# Git的基础

关于学习Git这个主题，有各种同学多次要求写一写，而且要求文艺的写、抒情的写、充满社会主义特色的写，要做到：问题看起来巨复杂，读起来巨简单，学起来巨容易！看把你们惯成什么样子了，好吧不管他，我们接下来就聊一聊Git。

## 安装Git

所需要的软件：Git for Windows、TortoiseGit。

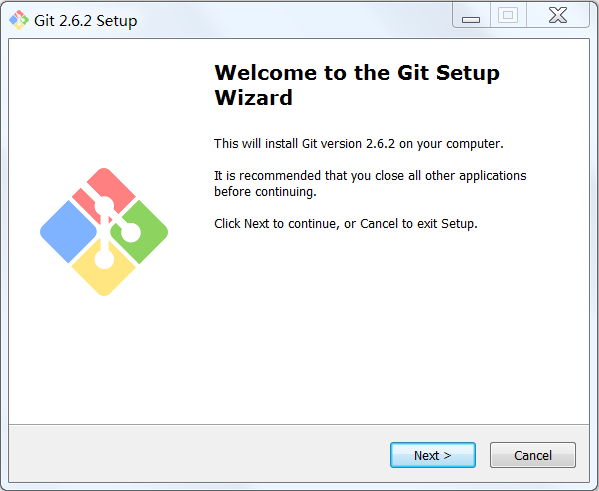
### 安装Git for Windows

要使用TortoiseGit，首先需要安装Git for Windows。因为TortoiseGit其实只是一个壳，它需要调用Git命令行才能发挥作用。请从以下地址下载Git for Windows并安装：

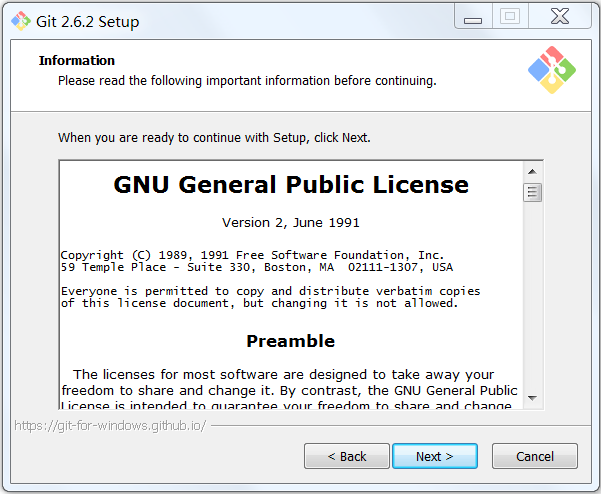
https://git-for-windows.github.io/



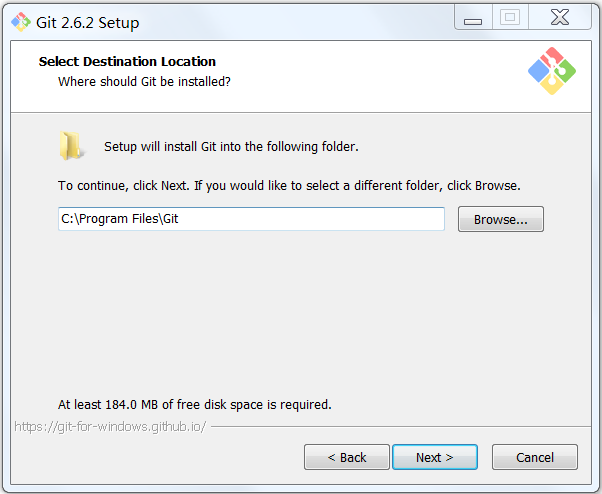
双击已下载的安装程序文件，然后在下一个画面点击Next。



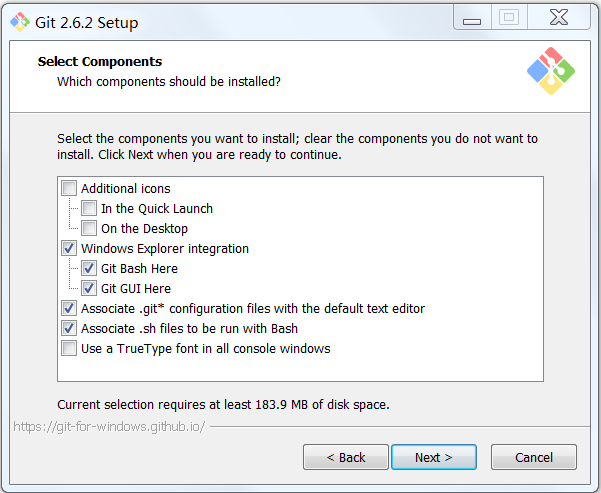
再点击Next。



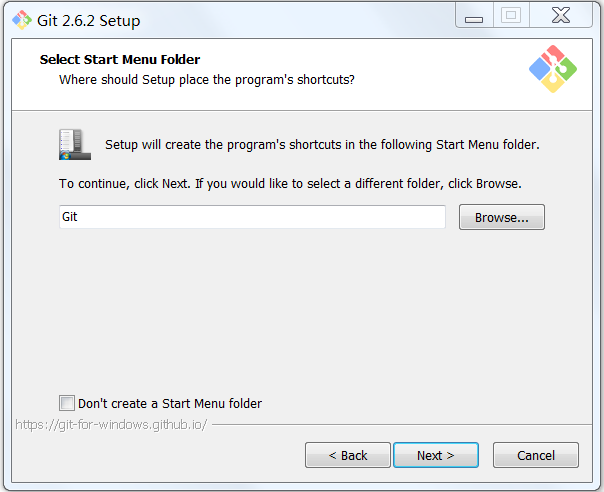
选择安装路径，然后点击Next，这里选择默认安装路径C:\Program Files\Git。



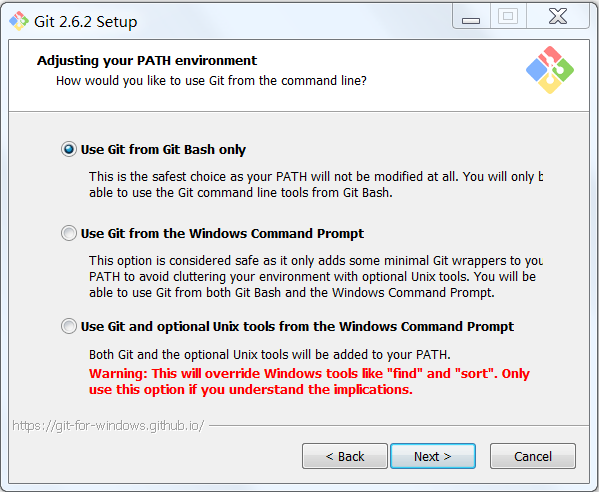
继续点击Next。



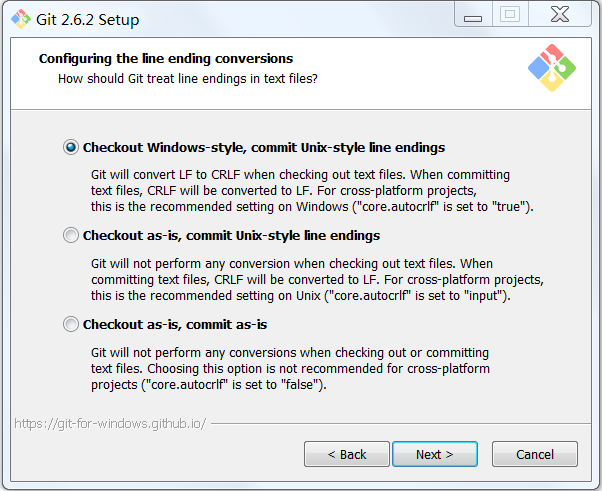
继续点击Next。



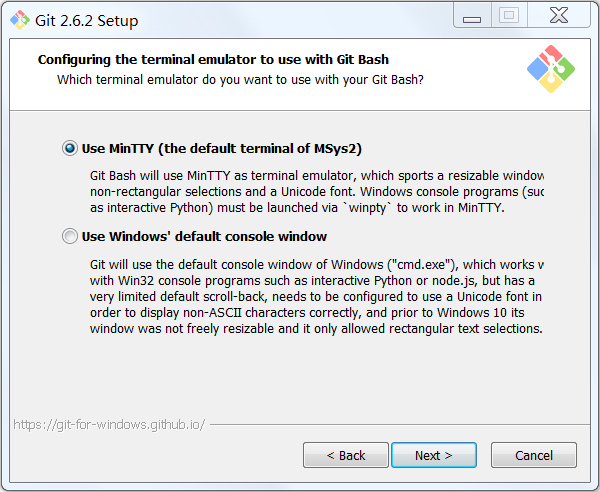
选择Use Git from Git Bash only，然后点击Next。



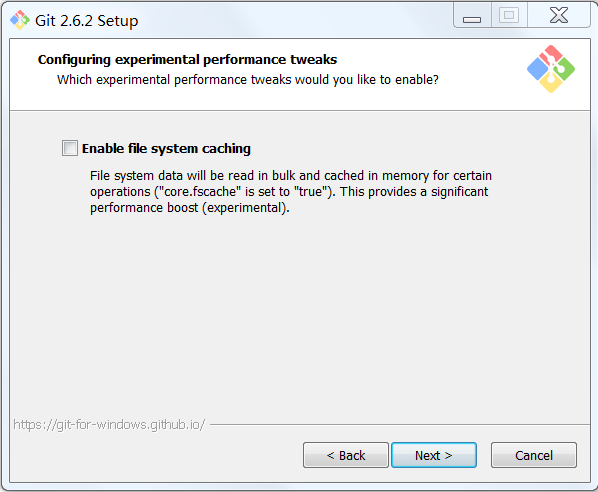
保持默认选择Checkout Windows-style, commit Unix-style line endings，然后点击Next。



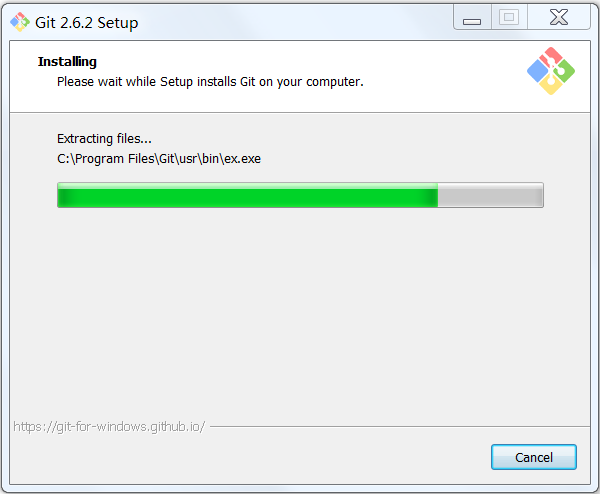
保持默认选择Use MinTTY，然后点击Next。



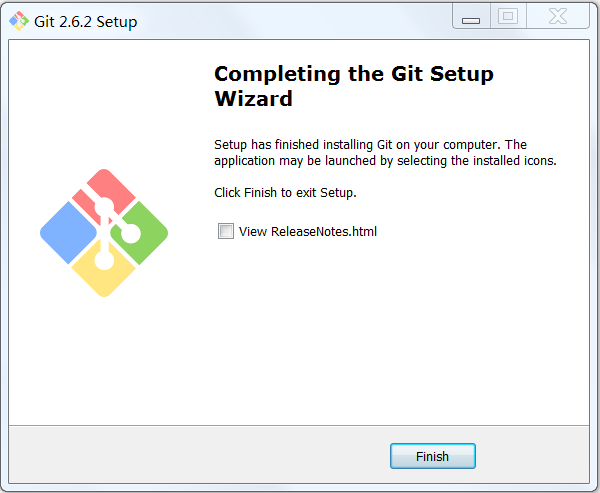
继续点击Next。



安装开始。



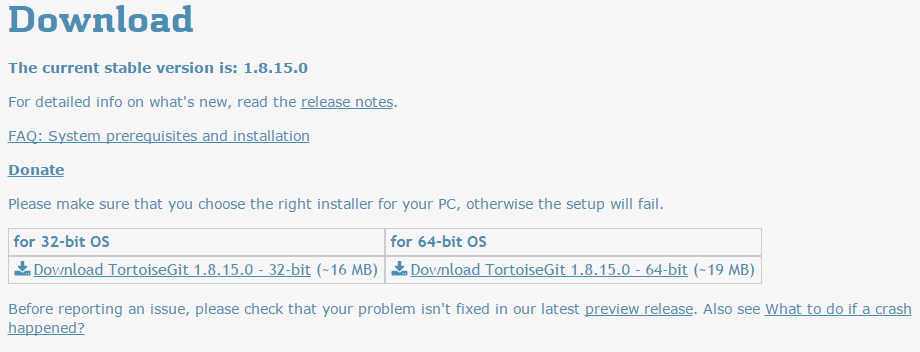
安装完毕，请点击Finish以退出。



### 安装TortoiseGit

从下面的网站下载TortoiseGit的安装程序。请选择与你的Windows相匹配的32bit或64bit的版本下载。

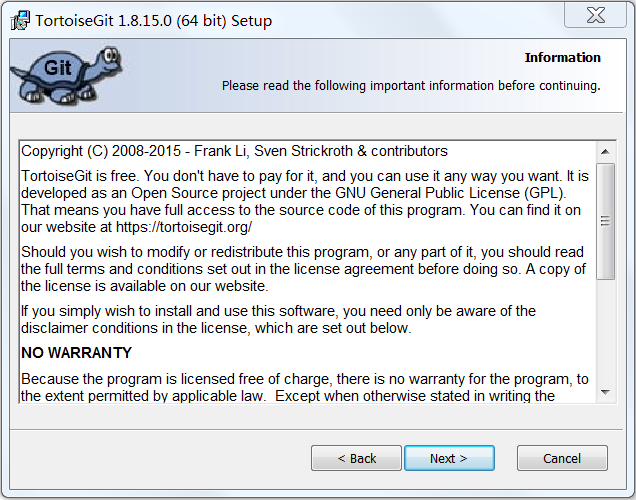
https://tortoisegit.org/download/（需要翻墙）



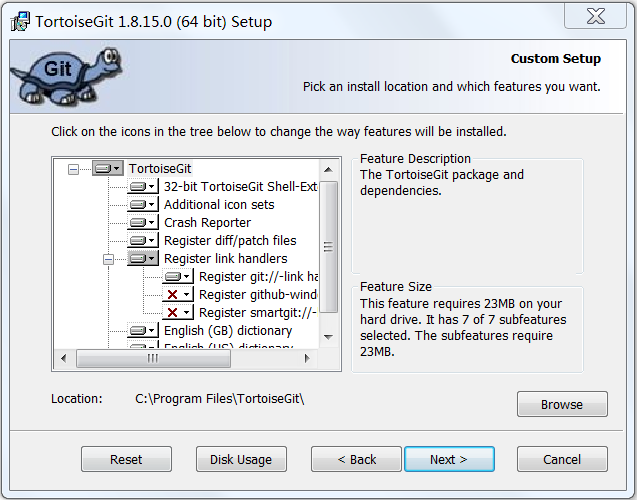
双击已下载的安装程序文件，然后在下一个画面点击Next。



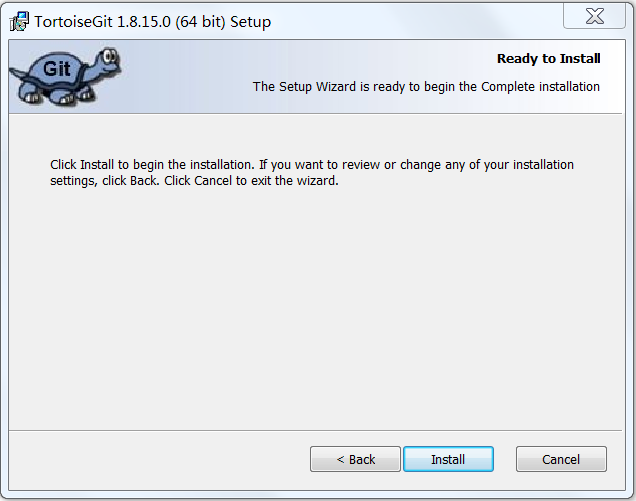
再点击Next。



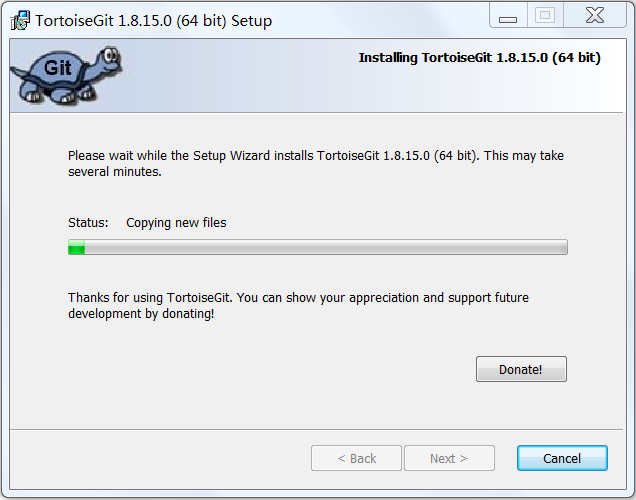
选择安装路径，然后点击Next，这里选择默认安装路径C:\Program Files\TortoiseGit\。



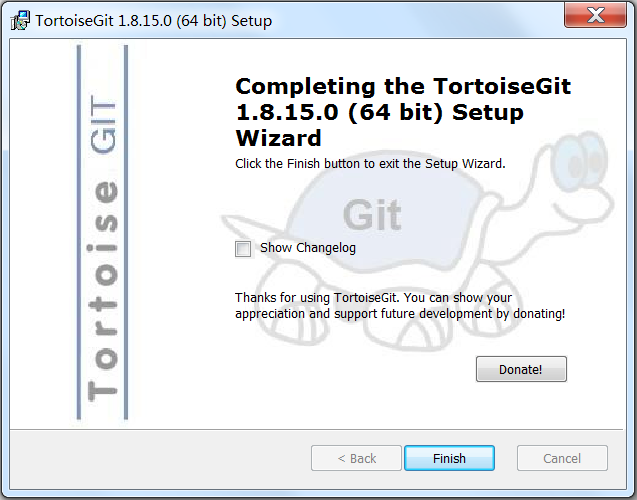
点击Install。



安装开始。



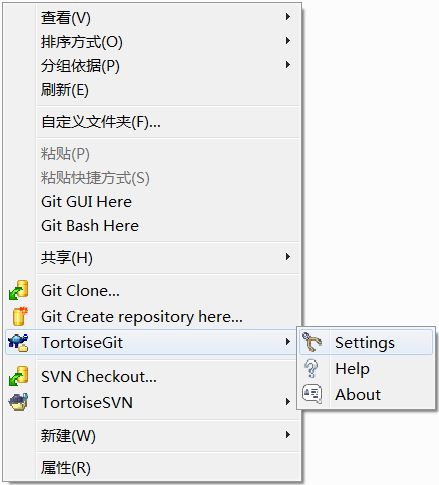
安装完毕，请点击Finish以退出。



## 初期设定

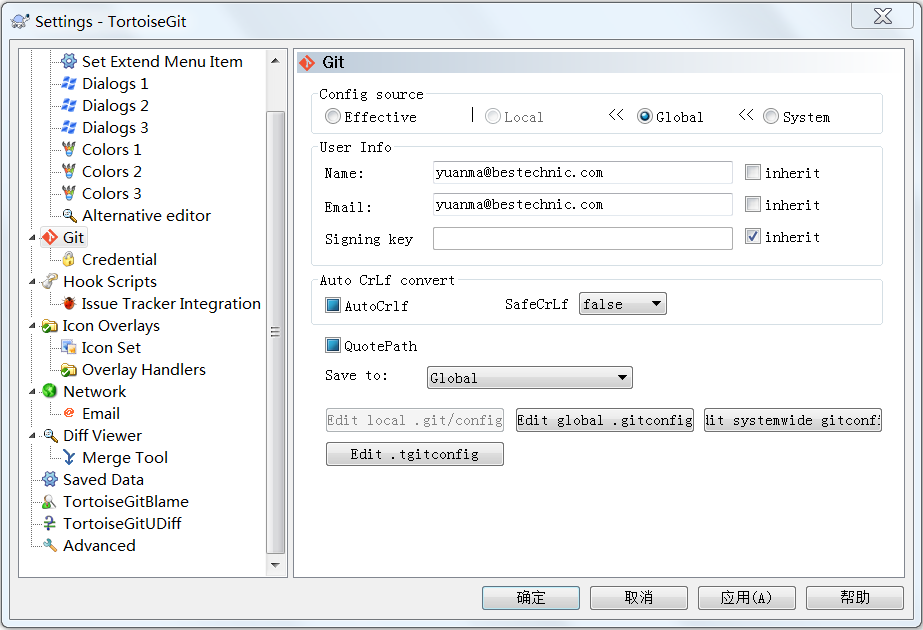
安装Git之后，请输入你的用户名和电子邮件地址。该设置在安装Git后进行一次就够了。这些信息将作为提交者信息显示在更新历史中。

在任意目录空白处单击右键选择TortoiseGit -> Settings。



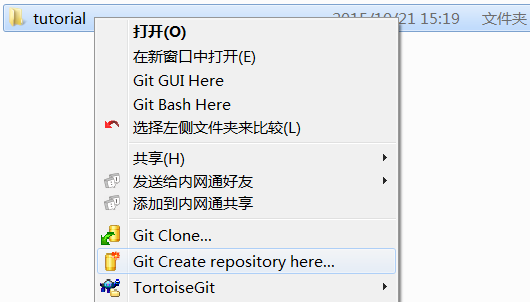
将显示Settings的界面。在左侧列表中选择Git，在右侧Git选项卡中，选择Global选项，并输入你的Name和Email，然后点击应用。这里我将Name和Email都设置成自己的邮箱地址yuanma@bestechnic.com。

***向管理员申请一个Git权限。同时，管理员会帮你生成一个公钥和私钥，公钥保存在Git服务器，私钥会发给你，需要你添加到TortoiseGit中（后续会讲如何添加私钥）。这里，公钥和私钥的格式都是OpenSSH格式。***

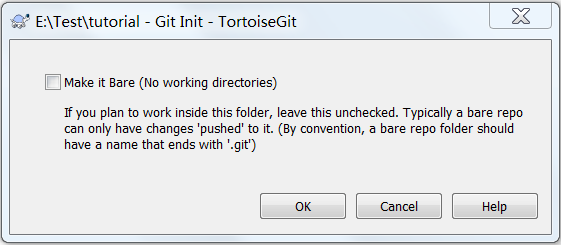


## 新建本地仓库

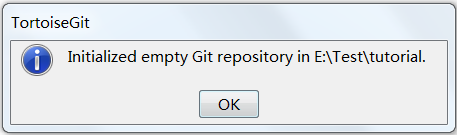
这里我们在E盘根目录新建Test文件夹，进入Test，再创建tutorial文件夹。若要把tutorial目录放在Git的管理之下，单击右键选择Git Create repository here...。



接着会显示以下画面。不要勾选“Make it Bare(No working directories)”，直接点击“OK”。



若本地仓库创建成功，将显示以下画面。请点击“OK”以关闭窗口。



## 提交文件到本地仓库

在tutorial目录新建一个文件，然后将文件添加到本地仓库。

首先在tutorial目录里新建一个名为sample.c的文件，请在文件中输入以下内容：

int main(int argc, char \*argv[]) {

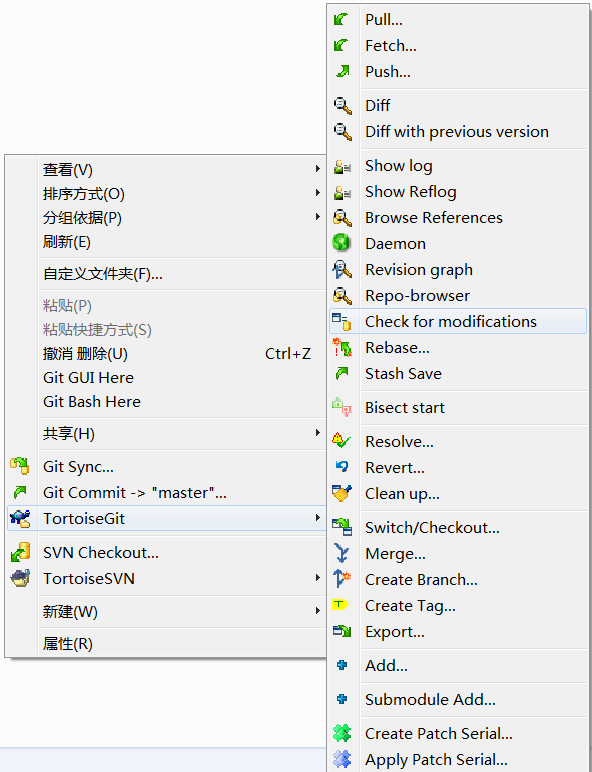
int a;

a = 1;

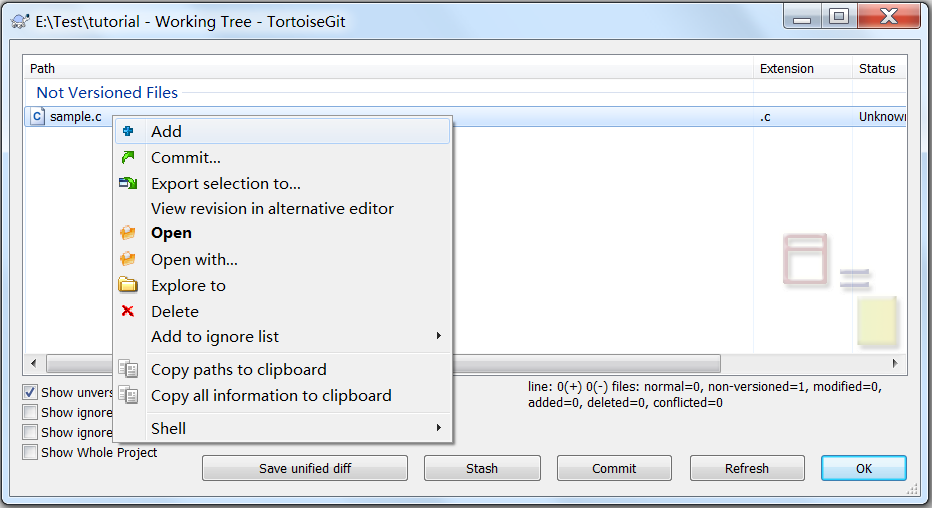
return 0;

}

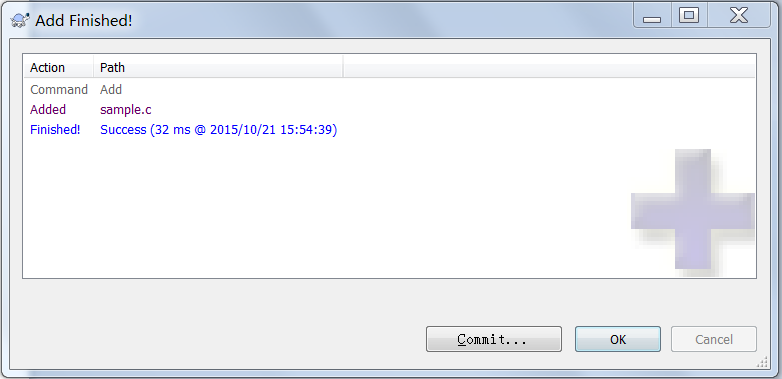
打开tutorial目录，在任意空白地方，单击右键选择TortoiseGit -> Check for modifications，***该选项可以列出tutorial目录及其子目录下所有新增（变动）的文件***。



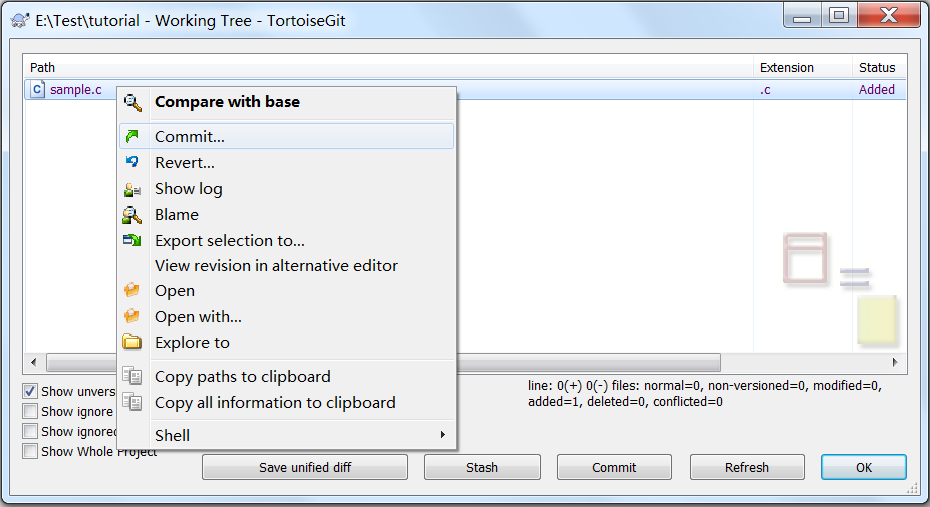
因为我们只新创建了一个sample.c，因此下面的对话框中只列出一个sample.c，右键单击列表中的sample.c，点击Add。所有新增文件，都需要先Add，然后才能够Commit。



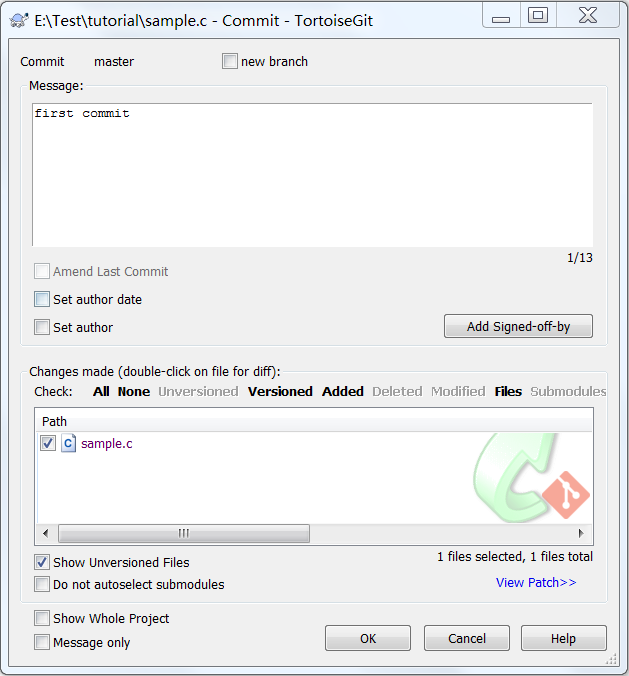
弹出以下对话框，则说明Add成功，点击“OK”以关闭对话框。



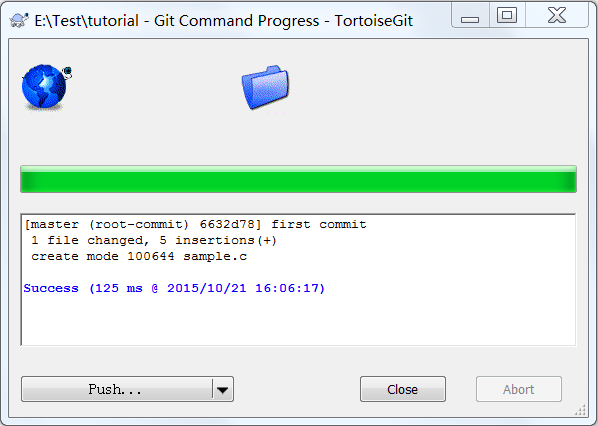
然后再右键单击列表中的sample.c，继续点击Commit。



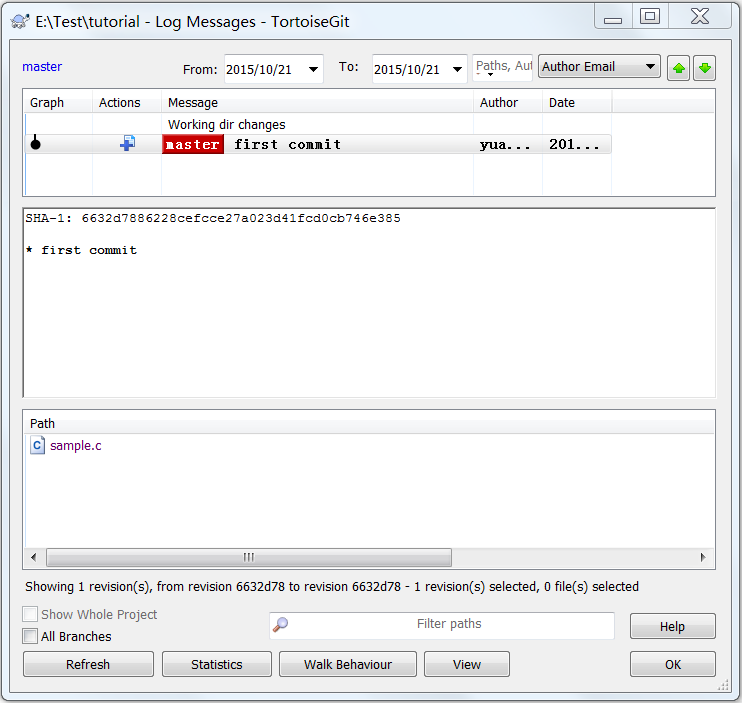
弹出以下对话框后，请点选“Changes made”里的sample.c，然后在“Message”对话框中输入提交信息，这里输入“first commit”，然后点击“OK”以关闭对话框。



如果显示Success，则说明Commit成功，点击“Close”以关闭对话框。



在tutorial目录空白处点击右键，选择TortoiseGit -> Show log，可以看到刚才的提交已添加在历史记录里。我们就算成功提交文件啦！



# 远程仓库

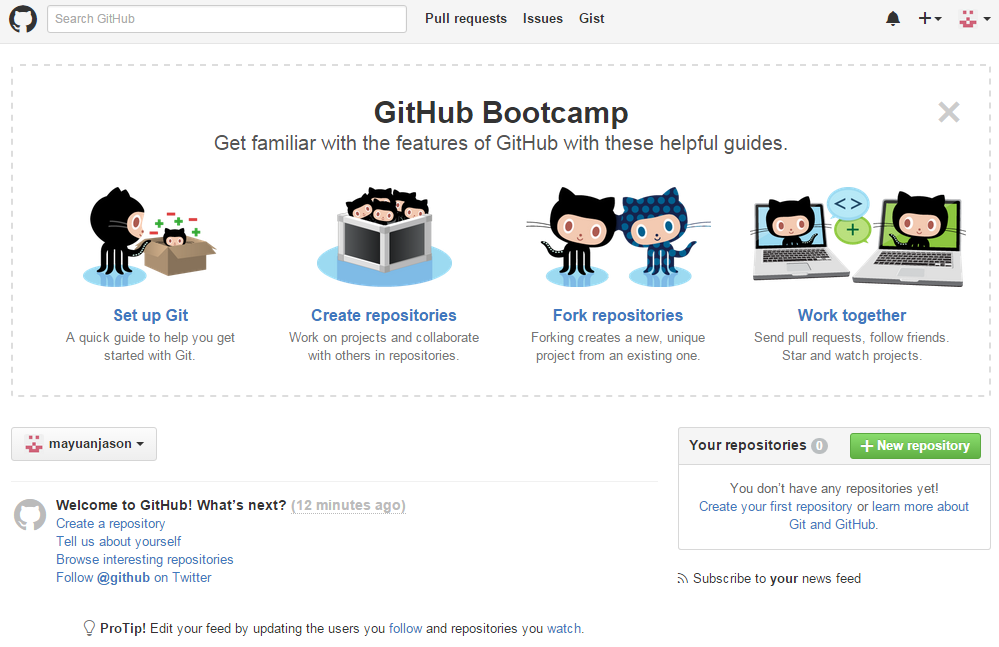
这里本来应该拿我们自己的Git远程仓库举例，但由于不方便在我们正式的Git远程仓库上进行操作。因此，这里使用GitHub提供的远程仓库来进行说明。

## 在GitHub上建立远程仓库

首先需要再GitHub上注册账号，网址：<https://github.com/。> liubing-git 密码:iuy.......



注册并登录后，在登录页面中点击New repository。

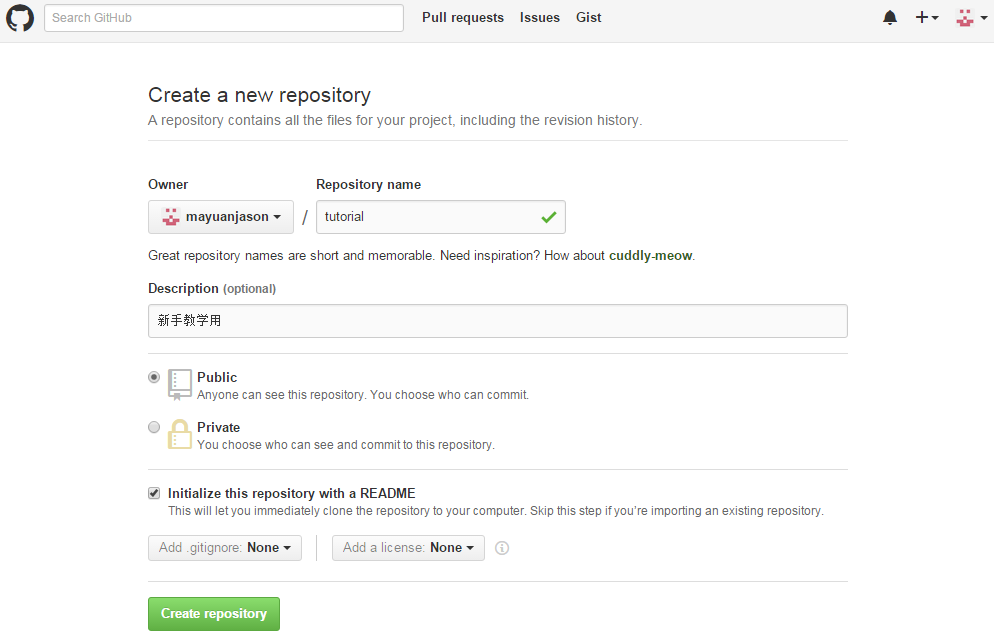


输入远程仓库的名字及其描述说明，这里我们在Repository name一栏输入“tutorial”，在Description一栏输入“新手教学用”。

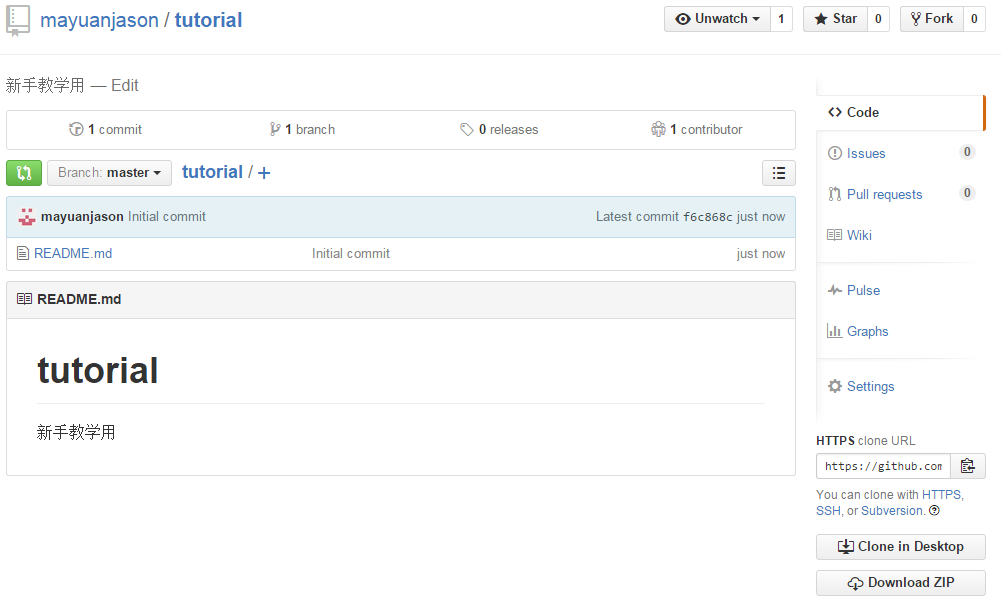
选择Public，所有人都可以访问这个远程仓库。

勾选Initialize this repository with a README。不用勾选

点击Create repository。



见下图，远程仓库就成功建立好了。



## push到远程仓库

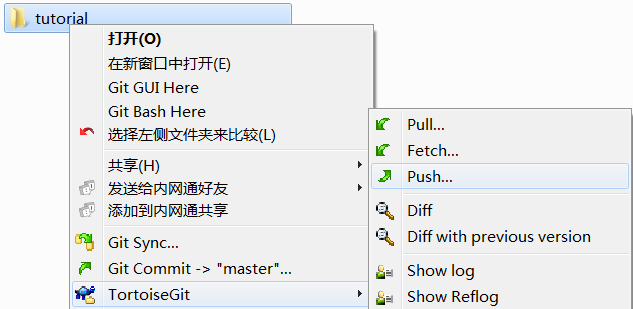
前面讲了本地仓库的基本使用方法。下面，接着讲解如何在GitHub的远程仓库上共享本地仓库的修改记录。

为了将本地仓库的修改记录共享到远程仓库，必须上传本地仓库中存储的修改记录。

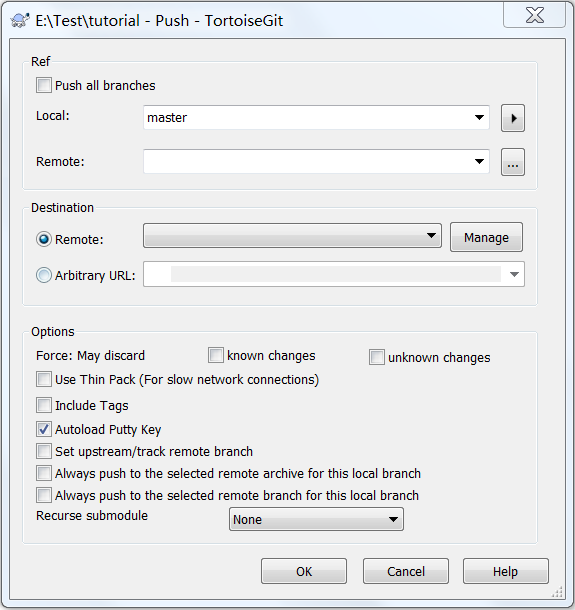
为此，需要在Git上执行Push操作。执行Push之后，本地的修改记录会被上传到远程仓库。所以远程仓库的修改记录就会和本地仓库的修改记录保持同步。这样，GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作，真是一举多得。

这里我们试试push在“Git的基础”章节中创建的本地仓库tutorial吧。

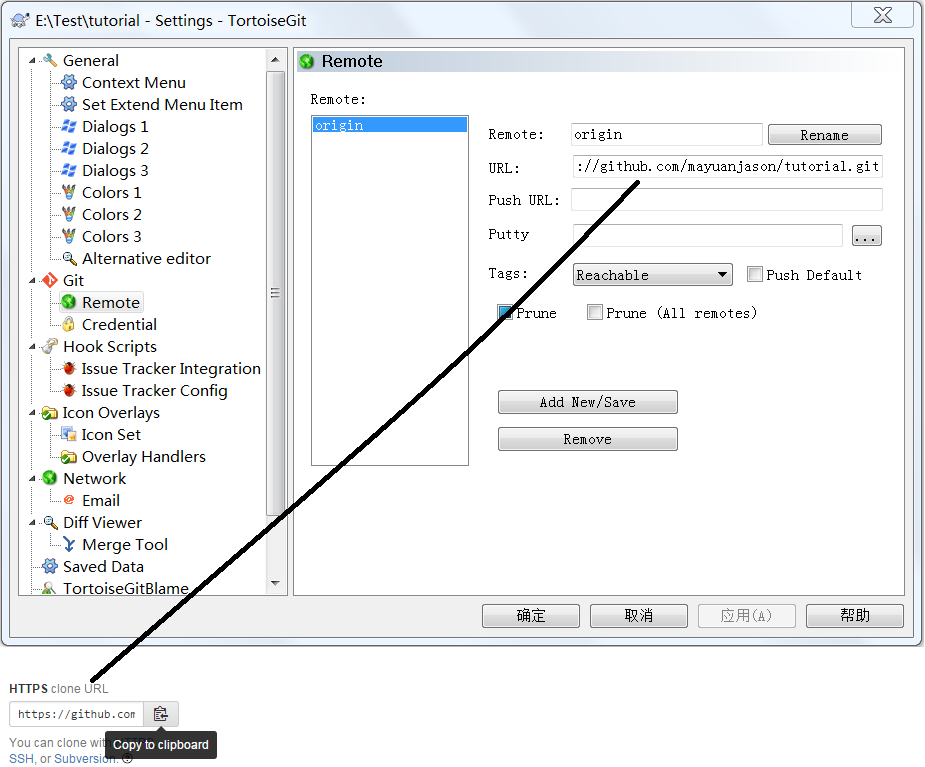
右击tutorial目录，然后选择TortoiseGit -> Push...。



在以下界面点击Manage。

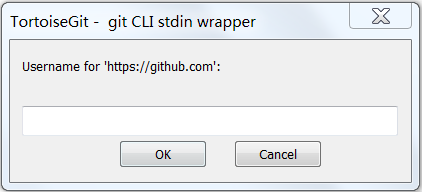


将出现以下画面，在“Remote”输入“origin”，在“URL”输入在GitHub上创建的远程仓库的URL，然后点击“Add New/Save”。这样，“origin”将被添加到远程列表，然后点击“确定”。

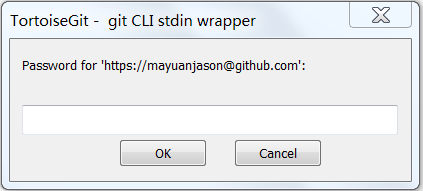


在“Push”界面的“Remote”选项中，选择刚才创建的“origin”，点击“OK”。

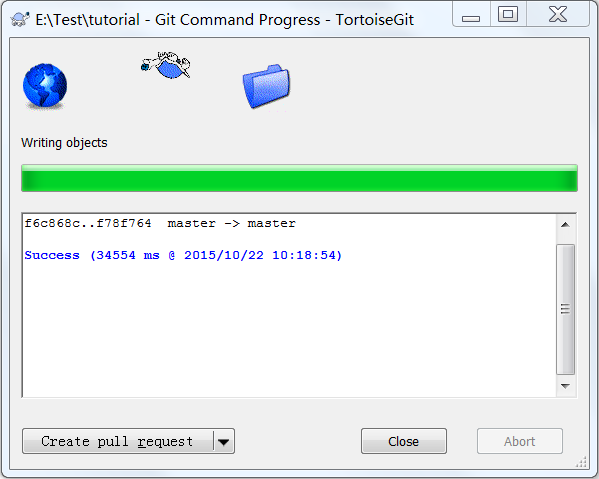
当被要求输入Username时，请使用在GitHub上注册的用户名。



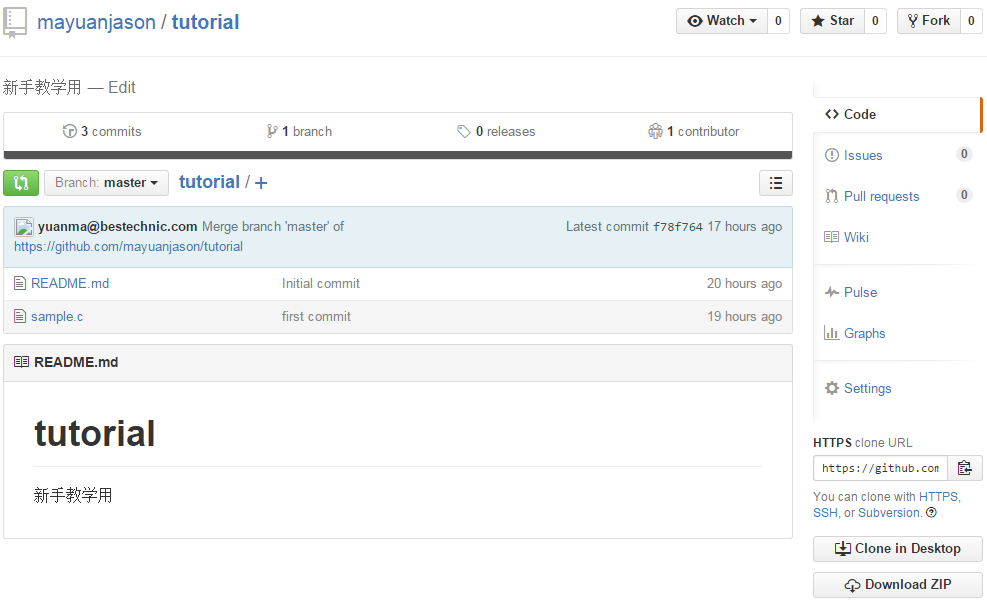
当被要求输入Password时，请使用在GitHub上注册的密码。



若下一个画面显示“Success”字样，就意味着您推送成功了。



打开/刷新GitHub中创建的tutorial仓库页面，可以看到被推送的文件已经添加到远程仓库的文件列表中。

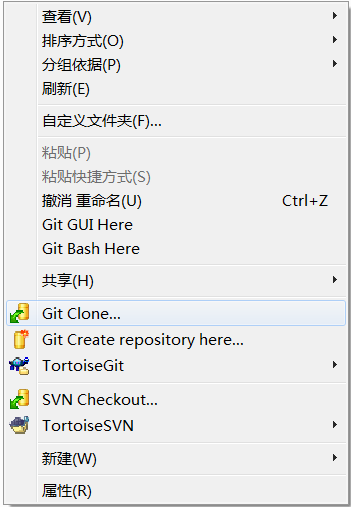


## Clone远程仓库

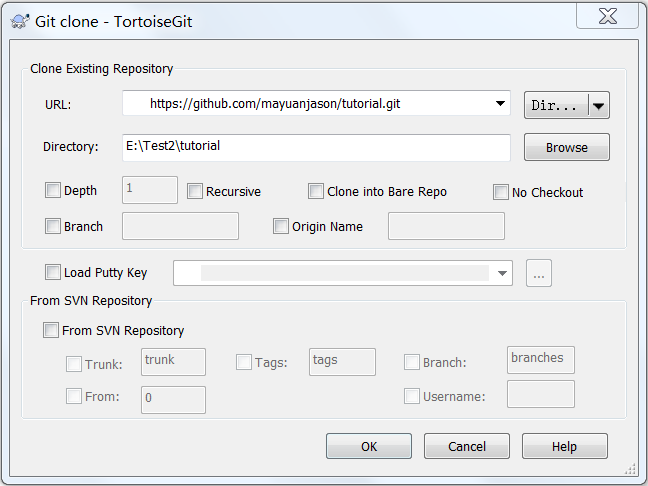
假设你是团队中的另一位成员，要协作一起开发tutorial中的内容，这就需要将tutorial远程仓库的内容Clone到你本地机器。

执行Clone后，远程数据库的全部内容都会被下载下来，之后你就可以在另一台机器的本地仓库上接着进行工作了。

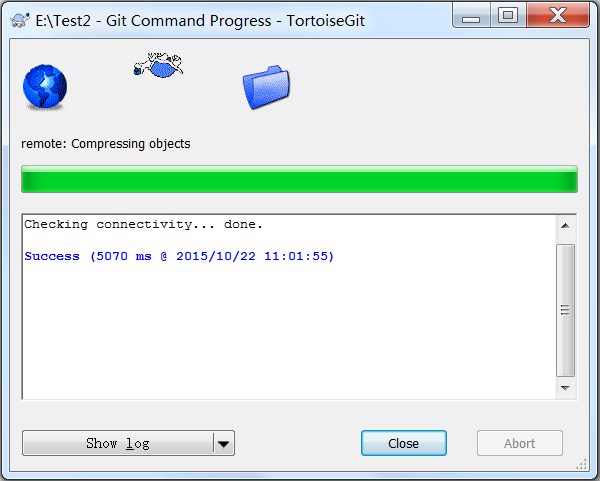
这里我们就在E盘根目录新建Test2文件夹，进入Test2，在空白处单击右键选择Git Clone...。



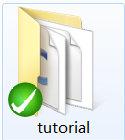
在弹出的“Git clone”对话框中，输入要Clone的远程仓库的URL和要保存的本地仓库的目录，然后点击“OK”。



Clone将在以下画面开始进行。完成Clone后请点击“Close”以退出。



已创建的“tutorial”目录。



若要验证Clone是否成功，请在“E:\Test2\tutorial”中的sample.c中是否有以下代码。

int main(int argc, char \*argv[]) {

int a;

a = 1;

return 0;

}

## 从Clone的仓库进行push

***注：以下操作全部都在新Clone的 tutorial本地仓库上进行（E:\Test2\tutorial）。***

首先，在E:\Test2\tutorial \sample.c中 添加以下代码，并提交。

int main(int argc, char \*argv[]) {

int a;

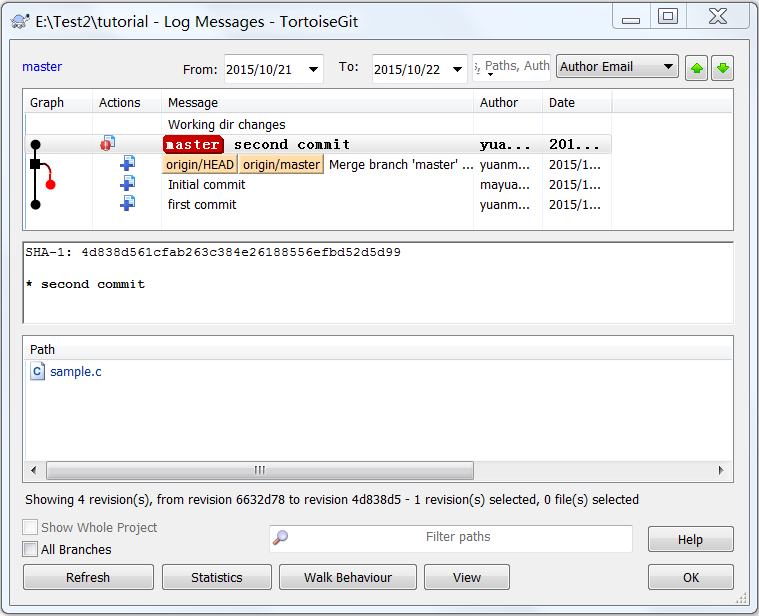
a = 1;

***a = a + 1;***

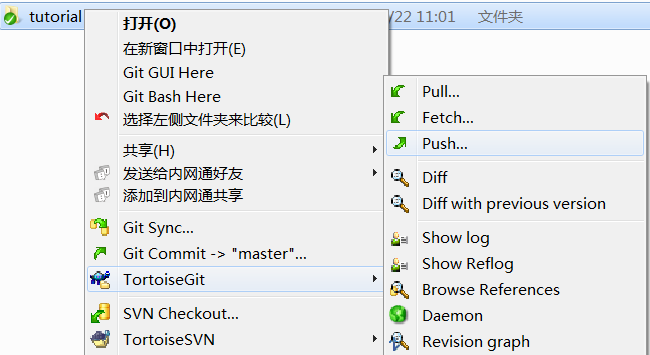
return 0;

}

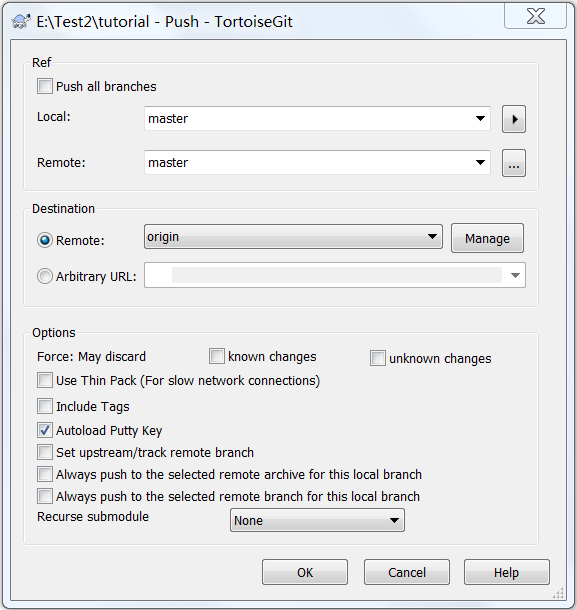
单击右键选择 TortoiseGit -> Show log，可以看到本地仓库分支的历史记录已更新，并位于远程数据库分支的前面



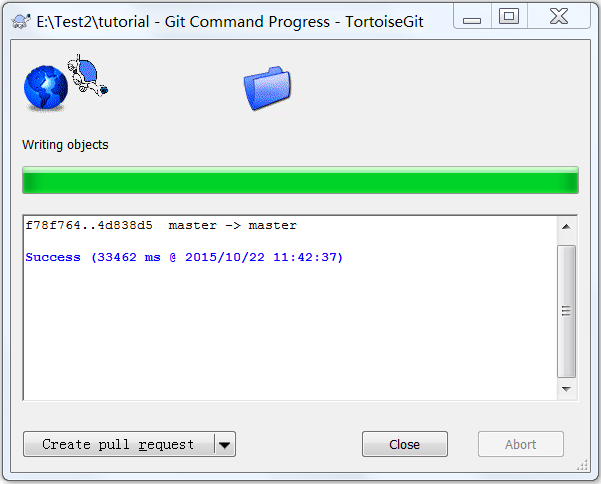
接着，让我们推送新提交到远程仓库。请右击目录"tutorial"并选择“TortoiseGit -> Push...”。



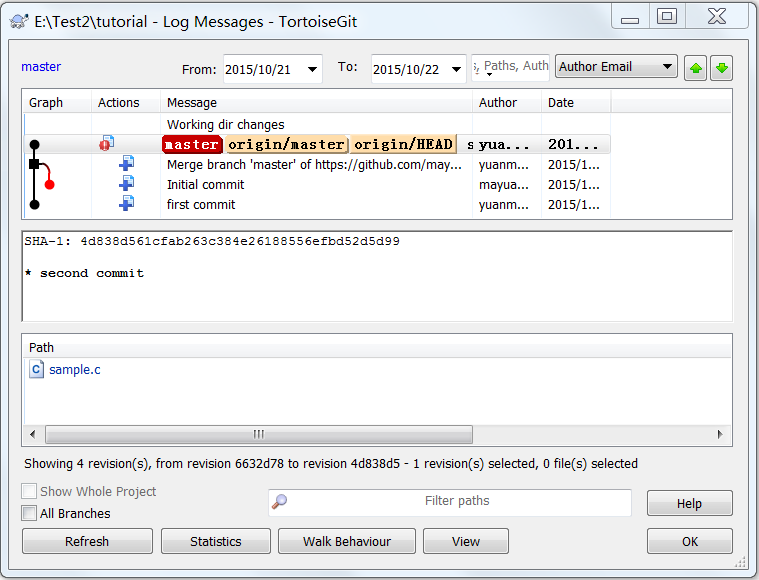
在以下画面点击“OK”。



Push将在以下画面开始进行。完成Push后请点击“Close”以退出。



单击右键选择 TortoiseGit -> Show log。现在“master” 和 “origin/master”在同一水平上，意味着远程仓库已完成更新，并且跟本地仓库同步。



# 从远程仓库pull

若是共享的远程仓库由多人同时作业，那么作业完毕后所有人都要把修改Push到远程仓库。然后，自己的本地仓库也需要更新其他人Push的变更内容。

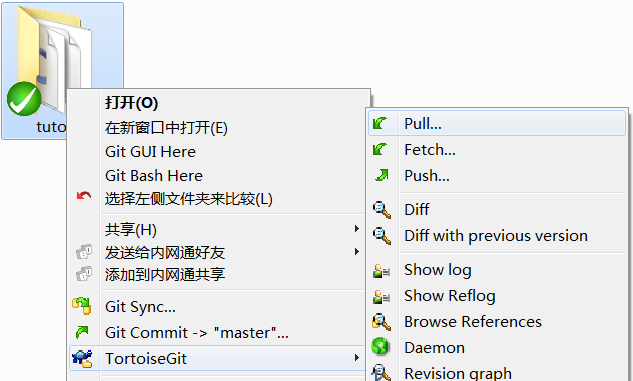
进行Pull操作就可以把远程仓库的内容更新到本地仓库。

进行Pull操作，就是从远程仓库下载最近的变更日志，并覆盖自己本地仓库的相关内容。

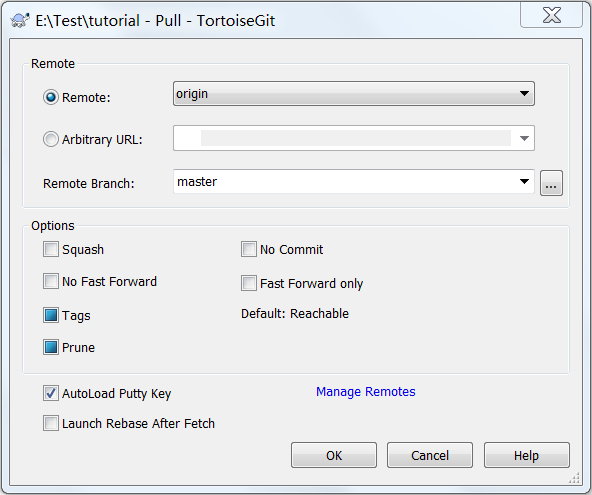
我们把在上一节中从“E:\Test2\tutorial”Push到远程仓库的内容拉取到本地仓库。

***注：以下操作全部都在原tutorial本地仓库上进行（E:\Test\tutorial）。***

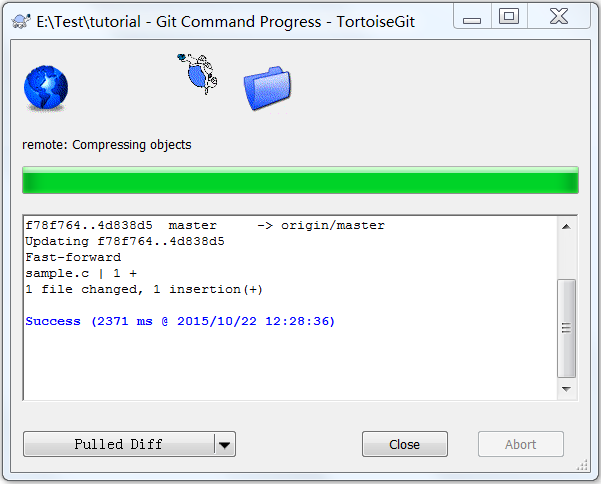
右击tutorial目录，然后从右击菜单中选择TortoiseGit -> Pull...，即可执行pull操作。



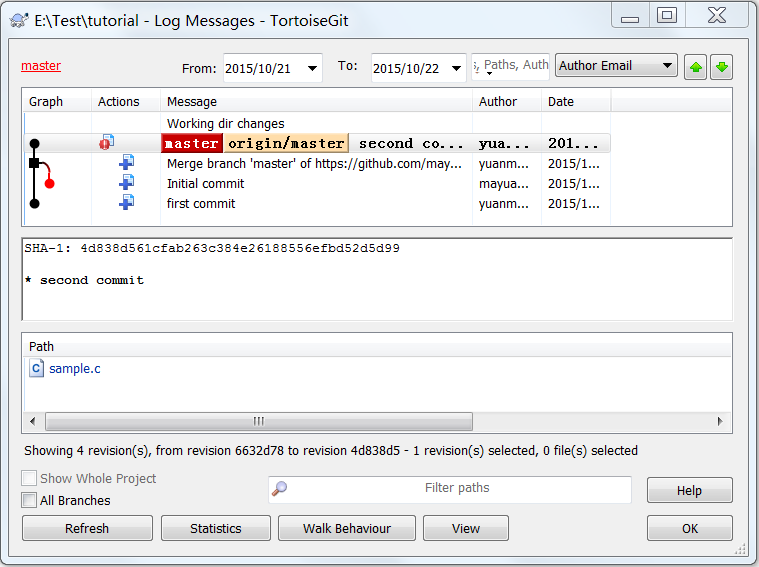
在以下画面点击“OK”。



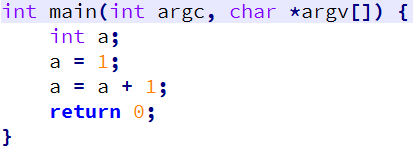
Pull操作将在以下画面开始进行。完成Pull后请点击“Close”以退出。



右击菜单选择 TortoiseGit -> Show log。现在“master” 和 “origin/master”在同一水平上，意味着本地仓库已完成更新。



打开E:\Test\tutorial\sample.c文件来确认一下内容吧。



我们可以看到追加的内容“a = a + 1”。

# 整合修改记录

## 合并修改记录

Git会把提交记录串成一条时间线。每次提交，这条时间线都会增加一个节点，记录最新一次的提交记录。这样，随着你不断提交，这条时间线也会越来越长。

但是，在执行Pull之后，进行下一次Push之前，如果其他人Push了内容到远程仓库的话，那么你的Push将被拒绝。

这种情况下，在读取别人Push的变更并进行合并操作之前，你的Push都将被拒绝。这是因为，如果不进行合并就试图覆盖已有的变更记录的话，其他人Push的变更（图中的提交C）就会丢失。

合并的时候，Git会自动合并已有的变更点！不过，也存在不能自动合并的情况。如果远程仓库和本地仓库的同一个地方都发生了修改的情况下，因为无法自动判断要选用哪一个修改，所以就会发生冲突。



## push冲突的状态

这一节我将讲一下怎样解决冲突。

首先，我们用“E:\Test\tutorial”和“E:\Test2\tutorial”制造一个冲突状态。

用“E:\Test\tutorial”进行操作

首先，打开E:\Test\tutorial目录的sample.c，添加以下标红内容之后提交本地仓库。

int main(int argc, char \*argv[]) {

int a;

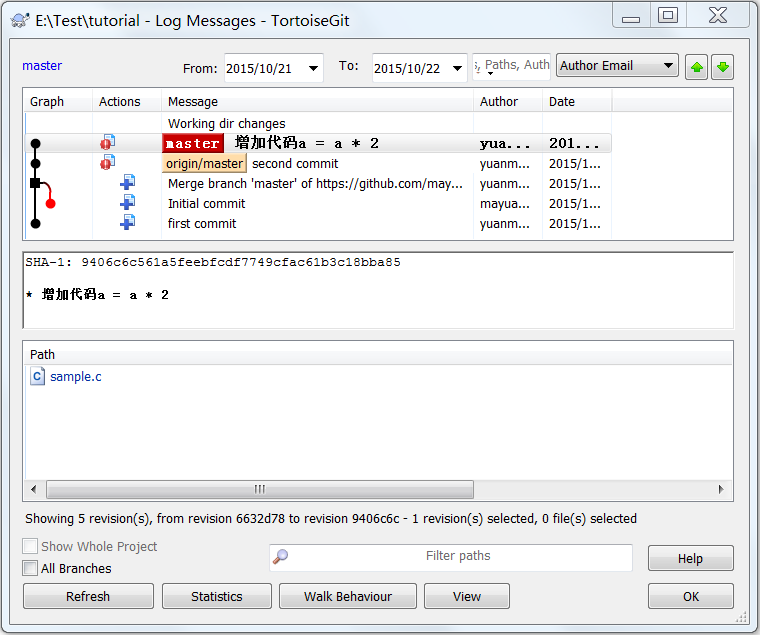
a = 1;

a = a + 1;

***a = a \* 2;***

return 0;

}



用“E:\Test2\tutorial”进行操作

接下来，打开E:\Test2\tutorial目录的sample.c，添加以下标红内容之后提交本地仓库。

int main(int argc, char \*argv[]) {

int a;

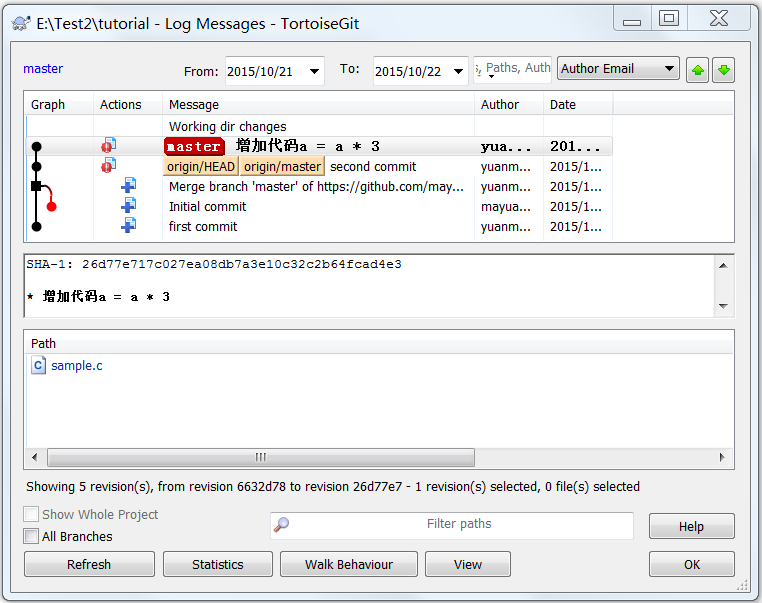
a = 1;

a = a + 1;

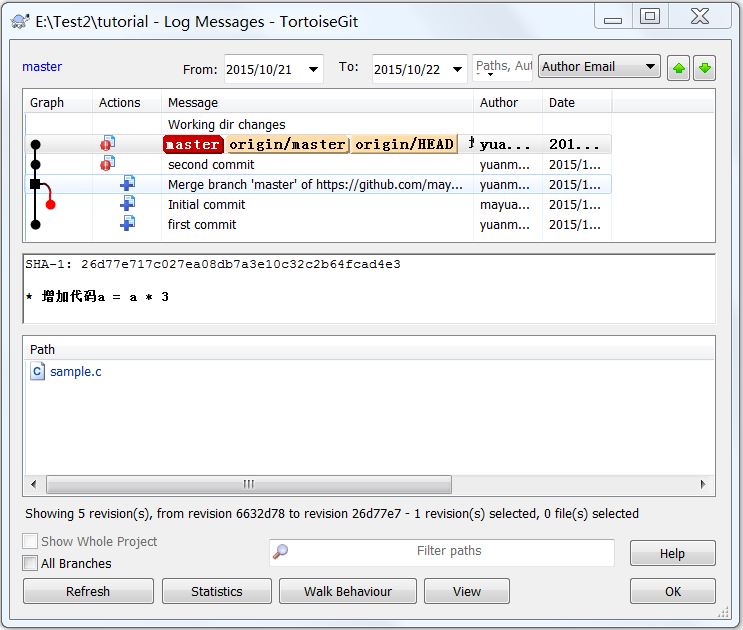
***a = a \* 3;***

return 0;

}



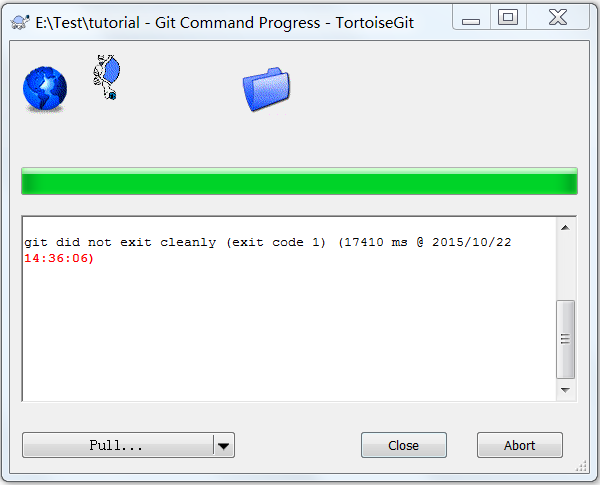
现在从E:\Test2\tutorial Push到远程仓库。



***在目前的远程仓库，sample.c已包含“a = a \* 3;”，并且已被存储到历史记录中啦。***

用“E:\Test\tutorial”进行操作

现在，我们将要把提交从“E:\Test\tutorial”本地仓库Push到远程仓库。

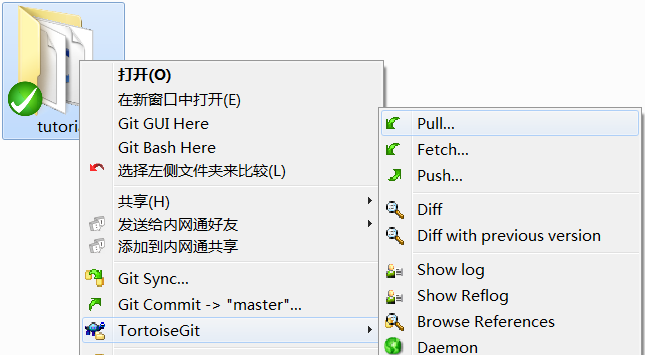


可以看到，Push并没有成功。

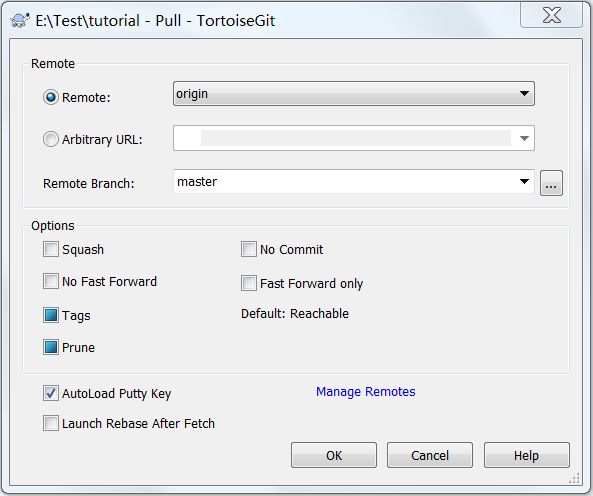
## 解决冲突

为了把变更内容Push到远程数据库，我们必须手动解决冲突。首先请运行Pull，以从远程仓库取得最新的变更记录吧。

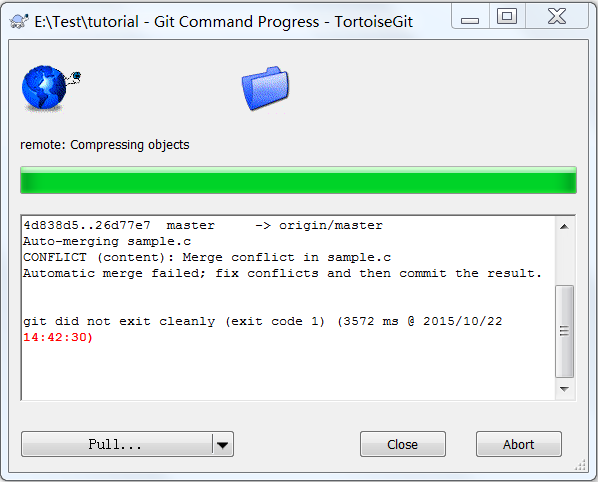
右键单击E:\Test\tutorial目录，并选择TortoiseGit -> Pull...。



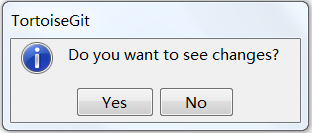
在以下画面点击“OK”。



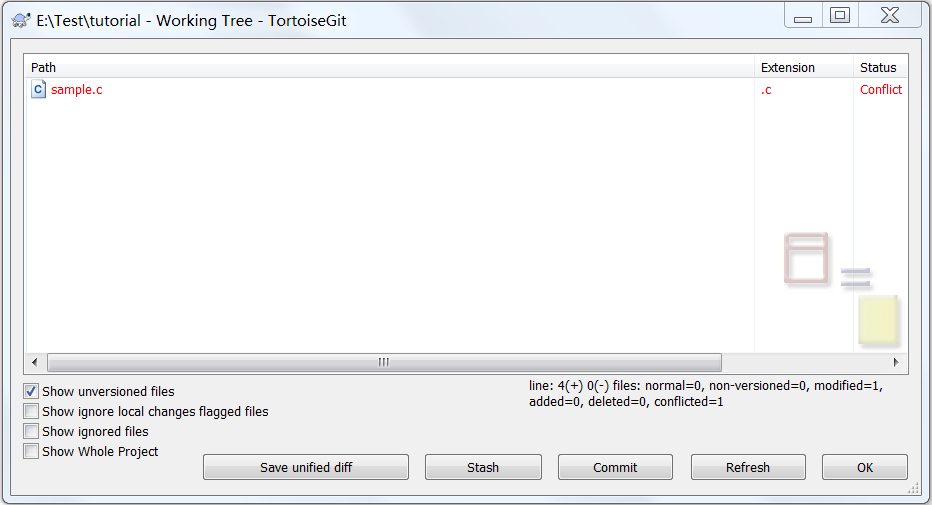
我们看到画面上的警告信息表示自动合并失败，发生冲突。请点击“Close”以退出。



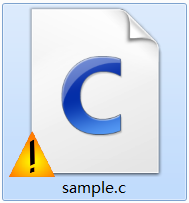
若你确认变更，请点击“Yes”。



TortoiseGit告诉我们：因“sample.c”出现了合并冲突，所以自动合并失败。请点击“OK”以关闭画面。



同时，sample.c的左下角会出现一个黄色的“！”号图标。

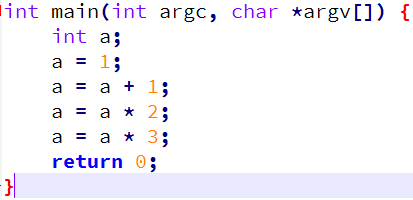


当打开sample.c，你会看到Git已添加标示以显示冲突部分。

其中，“=======”分割线上方是本地仓库的内容, 下方是远程仓库的内容。

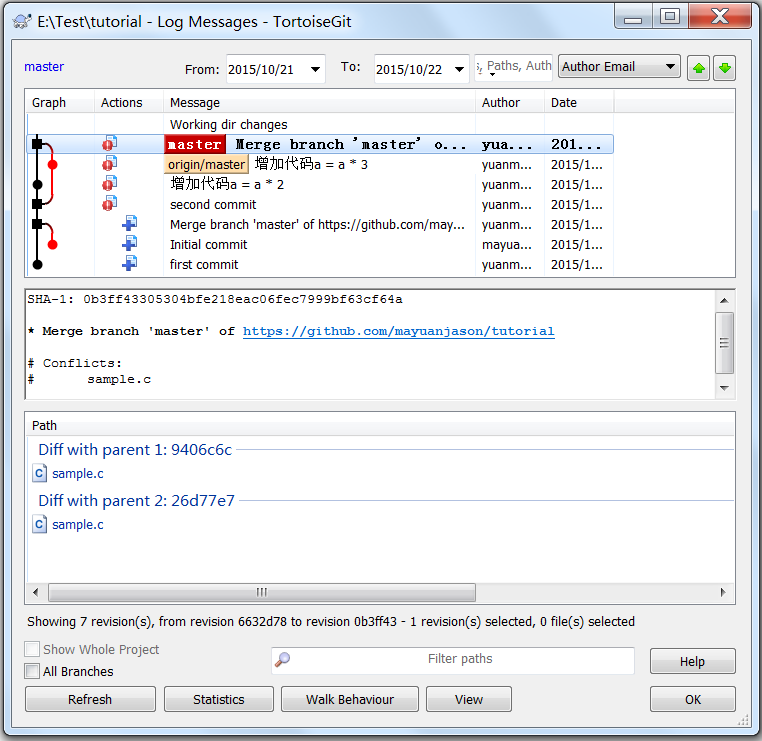


我们将保留两方的修改，并删除多余的标示行以解决冲突。



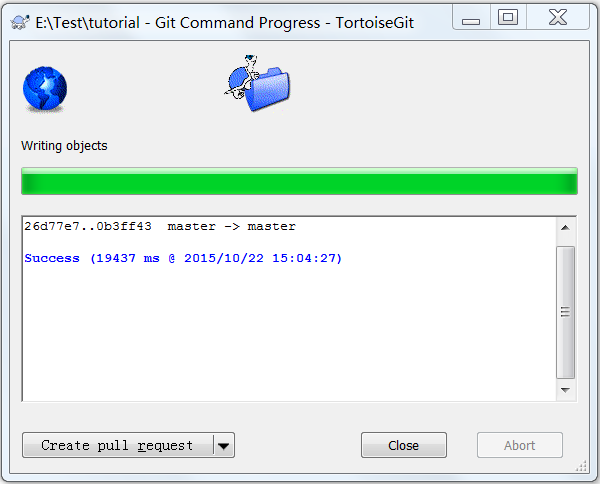
既然解决了冲突，并且文件的内容已修改，现在我们需要提交至本地仓库。

这样，我们已成功地从远程仓库导入最新的修改内容。



这表明两个修改记录通过新的合并提交已经成功整合了。

这时候可以把修改推送到远程数据库。



可以看到，这次没有发生合并冲突，Push成功。

# Download bes1000 rom source code using GIT

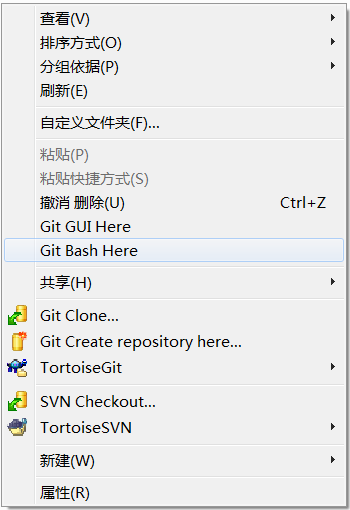
## 从Putty切换到OpenSSH

在“初期设定”章节中提到过，针对每个Git用户，管理员会生成一个公钥和私钥，公钥保存在远端Git服务器上，私钥需要添加到本地TortoiseGit中，并且公钥和私钥都是OpenSSH格式密钥。

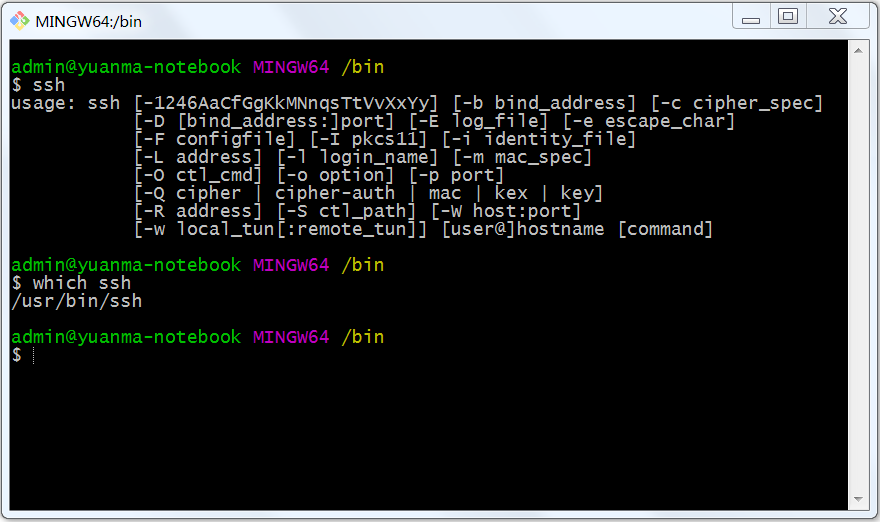
不幸的是，TortoiseGit默认使用Putty作为SSH客户端。因此TortoiseGit使用的是ppk（Putty）密钥格式，而不是使用OpenSSH密钥格式。为此，需要将TortoiseGit的SSH客户端从Putty切换到OpenSSH。

首先，寻找ssh.exe。

在任意目录空白处右键点击Git Bash Here。

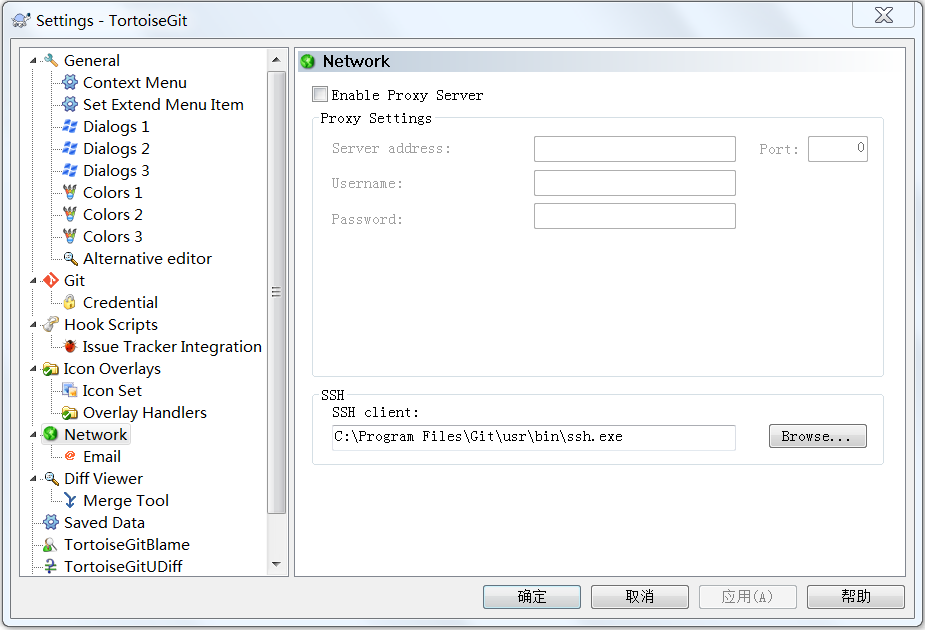


在弹出的shell界面中输入which ssh（Git for Windows默认已经包含ssh.exe）。



可以看到，ssh.exe安装在了“/usr/bin/ssh”中。其中最开始的“/”根目录就是Git for Windows的安装目录，这里是“C:\Program Files\Git”。因此，ssh.exe就存在于“C:\Program Files\Git\usr\bin\ssh.exe”中。

在空白处右键单击TortoiseGit -> Setting，在弹出的对话框左侧选择“Network”。这里只需要将“SSH client” 改成Git for Windows提供的ssh.exe即可。



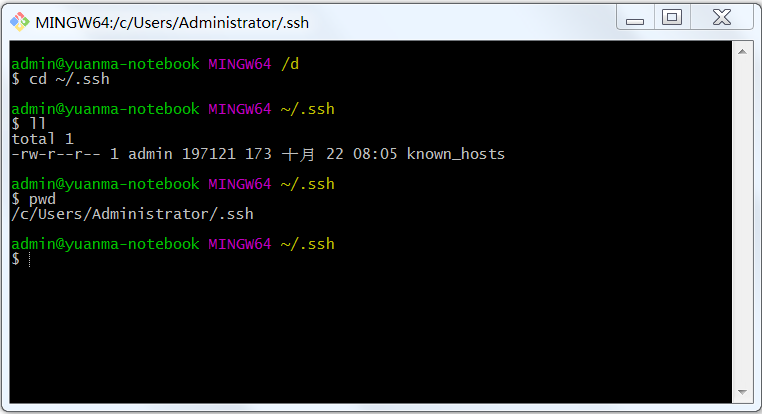
## 添加私钥到客户端

在任意目录空白处右键单击Git Bash Here。在弹出的shell对话框中输入下面几条命令：

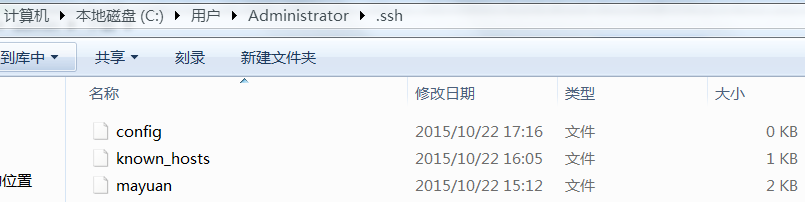
cd ~/.ssh //进入.ssh目录; cat id\_rsa.pub可以输出公钥,输入ssh查看指令语法

ll（两个小写的字母L）

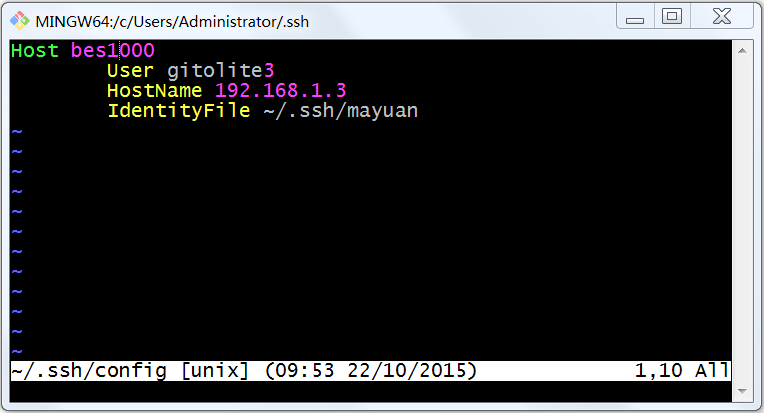
pwd //目录路径 /c/Users/EDZ/.ssh



可以看到，在“C:\Users\Administrator\.ssh”目录下只存在一个“known\_hosts”文件。这里还需要在该目录下新增一个config文件，并将管理员给你的私钥文件拷贝到这个目录中，这里，管理员分配给我的私钥文件名为“mayuan”。



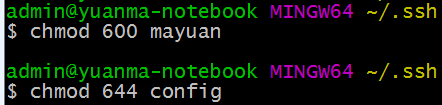
用vi打开config文件并输入以下命令并保存。



最后，在Git Bash shell输入以下命令以提升mayuan和config这两个文件的权限。

chmod 600 mayuan

chmod 644 config

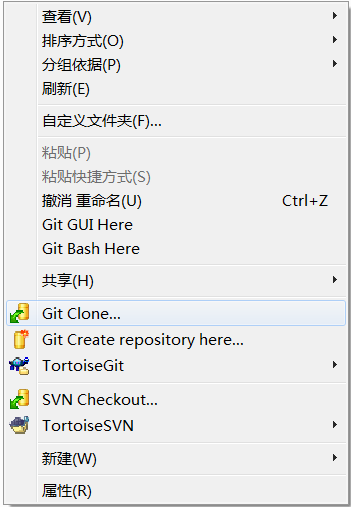


这样，整个私钥就已经成功添加到客户端。

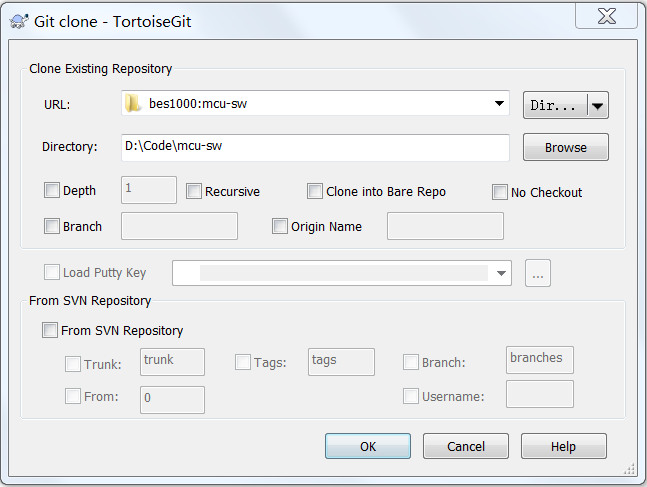
接下来，让我们Clone一份best1000 rom的代码到本地。

## Clone best1000 rom代码到本地

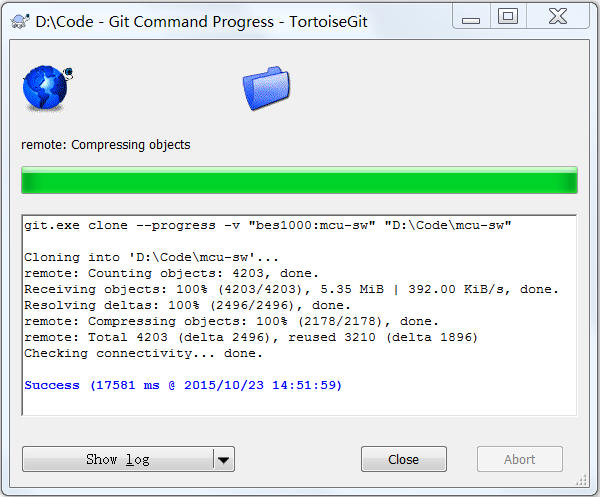
在所选定目录空白处单击右键选择Git Clone...。



在弹出的“Git clone”对话框中，输入要Clone的远程仓库的URL和要保存的本地仓库的目录，然后点击“OK”。这里，“bes1000:mcu-sw”就是best1000 rom的Git服务器地址。



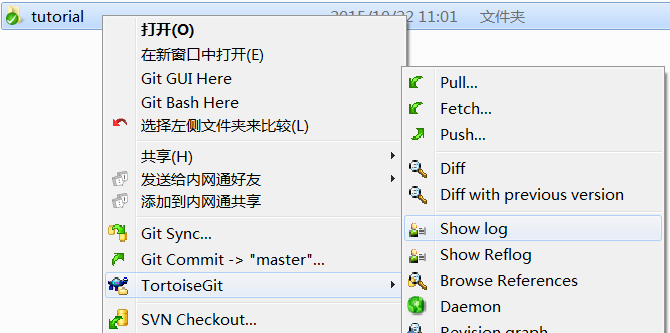
如果显示Success，则说明Clone成功，点击“Close”以关闭对话框。



# 查看每次提交的修改变化点

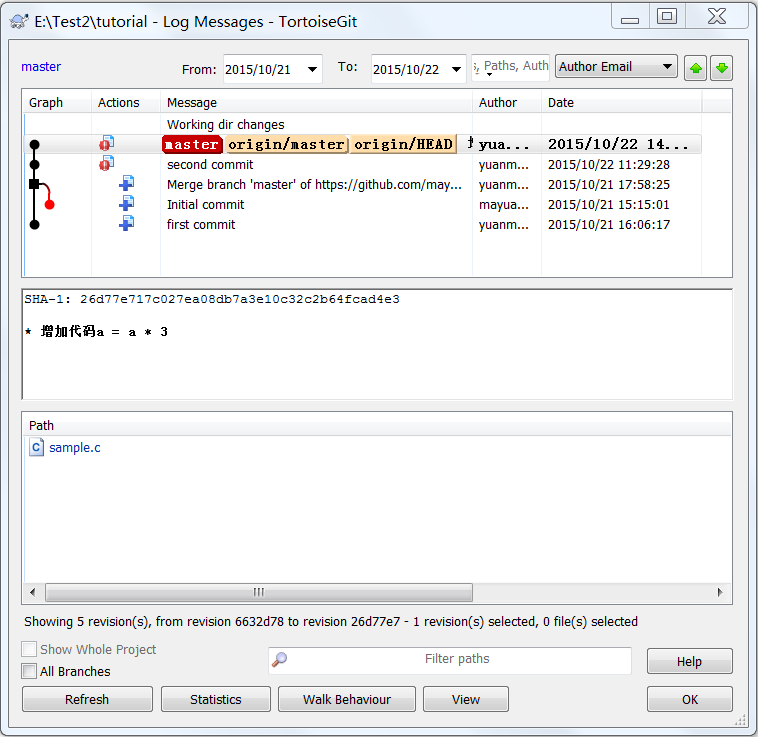
这里还是以tutorial目录为例。

右键单击tutorial目录，选择TortoiseGit -> Show log。

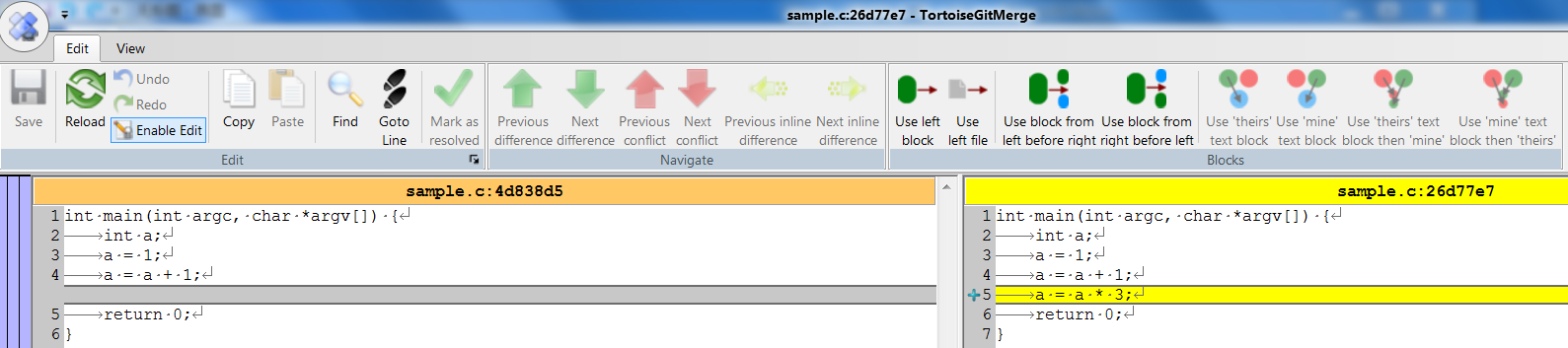


在弹出的对话框中，列出了所有的提交记录。我们就以最上面（最新）的提交记录为例，说明如何查看修改变化点。

鼠标选择最上面的提交记录，在下面的Path对话框中，就列出了相对于上一次的提交，这次提交所修改的文件。由于我们只修改了（只有）一个.c文件，因此这里就只有一个sample.c。

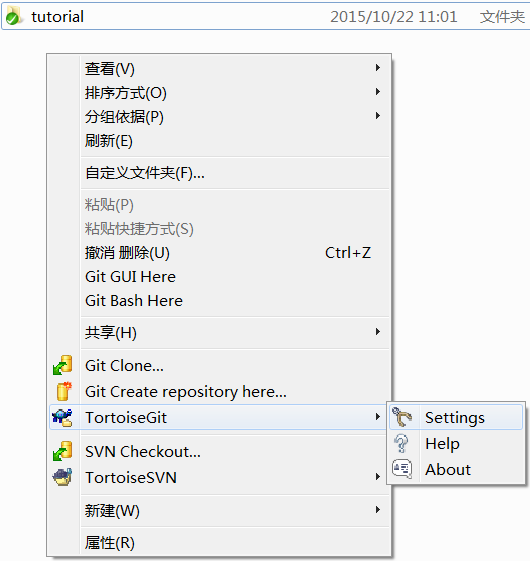


双击sample.c。会弹出一个diff对话框，左边显示的是上一次的提交结果，右边显示的是本次提交结果，并且TortoiseGit用黄色标识出了两次提交记录之间的差异。可以看出，本次提交，新增了一行代码“a = a \* 3”。

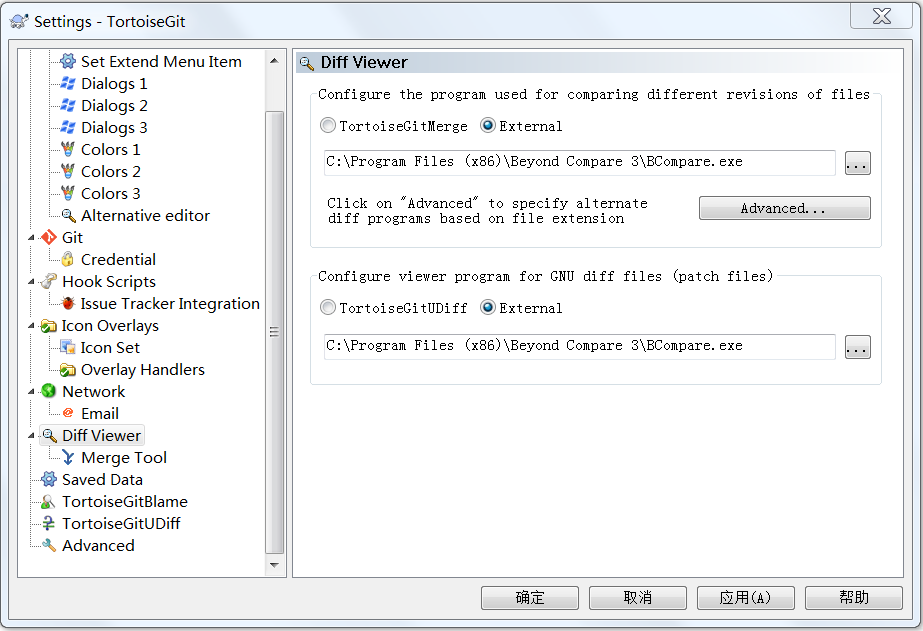


如果大家对这个简陋到丑陋的diff对话框感到不满意的话，还可以用第三方的对比工具来显示修改变化点。这里以Beyond Compare为例，讲解如何用Beyond Compare来代替TortoiseGit自带的比较工具。

在tutorial目录外层空白处右键单击TortoiseGit -> Settings。



在弹出的对话框左侧选择Diff Viewer，在右侧最上面选择External，继而选择你的机器上Beyond Compare的安装路径，然后点击“确定”以关闭。



现在右键单击tutorial目录，选择TortoiseGit -> Show log。在弹出的对话框中双击sample.c。看看效果吧，可以看出，Diff工具已经由TortoiseGit自带的变为Beyond Compare。

