# 源代码管理工具

# 使用指南

[源代码管理工具使用指南 1](#_Toc24436)

[1. Git概念 2](#_Toc22367)

[2. Git与SVN的区别 3](#_Toc17376)

[3. 安装配置 4](#_Toc15171)

[3.1. 安装Git 4](#_Toc12627)

[3.2. 安装TortoiseGit 4](#_Toc16811)

[4. TortoiseGit的基本操作 6](#_Toc25467)

[4.1. 基本概念 6](#_Toc17970)

[4.2. 常用TortoiseGit命令 6](#_Toc17289)

[4.3. 创建本地Git版本库（克隆：Clone） 7](#_Toc8511)

[4.4. 切换本地分支（切换/检出：Switch/Checkout） 14](#_Toc29784)

[4.5. 增加新文件/文件夹（添加：Add） 15](#_Toc11790)

[4.6. 提交到本地库（提交：Commit） 16](#_Toc16602)

[4.7. 推送到远程库（推送：Push） 19](#_Toc32226)

[4.8. 修改文件 22](#_Toc1777)

[4.9. 删除文件/文件夹（删除：Delete） 23](#_Toc2556)

[4.10. 文件/文件夹改名（更名：Rename） 25](#_Toc18980)

[4.11. 从远程仓库更新到本地 26](#_Toc516)

[4.11.1. 拉取（Pull） 26](#_Toc13510)

[4.11.2. 获取（Fetch）与合并（Merge） 28](#_Toc17290)

[4.12. 同步 29](#_Toc9042)

[4.13. 处理冲突（Edit conflicts/resolve） 29](#_Toc30370)

[4.13.1. 显示冲突文件 29](#_Toc24939)

[4.13.2. 解决冲突 31](#_Toc31953)

[4.14. 查看历史（Show log） 34](#_Toc12319)

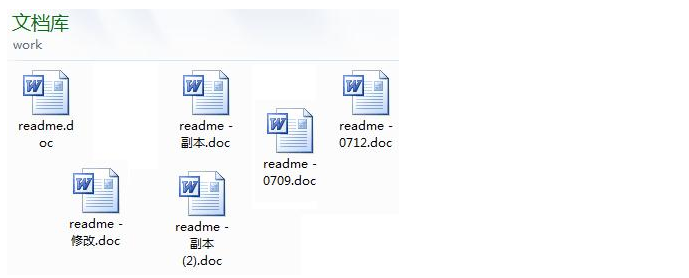
[4.15. 合并到master分支 35](#_Toc31195)

# Git概念

Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理项目代码的版本管理。

什么是版本控制？

*如果你用Microsoft Word写过长篇大论，那你一定有这样的经历：想删除一个段落，又怕将来想恢复找不回来怎么办？有办法，先把当前文件“另存为……”一个新的Word文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的Word文档变成了这样：*



*过了一周，你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了，只好一个一个文件去找，真麻烦。*

*看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。*

*更要命的是，有些部分需要你的财务同事帮助填写，于是你把文件Copy到U盘里给她（也可能通过Email发送一份给她），然后，你继续修改Word文件。一天后，同事再把Word文件传给你，此时，你必须想想，发给她之后到你收到她的文件期间，你作了哪些改动，得把你的改动和她的部分合并，真困难。*

*于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便？*

*这个软件用起来就应该像这个样子，能记录每次文件的改动：*



*这就实现了版本控制。*

怎么理解Git的分布式？

它支持中央代码仓库和个人本地的代码仓库（也就是每个开发人员从中央版本库上check out代码后在自己的机器上克隆一个跟中央版本库一模一样的本地版本库。）

SVN本地没有仓库的概念。

# Git与SVN的区别

1. Git与SVN的最大区别是分布式

Git支持中央代码仓库和个人本地的代码仓库，在未联网时，开发者可以在本地代码仓库上进行提交、回退、分支、合并等所有操作；等可以联网时再推送到中央代码仓库。

1. Git把内容按元数据方式存储，而SVN是按文件

所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。如果你把.git目录的体积大小跟.svn比较，你会发现它们差距很大。因为.git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。

1. Git分支和SVN分支不同
2. Git没有一个全局的版本号，SVN有
3. Git的内容完整性优于SVN

# 安装配置

这里仅介绍Windows环境下使用Git客户端软件的方法。

## 安装Git

* 进入Git官网

[https://git-scm.com/download](https://git-scm.com/download/win)



* 右侧已经列出适合当前电脑的Windows版本，目前最新发布版为v2.18.0
* 下载后按照默认选项安装。安装完成后进入命令行模式，输入命令git –version（注意是两个短杠）。如能看到下列的版本信息，说明Git环境安装成功。
* 如不出现下列内容，尝试配置path环境变量来解决。

## 安装TortoiseGit

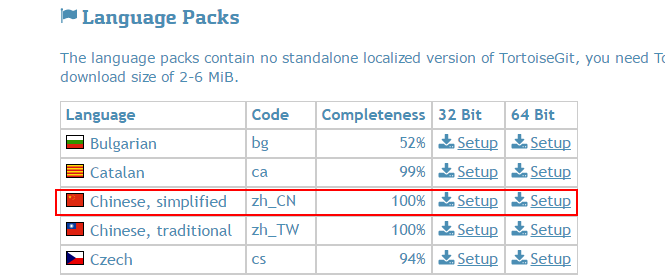
* 进入TortoiseGit官网。

<https://tortoisegit.org/>



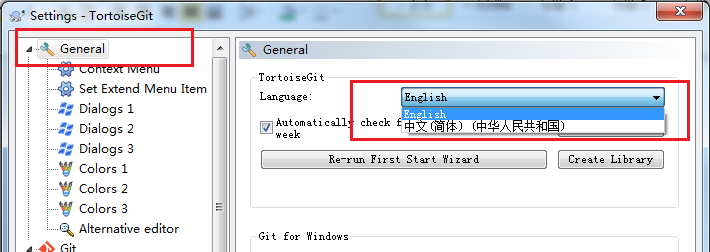
点击“Download”进入下载页面，看到当前最新版本为v2.6.0，根据自己操作系统是32位还是64位，选择下载合适的版本。

* 页面下方可以下载中文语言包。



* 安装TortoiseGit和中文语言包后，需要重新启动计算机。
* **启用中文版：**

【开始菜单/TortoiseGit/Settings】打开设置界面，选择General一栏，把Language切换到简体中文即可。



# **[TortoiseGit](http://www.baidu.com/link?url=PjAhBdglowP4qB7QAe5Rc6qVdzKjsHPR-K5SmgYs20m5MuFZOWjdxyqP3Dra90EV" \t "https://www.baidu.com/_blank)的基本操作**

本章描述了代码仓库的日常操作，包括克隆、拉取代码、修改、提交、解决冲突、合并分支代码等方面。

## 基本概念

Git版本库分为远程仓库（中央仓库）、本地仓库、工作区三个区域。



## 常用TortoiseGit命令

以下为本文涉及到的常用TortoiseGit命令清单。

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **命令** | **英文** | **说明** |
| 1 | 克隆 | Clone | 创建本地仓库，将远程仓库的所有分支获到本地仓库，默认将master分支作为当前工作区 |
| 2 | 拉取 | Pull | 把远程仓库更新到到本地仓库，并自动将当前分支与工作区代码合并 |
| 3 | 获取 | Fetch | 把远程仓库更新到到本地仓库，但不合并进工作区。需再使用合并(Merge)命令才进行合并。 |
| 4 | 推送 | Push | 把本地仓库推送到远程仓库。 |
| 5 | 同步 | Sync | 远程与本地仓库之间数据传递功能向导，允许进行拉取(Pull)、获取(Fetch)、推送(Push) |
| 6 | 提交 | Commit | 把工作区的添加、修改、删除的文件更新到本地仓库。 |
| 7 | 显示日志 | Show log | 显示提交日志。（注意此处是本地仓库的日志，不一定是远程仓库最新的日志。） |
| 8 | 编辑冲突 | Edit conflicts | 自动合并发生冲突后，通过该功能打开TortoiseGitMerge工具，解决冲突。 |
| 9 | 解决冲突 | resolve | 自动合并发生冲突后，把冲突文件标记为已解决，手工打开文件自行解决。 |
| 10 | 更名 | Rename | 修改工作区的文件名。后续需提交(Commit)或还原(Revert)。 |
| 11 | 删除 | Delete | 删除工作区的文件。后续需提交(Commit)或还原(Revert)。 |
| 12 | 还原 | Revert | 添加、修改、删除、改名后在提交(Commit)之前撤销修改。 |
| 13 | 切换/检出 | Switch/Checkout | 切换工作区分支，如果未检出则需要创建本地分支 |
| 14 | 合并 | Merge | 把本地仓库合并到工作区，一般与获取(Fetch)配合使用。 |
| 15 | 添加 | Add | 把工作区新增的未受控文件增加到暂存区。后续需提交(Commit)或还原(Revert)。 |

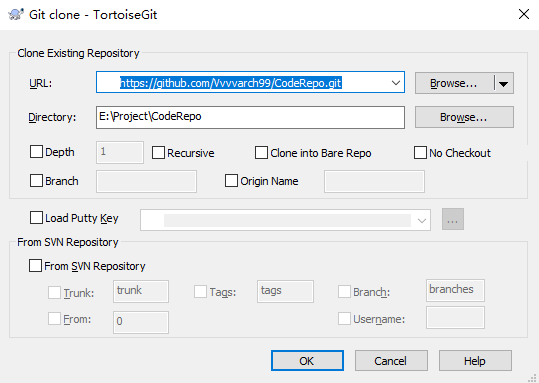
## 创建本地Git版本库（克隆：Clone）

以下以gitHub为中央代码仓库服务器为例。

有两种方式：Http、SSH

* 方式一（Http）：

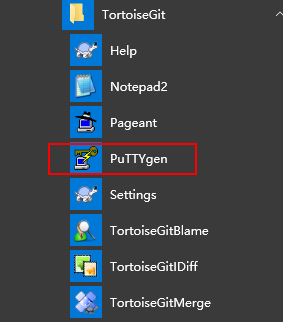
在你想要存放的目录下（如E:\Project）右键选择【Git Clone…】命令，在【URL】中输入代码仓库的URL【URL】中输入本地工作区的代码仓库目录，点击【OK】执行。



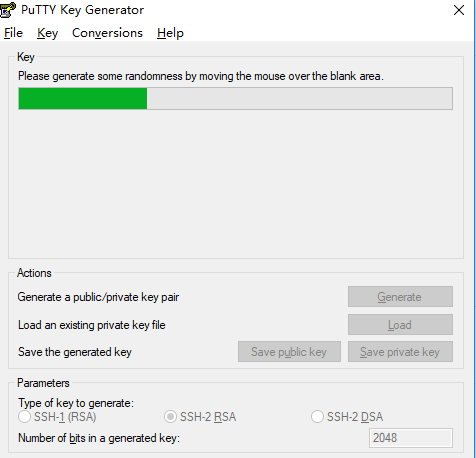
方式二（SSH）：

参考：<https://blog.csdn.net/gulingfengze/article/details/69665223>

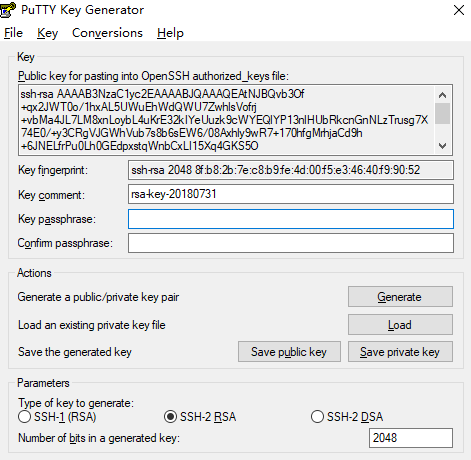
使用TortoiseGit下的PuttyGen生产key



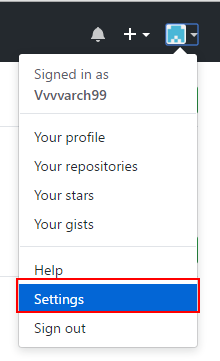
1）点击Generate按钮，会出现绿色进度条，等下生成。

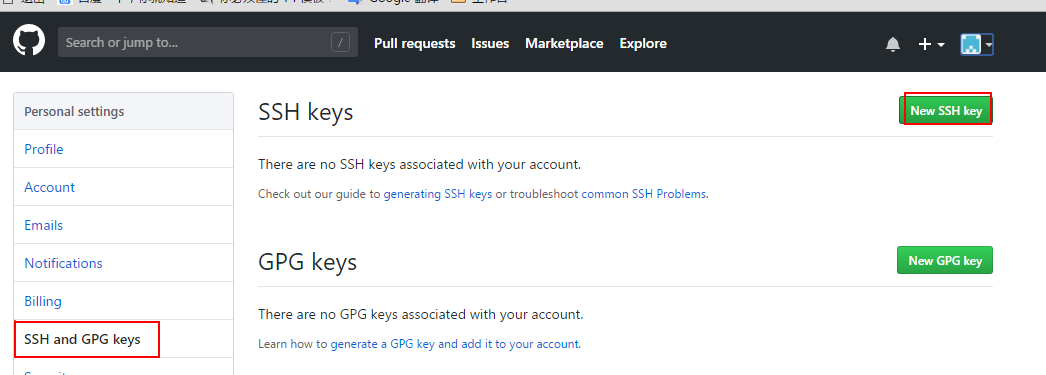


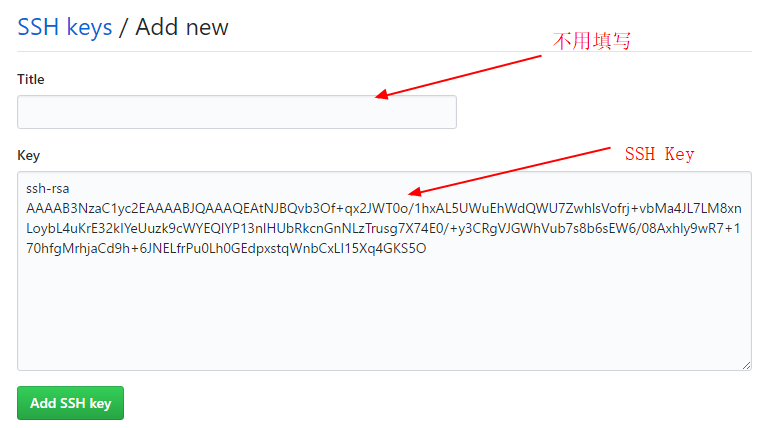
1. 生成之后复制生成的全部内容，窗口先留着不关闭。



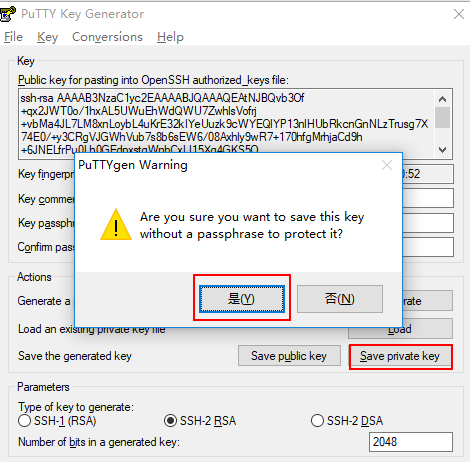
1. 添加SSH Key 到GitHub 上



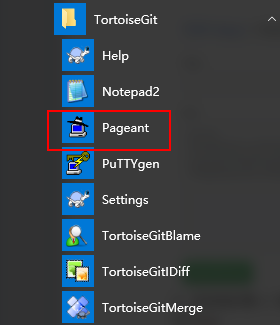


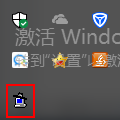


1. 返回到步骤2）的窗口，点击Save private key按钮保存为适用于TortoiseGit的私钥扩展名为.ppk



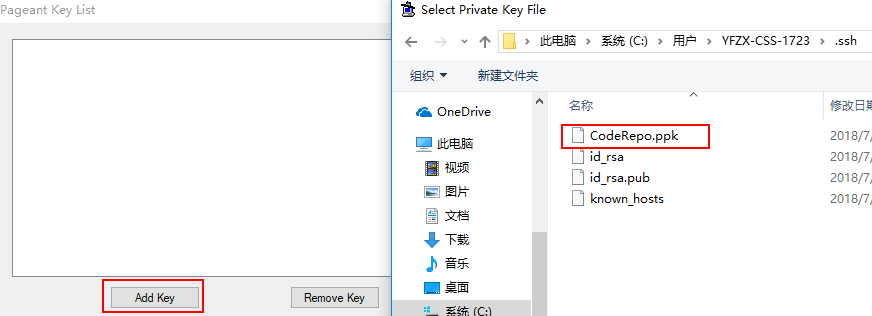
1. 运行TortoiseGit开始菜单中的Pageant程序，程序启动后将自动停靠在任务栏中，双击该图标，弹出key管理列表

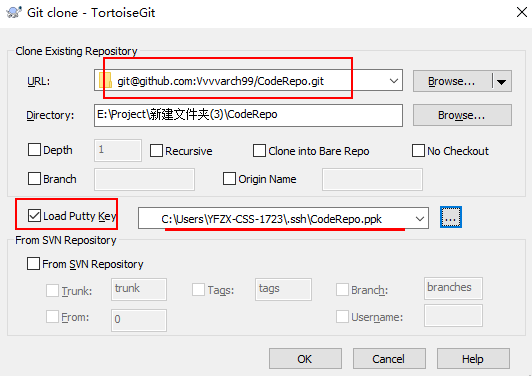




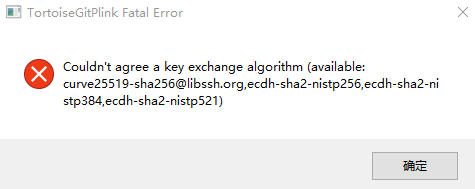
6）在弹出的key管理列表中点击add key,将第4步中保存的私钥（.ppk）文件加进来，关闭对话框即可。

经上述配置后，就可以使用TortoiseGit进行push、pull操作了，不用每次都输入密码了。

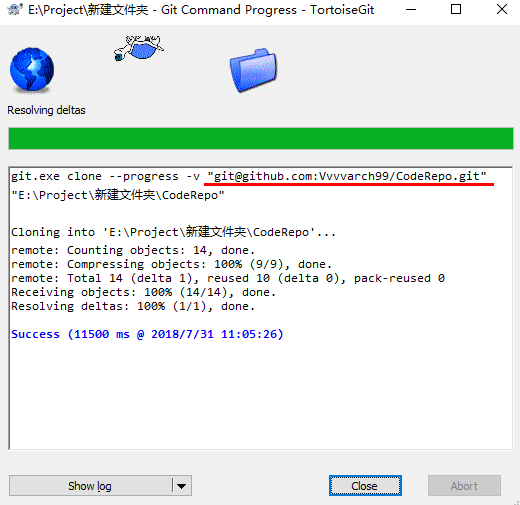




若点OK之后出现异常，如下图，请检查TortoiseGit的版本号，需要升级到2.6.0（当然，与之相匹配的Git也要升级到2.18.0）。



重新clone即可成功。



* 创建好以后，会新建一个CodeRepo的目录，里面会存放本地工作区文件。

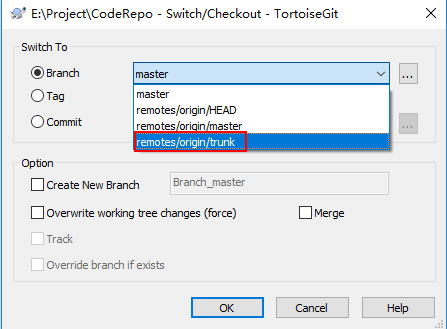
在CodeRepo目录下会创建一个.git目录（即E:\Project\CodeRepo\.git\），里面会存放本地仓库。Clone出的目录以及文件图标会有“绿色勾号”，表示已受到Git控制。



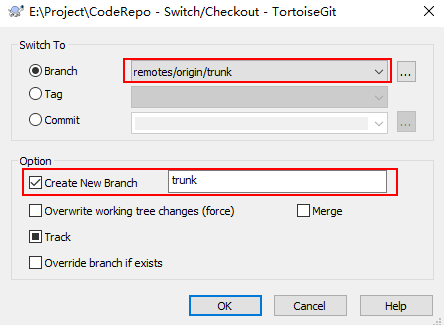
* 克隆本地仓库后，默认会把master分支作为当前工作区。

## 切换本地分支（切换/检出：Switch/Checkout）

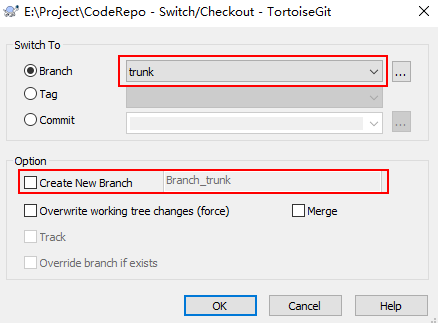
* 通过克隆创建本地仓库后，默认把master分支作为当前工作区。
* 在本地工作区目录下（E:\Project\CodeRepo\），右键选择【TortoiseGit】->【Switch/Checkout…】。
* 【分支】一栏显示master，表示为当前的工作区分支为“master”。下拉选择“remotes/orign/trunk”，准备切换到trunk分支。



* 选择了“remotes/orign/trunk”以后，由于之前本地库并未检出trunk分支，下面选项将自动勾选【Create New Branch】选项，且自动填上了trunk。不要修改此处设定，直接点【确定】即可。



* 本地trunk分支创建后，以后如果再此切换到trunk分支，则不会自动勾选【Create New Branch】选项。也不要手工去勾选，并输入trunk，否则会创建同名本地分支。



## 增加新文件/文件夹（添加：Add）

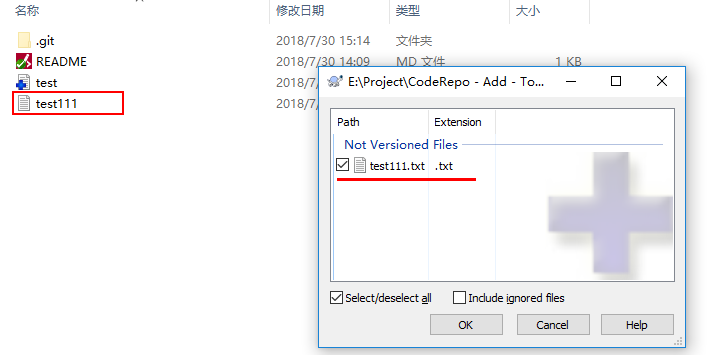
* 在工作区增加了新文件或者文件夹，准备提交。流程如下：



* 方法一：选择文件或文件夹，右键【TortoiseGit】->【Add…】即可，之后文件图标上会显示“蓝色加号”。

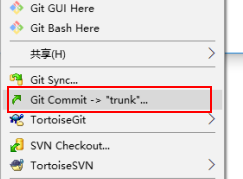


* 方法二：在工作区文件夹空白处，右键【TortoiseGit】->【Add..】，会列出文件夹下所有未添加的文件，勾选好后点【OK】，一次性全部添加进去。

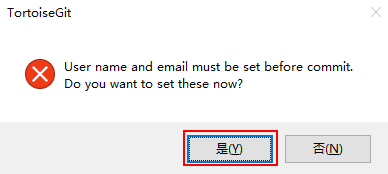


## 提交到本地库（提交：Commit）

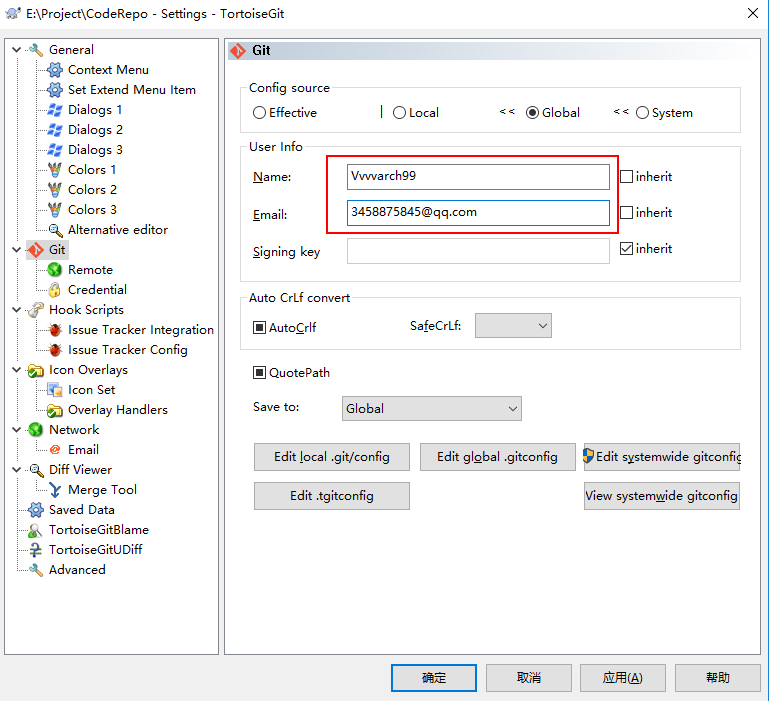
* 已增加到暂存区的文件/文件夹，需要提交到本地库。在工作区目录下右键【GitCommit ->”trunk”…】



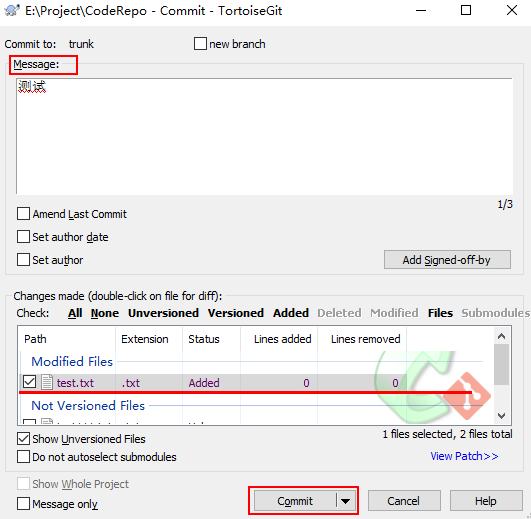
* 如果是第一次使用TortoiseGit，则会出现如下提示。需要去配置git中央代码库的用户名和邮箱（此处为github的配置信息）



* 在【Git】->【Name】【Email】中配置好信息之后，点击【确定】，即可进行提交。

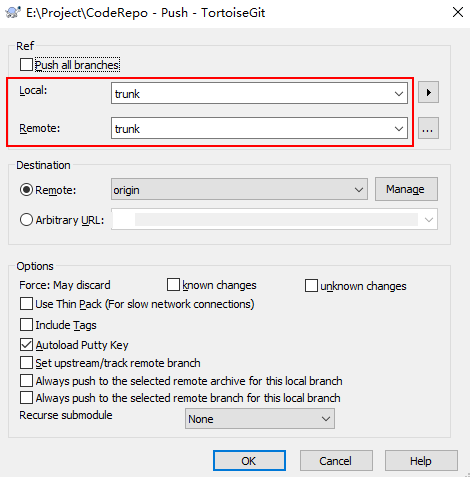


* 在【Message】中必须填写本地提交的日志说明，然后下面的提交列表中勾选，最后点击【Commit】即可。
* 右键菜单中箭头后面的“trunk”字样，表示当前工作区对应的是哪一个分支（本示例中为trunk分支），顶部的【Commit to】也表明了将提交到本地哪一个分支。
* **同时注意不要勾选顶部的【新建分支】选项。**



## 推送到远程库（推送：Push）

* 在工作区文件夹空白处，右键【TortoiseGit】->【push…】，默认会推送当前的trunk分支。也可以更换推送其他分支。
* 注意不要勾选【Push all branches】，否则可能会把本地新建的分支推送到远程仓库中。

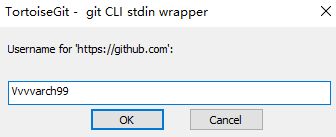


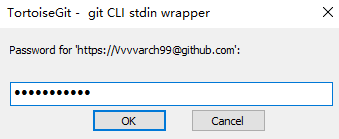
* 注意：

若使用的是“http”的方式，则需要配置，否则每次push都需要输入账号密码；

若使用的是“SSH”，则不会出现这种情况。

具体采用哪种方式，以及是否配置成global Username Password，还应根据实际情况来决定。



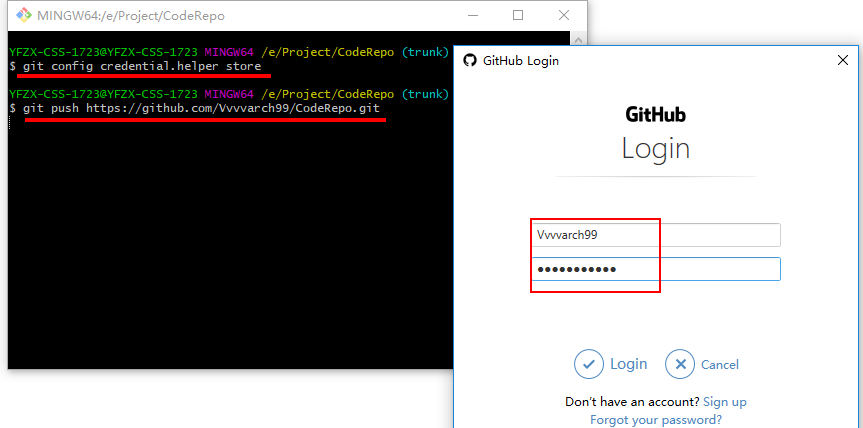


* 注意，推送仅仅是把已提交到本地库的代码推送到远程库。如果有代码仅仅是添加或修改，还未提交到本地库，则这部分代码不会推送到远程库。
* 推送过程可能会出现版本冲突，请参考“4.13处理冲突”小节处理冲突。

拓展：

以“http”的push方式为例，设置用户名、密码（即不需要每次push都输入）。

1. 打卡Git Bash，定位到项目的工作路径（这里是E:\Project\CodeRepo）
2. 执行命令$ git config credential.helper store
3. 执行命令$ git push <https://github.com/Vvvvarch99/CodeRepo.git> （项目的url）
4. 这时会弹出GitHub的登录页面，需要登录进去



5）输入用户名和密码。



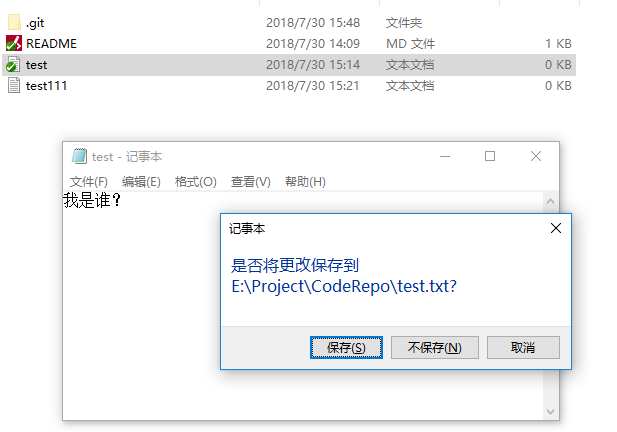
出现上图，即设置成功。

1. 执行命令，还可以设置它的缓存时间

git config --global credential.helper 'cache --timeout 7200'

## 修改文件

* 使用正常方式修改文件，修改过的文件，图标会变成“红色感叹号”，表示该文件的工作区版本与本地库版本不一致。后续需要提交或还原。





* 流程如下：



* 如果修改的文件不要提交，需要撤销为原版本。可以右键【TortoiseGit】->【Revert】即可。
* **注意，还原以后，工作区修改的文件会被本地库的文件替换掉。需要保留的文件请先备份。**

## 删除文件/文件夹（删除：Delete）

* **注意：要删除Git库中的文件或文件夹，不能在本机中直接删除（即通过操作系统自带的删除功能或Delete键）。必须使用TortoiseGit下的删除命令。**

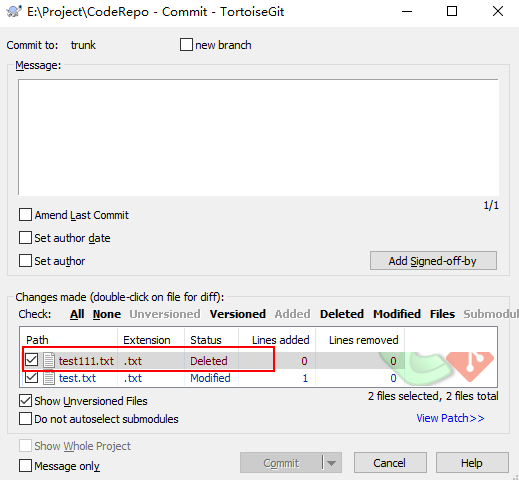
下图红框中的删除为系统自带的删除，不能使用。应当使用绿框中的【TortoiseGit】->【Delete】



* 流程如下：



* 在工作区选择文件或文件夹，【TortoiseGit】->【Delete】，选择的文件或文件夹会消失。
* 在上级目录或工作区根目录中进行提交操作。即可更新到本地库。



* 如果删除操作后未提交之前，可以通过右键【TortoiseGit】->【Revert…】,恢复文件。

## 文件/文件夹改名（更名：Rename）

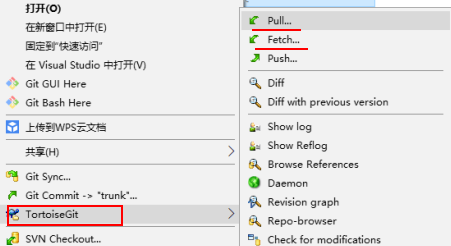
* 选择文件后，通过右键【TortoiseGit】->【Rename…】，可以给文件更名。
* 流程如下：



* 文件更名后，也需要提交到本地库。

## 从远程仓库更新到本地

* 远程仓库代码更新后，需要从把远程仓库代码同步更新到本地仓库中。可通过拉取（pull）、获取（fetch）两种方式来完成。



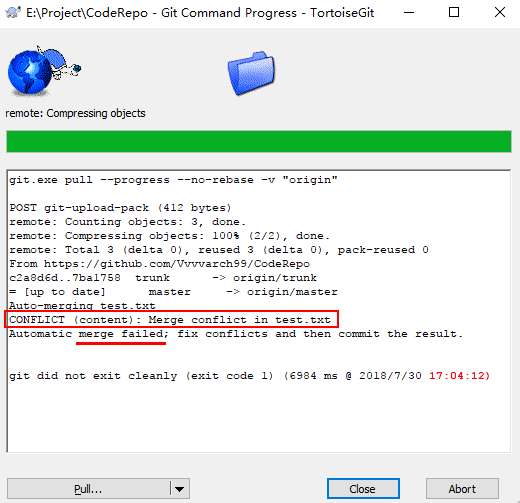
* Pull：拉取并合并。
* Fetch：仅获取，不进行合并。

### 拉取（Pull）

* 工作区根目录中右键，选择【TortoiseGit】->【pull…】。不要修改任何选项。这样操作是把远程仓库的分支里拉取到本地分支，并自动合并到当前工作区。（可通过右键的【Git Commit>xxx】菜单，查看工作区当前为哪个分支。

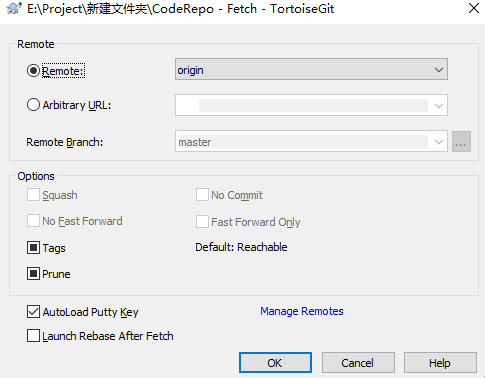
## 

* **注意不要修改远端分支选项。如果选择一个本地不存在或未检出的远端分支，则会导致代码混乱。**
* 如果要改用另一个分支，应当先pull，然后再Switch/Checkout。
* 如果远程分支代码与本地分支代码不一致，则会自动尝试合并。此时更新界面会显示以下提示信息。表示版本冲突自动合并失败。如果出现冲突，请参考“4.13处理冲突”小节处理冲突。



### 获取（Fetch）与合并（Merge）

* 工作区根目录中右键，选择【TortoiseGit】->【Fetch…】。不要修改任何选项。这样操作是把远程仓库的分支里拉取到本地分支，但不合并到当前工作区。



* 获取后需要使用【TortoiseGit】->【Merge…】命令，才能最终显示在工作区。
* 更新所有远程仓库分支，合并过程中可能会出现冲突，请参考“4.13处理冲突”小节处理冲突。

## 同步

* 同步功能包含了拉取(Pull)、获取(Fetch)、推送(Push)等多个同步功能，方便操作。其执行过程与原命令一样。

## 处理冲突（Edit conflicts/resolve）

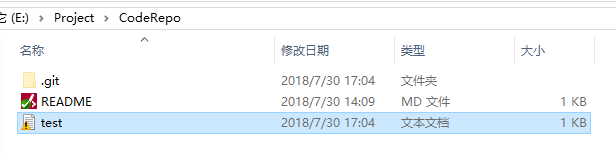
* 当在做远程仓库与本地仓库交换数据过程中，例如推送(Push)、拉取(Pull)、获取(Fetch)过程中，或者在合并(Merge)过程中，都可能产生版本冲突。

### 显示冲突文件

* 提示冲突后，工作区不一定会立刻显示冲突的文件。此时应当按照以下方式处理来显示冲突文件：



* 首先检查当前是否已出现冲突的文件。冲突文件显示为“黄色惊叹号”。如果已经出现，则转入冲突文件处理过程。



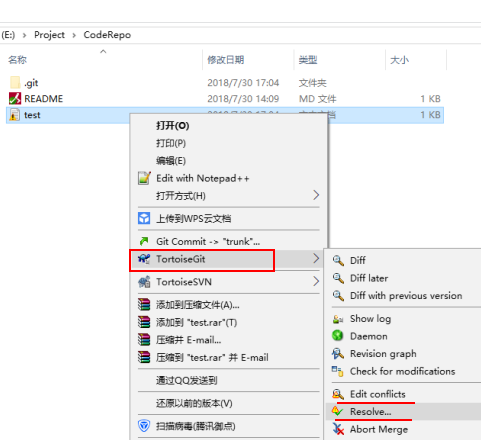
* 如未显示冲突文件，在工作区检查是否有未提交的内容，先提交(Commit)或还原(Revert)。
* 接下来尝试拉取(Pull)，如果拉取后出现冲突文件，则转入冲突文件处理过程。
* 如果仍然未显示冲突文件，则进行合并(Merge)，转入冲突文件处理过程。

### 解决冲突

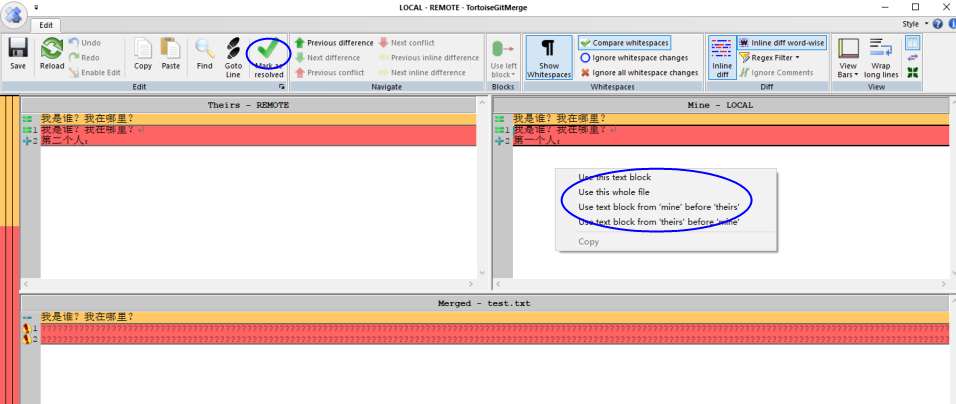
* 等冲突文件显示出来后，可以用两种方式处理。

# 

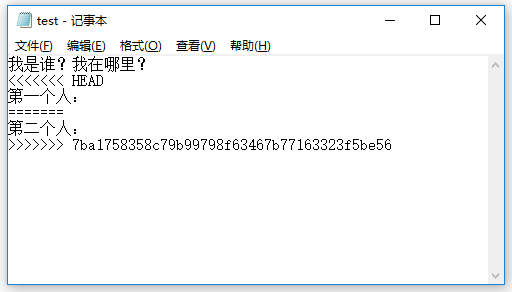
如下图：



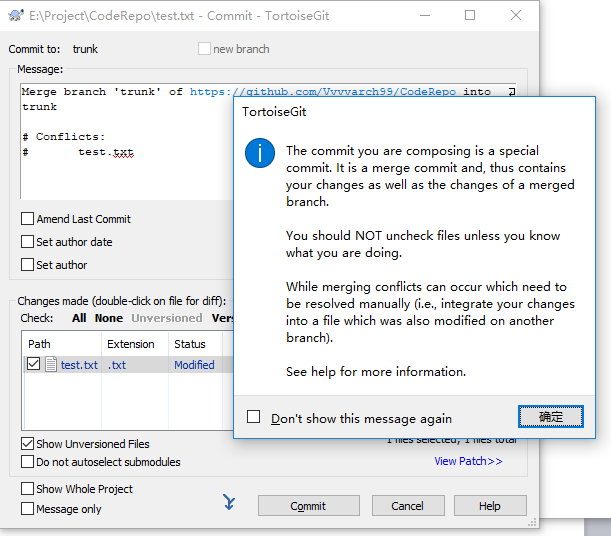
* 处理方式一，Edit conflicts：选择冲突文件，右键【TortoiseGit】->【Edit conflicts…】，将打开TortoiseGit自带的三方冲突合并工具(TortoiseGitMerge)，界面显示如下：
* 在冲突位置右键，决定如何合并成一份文本。全部处理完成后，点击工具栏中的【Mark as resolved】



* 处理方式二：resolve：选择冲突文件，右键【TortoiseGit】->【resolve…】，将把该文件标记为已解决。但是此时文件内容为合并后的内容，还有合并的特殊标签，类似下图。必须要手工编辑解决。

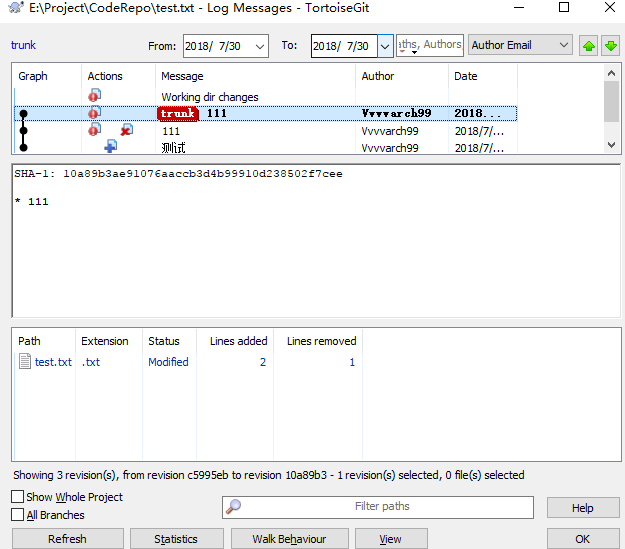


* **解决冲突文件后，一定要进行一次Commit并Push到远程仓库上。即使冲突文件显示为正常的“绿色打勾”标记，也一定要提交并推送一次。这是一个特殊提交：合并提交，在日志中显示也与一般提交不同。**



## 查看历史（Show log）

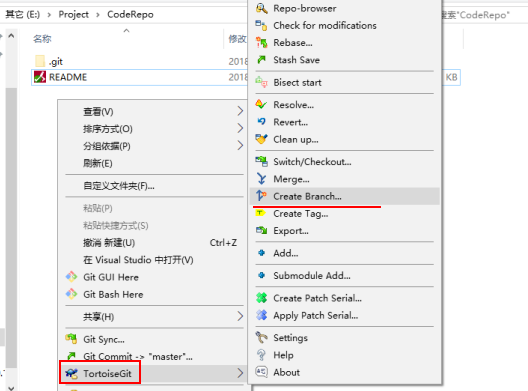
* 在文件或文件夹上右键，在右键菜单中选择【TortoiseGit】->【Show lo…】命令，即可查看更新文件或文件夹的更新历史。如下图:

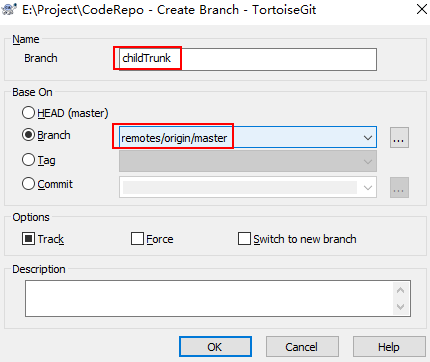


## 合并到master分支

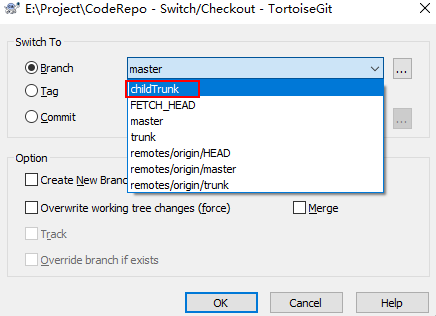
* master分支受保护，不能直接提交。因此需要把开发主分支trunk定期合并到master上。

1. 本地创建一个新的分支，命名为childTrunk

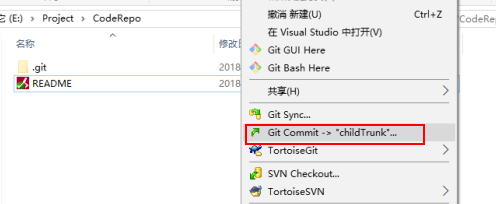




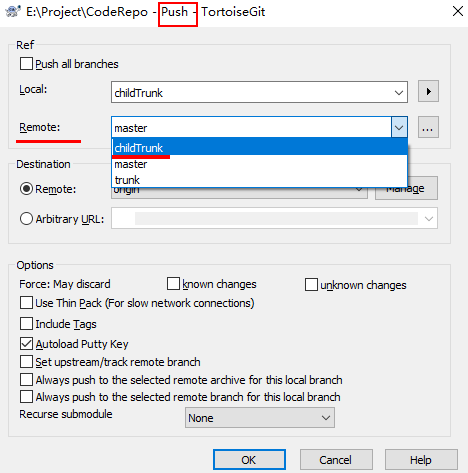
2）切换到新建的分支上【TortoiseGit】->【Switch/Checkout…】命令，选择childTrunk分支。



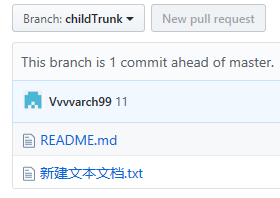
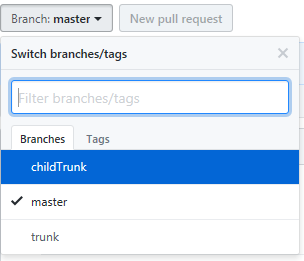
1. 确认已切换到childTrunk分支，然后在此分支上新建一个文本文件。



1. 提交并push到childTrunk分支上

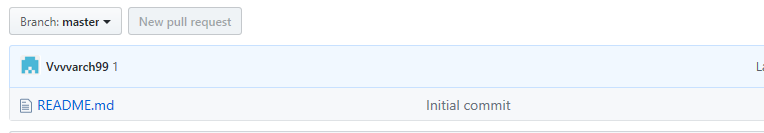


1. 此刻，GitHub上便有了childTrunk分支，以及该分支下的文本文件

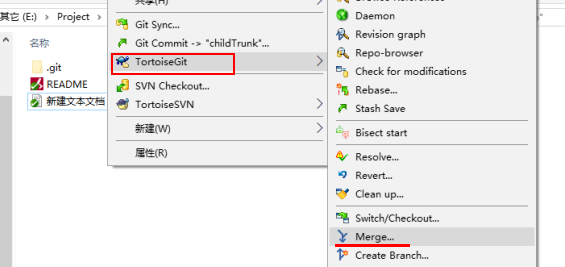


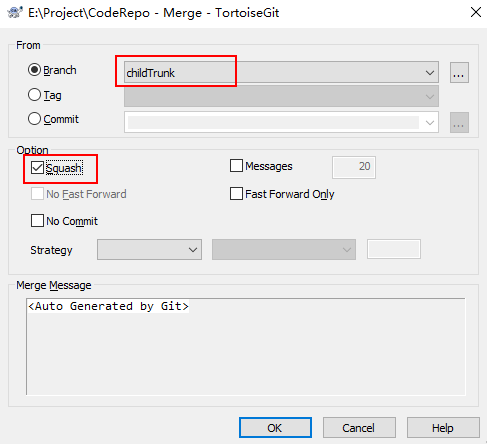
1. 合并：（将childTrunk分支合并到master上）

当前master上仅有一个md的文件



a: 切换本地分支到master上，然后【TortoiseGit】->【Merge…】打开合并窗口，选择childTrunk分支，勾选Squash，点击OK





注意：此时已经把自己的分支代码合并到master分支了，这个时候如果有冲突，请参考“4.13处理冲突”小节处理冲突，若没冲突测试好自己的功能之后，进行Commit、Push到master，到此已经完成了分支的合并功能。

现在可以看到master上已经包含了childTrunk分支的内容。

