bug 修复等，为了更好的管理代码，需要制定一个工作流程，这就是我们说的工作流，也有人叫它分支管理策略。
- "工作流程"在英语里，叫做"workflow"或者"flow"，原意是水流，比喻项目像水流那样，顺畅、自然地向前流动，不会发生冲击、对撞、甚至漩涡。

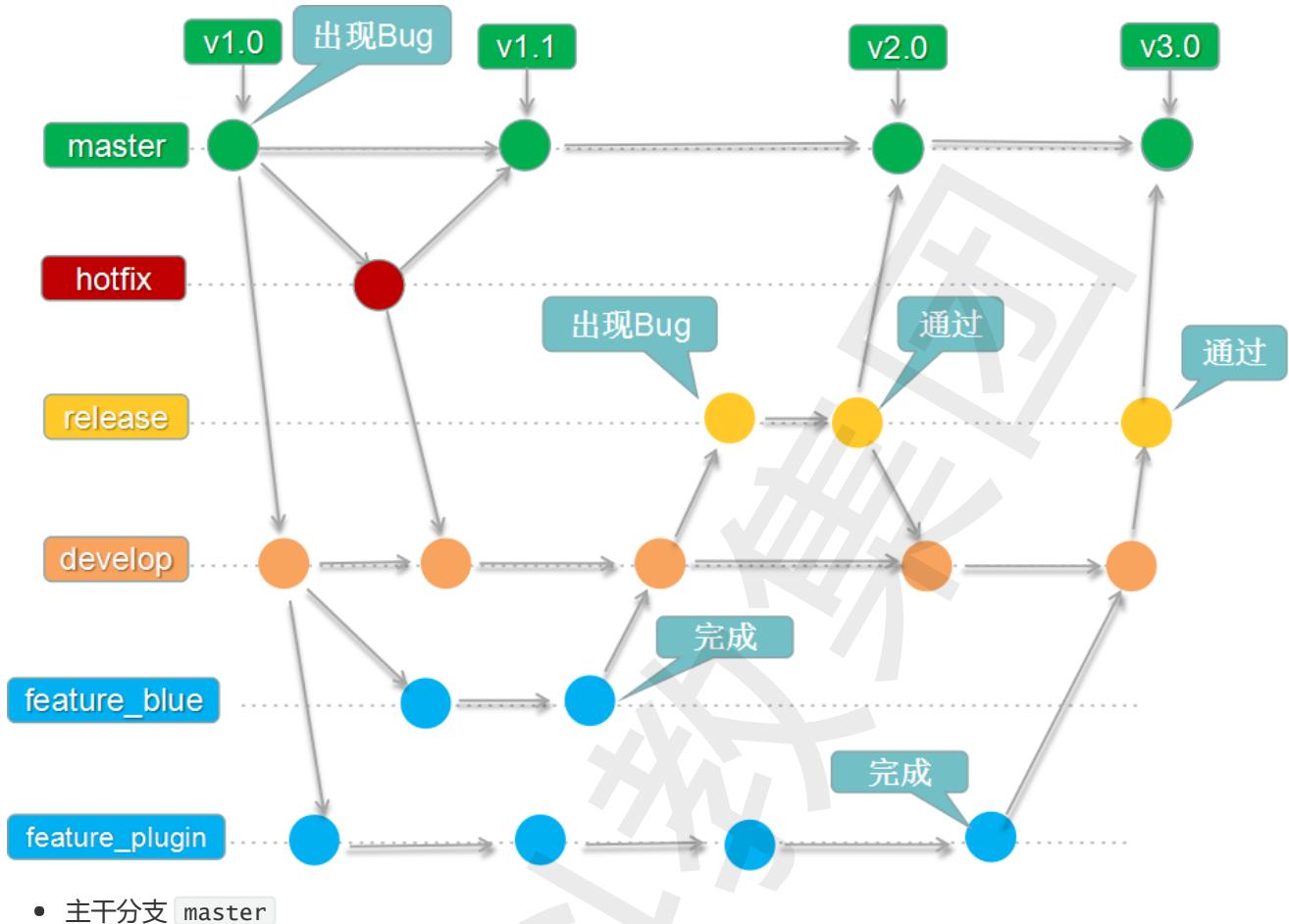
11.2 GitFlow 工作流说明

- Gitflow 工作流通过为功能开发、发布准备和维护设立了独立的分支，让发布迭代过程更流畅。严格的分支模型也为大型项目提供了一些非常必要的结构。



剩下要说明的问题围绕着这2个分支的区别展开。

11.3 分支种类



- 主干分支 `master`

主要负责管理正在运行的生产环境代码。永远保持与正在运行的生产环境完全一致。

- 热修复分支 `hotfix`

主要负责管理生产环境下出现的紧急修复的代码。从主干分支分出，修理完毕并测试上线后，并回主干分支。并回后，视情况可以删除该分支。

- 开发分支 `develop`

主要负责管理正在开发过程中的代码。一般情况下应该是最新的代码。

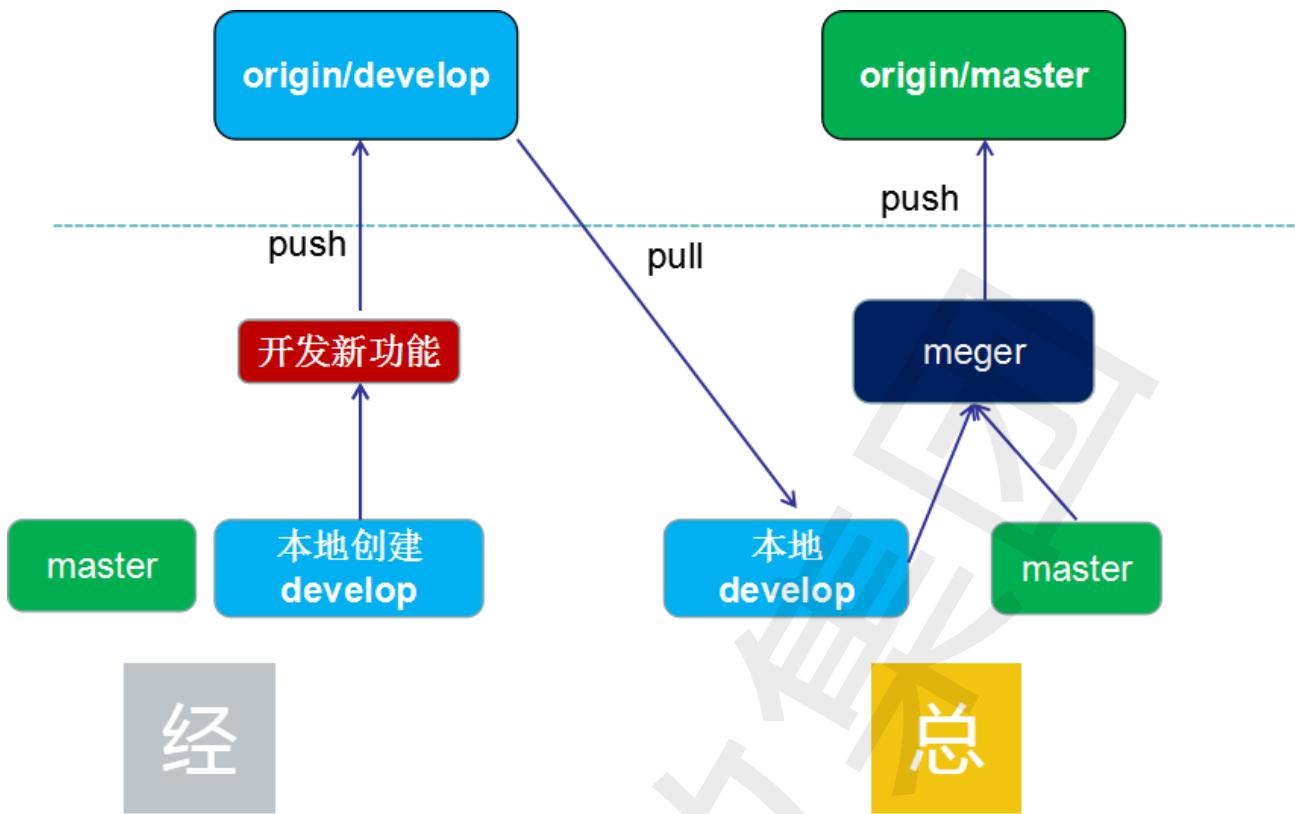
- 准生产分支（预发布分支） `release`

较大的版本上线前，会从开发分支中分出准生产分支，进行最后阶段的集成测试。该版本上线后，会合并到主干分支。生产环境运行一段阶段较稳定后可以视情况删除。

- 功能分支 `feature`

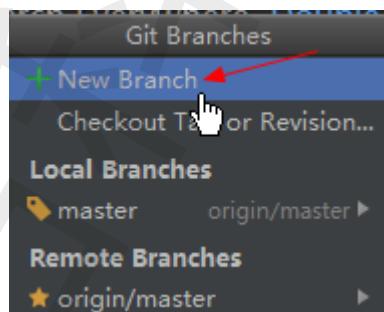
为了不影响较短周期的开发工作，一般把中长期开发模块，会从开发分支中独立出来。开发完成后会合并到开发分支。

11.4 工作流实战演练

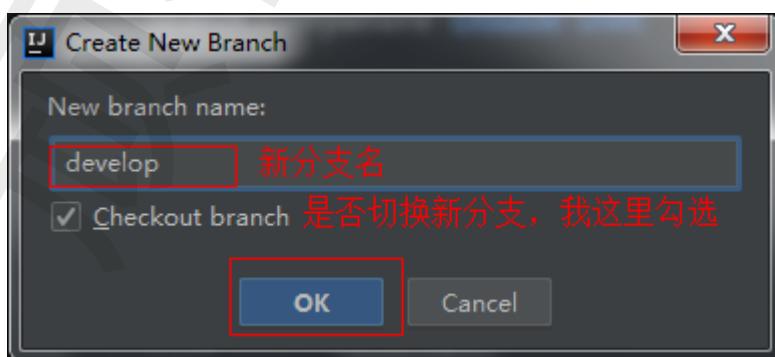


11.4.1 第1步：经理 创建开发分支

- 使用经理用户，在本地仓库中新创建一个develop分支
 - 项目右键 → Git → Repository → Branches
 - 点击 New Branch 创建分支



- 框中输入新分支名：develop，Checkout branch 勾选后，会马上切换到新创建的这个分支



- 右下角可查看到当前所在分支：develop

```
7 * @Version: 1.0
8 */
9 public class HelloWorld {
10    public static void main(String[] args) {
11        System.out.println("hello");
12        System.out.println("helloworld 2222222");
13        System.out.println("111111111111111");
14        System.out.println("333333333");
15        System.out.println("我是经理，我在添加新的功能 . . . . ");
16    }
17    public void add() {
18        int i = 0;
19        System.out.println(i);
20    }
21    public void test() {
22        System.out.println("123345");
23    }
24
25 }
```

HelloWorld

Event 1

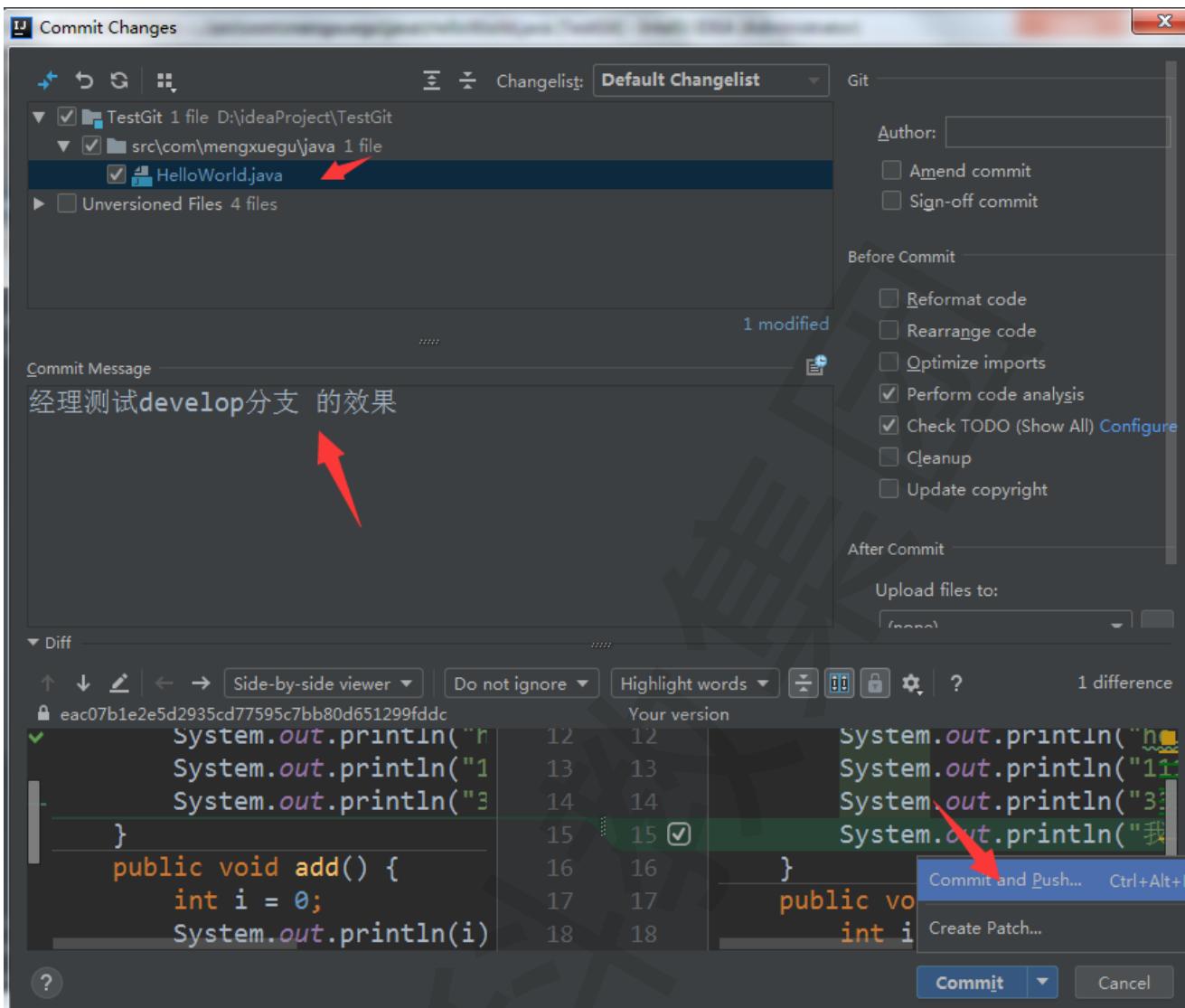
16:6 CRLF UTF-8 Git: develop

11.4.2 第2步：经理 develop 分支上开发新功能

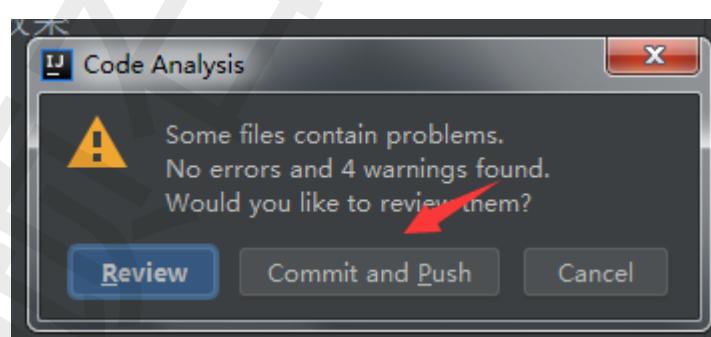
```
7 * @Version: 1.0
8 */
9 public class HelloWorld {
10    public static void main(String[] args) {
11        System.out.println("hello");
12        System.out.println("helloworld 2222222");
13        System.out.println("111111111111111");
14        System.out.println("333333333");
15        System.out.println("我是经理，我在添加新的功能 . . . . ");
16    }
}
```

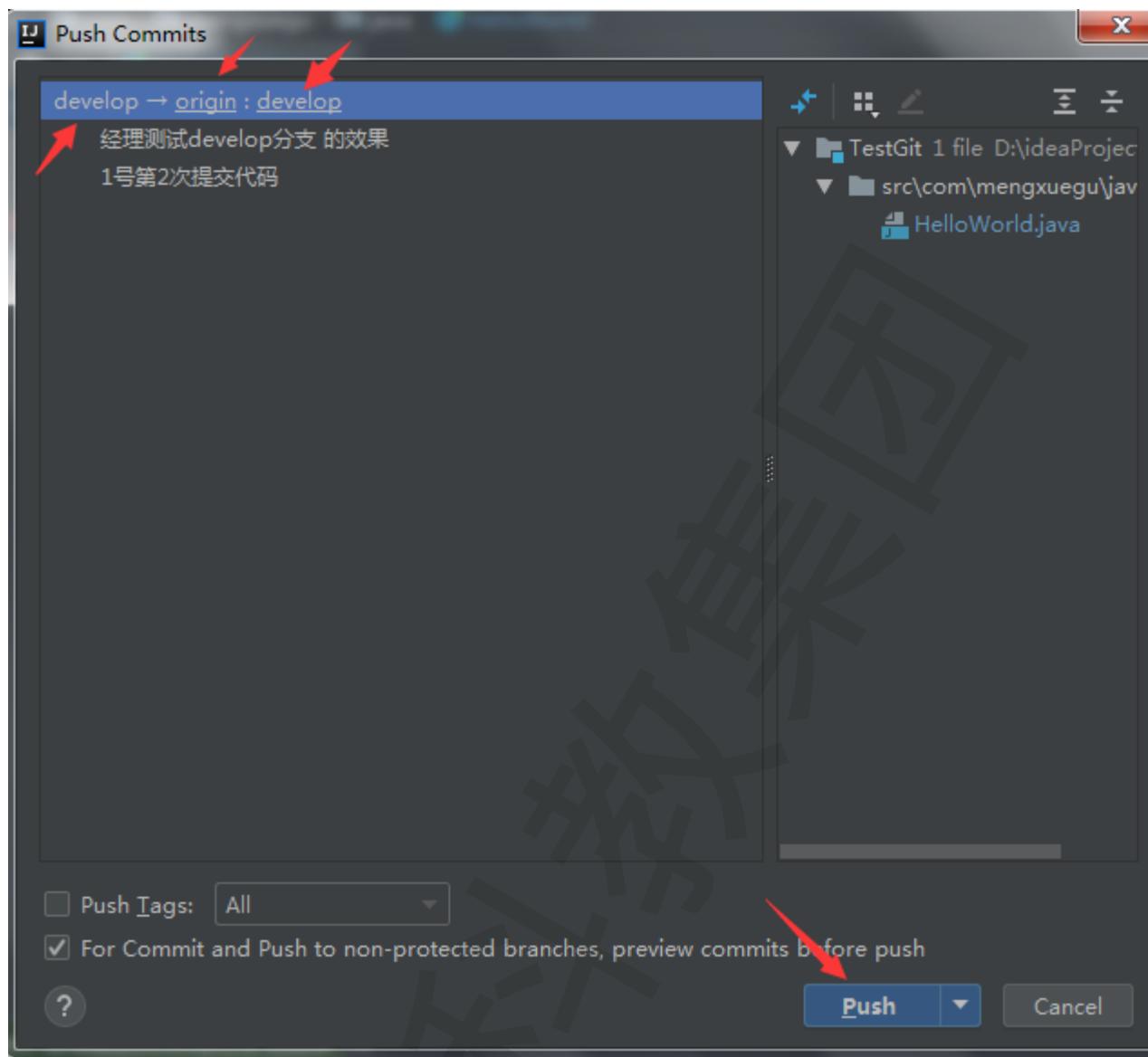
新加的功能

11.4.3 第3步：经理 提交到本地库并推送到远程develop分支



- 上面点击 `commit and Push` 后，会弹出下面窗口，提示进行 `push` 到远端库





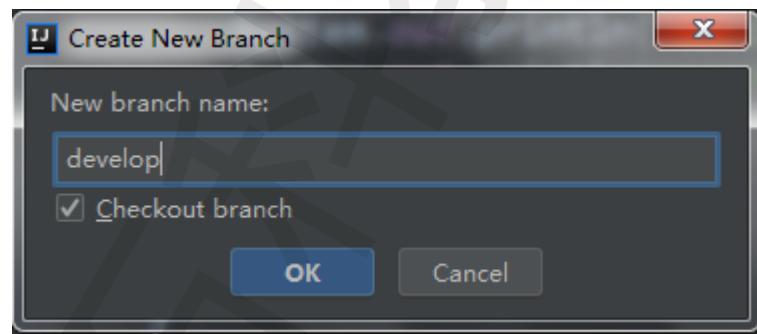
- 在码云上切换develop分支查看

The screenshot shows a Git interface with a 'develop' branch selected. The code editor displays the following Java code:

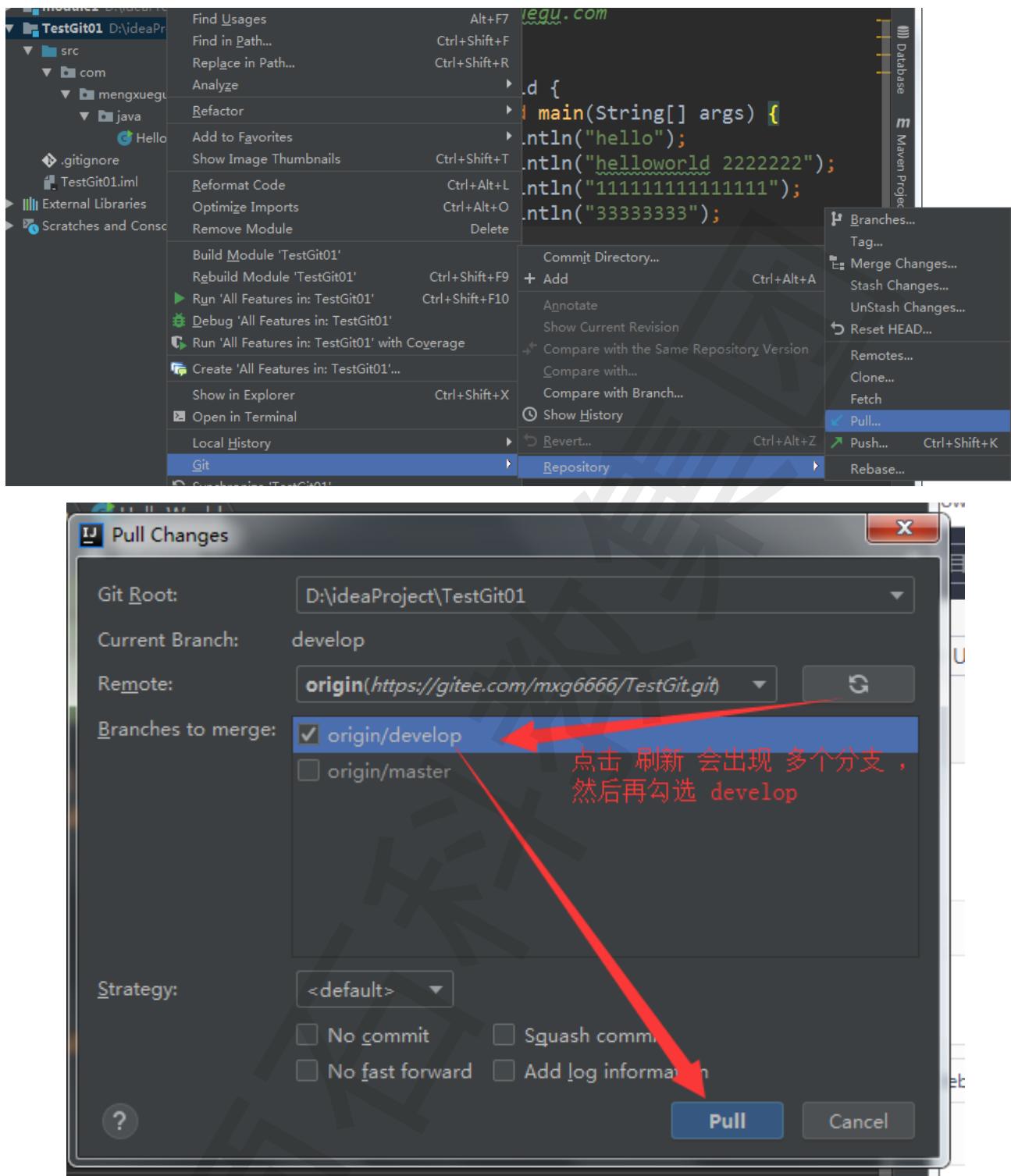
```
6  * @Auther: www.mengxuegu.com
7  * @Version: 1.0
8  */
9 public class HelloWorld {
10    public static void main(String[] args) {
11        System.out.println("hello");
12        System.out.println("helloworld 2222222");
13        System.out.println("1111111111111111");
14        System.out.println("33333333");
15        System.out.println("我是经理，我在添加新的功能 . . . . ");
16    }
17    public void add() {
18        int i = 0;
19        System.out.println(i);
}
```

11.4.4 第4步：总监 拉取远程库develop分支代码到本地develop分支

- 总监先在本地创建好develop分支

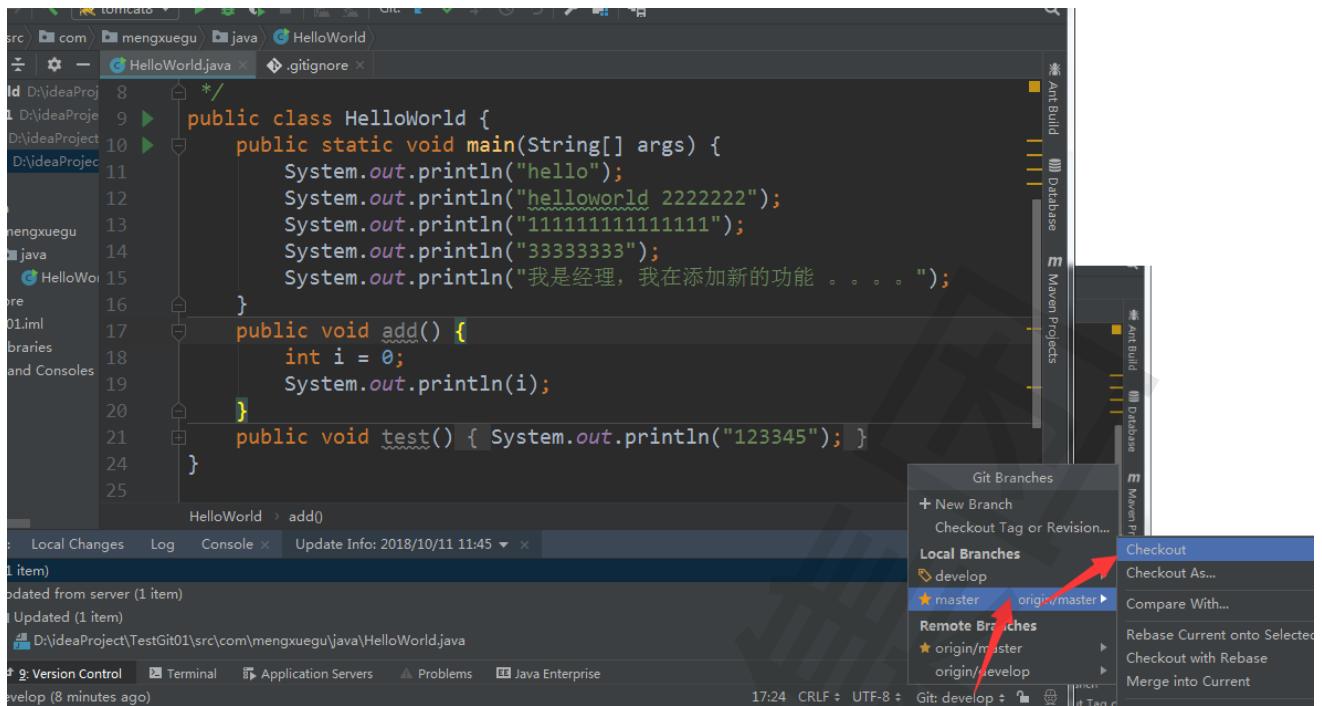


- 总监再进行远程库develop分支代码到本地develop分支

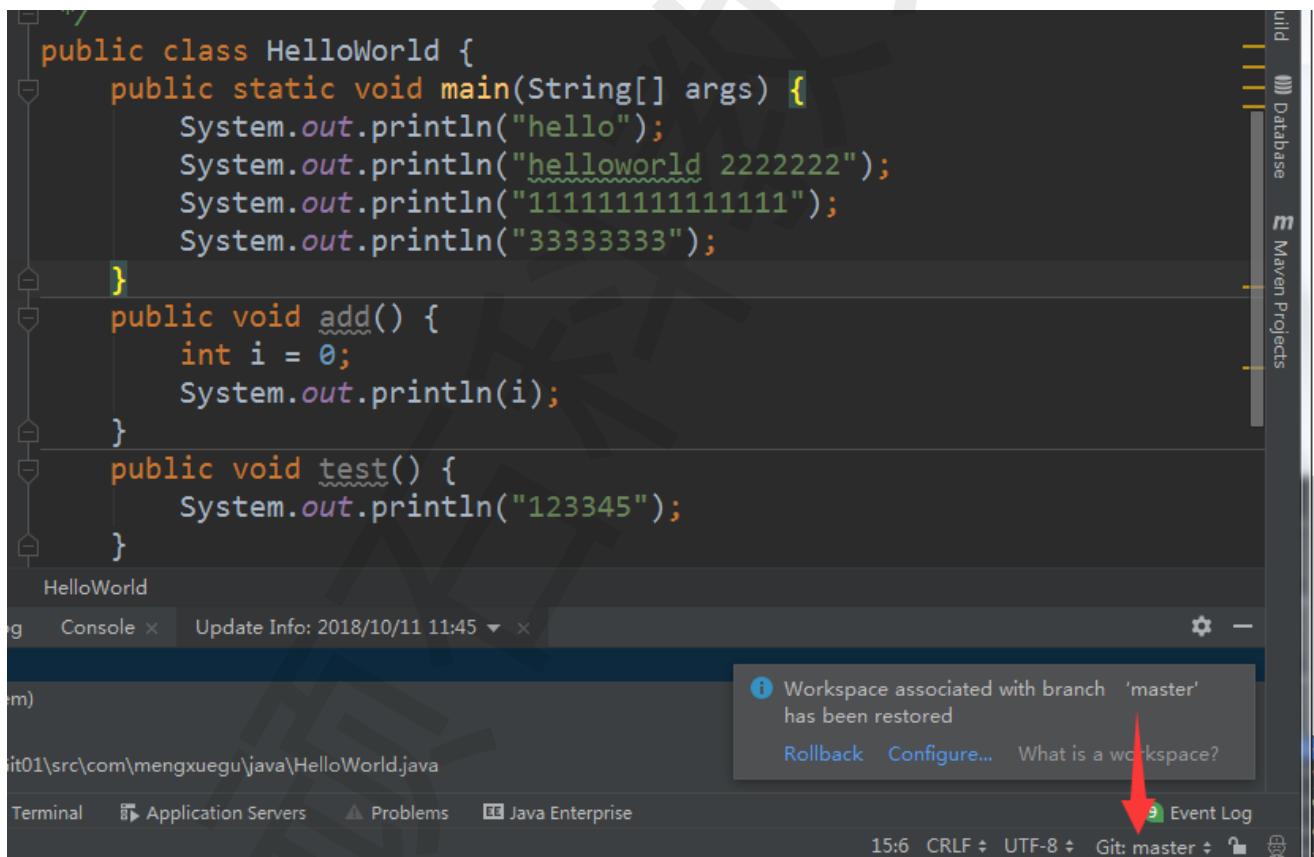


11.4.5 第5步：总监切换回主分支master（本地库）

- 项目右键 → Git → Repository → Branches

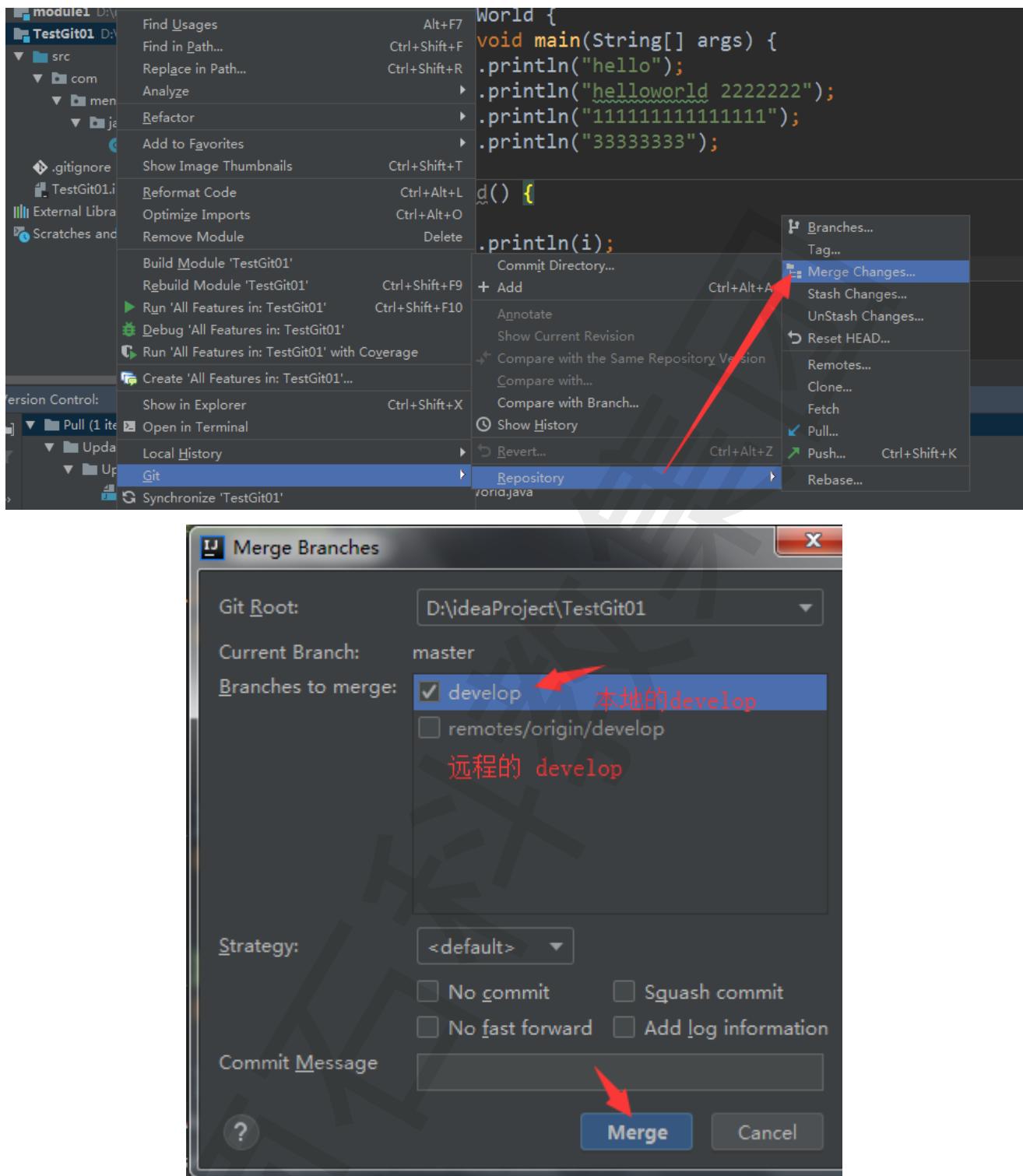


- 如下已经切换成功：



11.4.6 第6步：总监将本地develop合并到mater分支上

- 项目右键 → Git → Repository → Merge Changes...



- 合并结果：

```
8 */  
9 public class HelloWorld {  
10     public static void main(String[] args) {  
11         System.out.println("hello");  
12         System.out.println("helloworld 2222222");  
13         System.out.println("1111111111111111");  
14         System.out.println("33333333");  
15         System.out.println("我是经理，我在添加新的功能 . . . . ");  
16     }  
17     public void add() {  
18         int i = 0;  
19         System.out.println(i);  
20     }  
21     public void test() {  
22         System.out.println("123345");  
23     }  
HelloWorld > main()
```

Version Control: Local Changes Log Console > Update Info: 2018/10/11 11:57 >

Merge (1 item)
Updated from server (1 item)
Updated (1 item)
D:\ideaProject\TestGit01\src\com\mengxuegu\java\HelloWorld.java

Event Log
checked out master (3 minutes ago)

11.4.7 第7步：总监将合并后的master推送到远程库master

- 项目右键 → Git → Repository → push

