TortoiseGit入门教程

--参考资料：TortoiseGit帮助文档

目录

[1、关于版本控制 2](#_Toc472805949)

[2、TortoiseGit安装： 2](#_Toc472805950)

[2、TortoiseGit文件夹图标说明： 8](#_Toc472805951)

[3、克隆远程库 9](#_Toc472805952)

[4、对文件进行修改 12](#_Toc472805953)

[5、提交更改。 13](#_Toc472805954)

[6、push推送变更到库 17](#_Toc472805955)

[7、从远程库更新 18](#_Toc472805956)

[8、与远程库同步 19](#_Toc472805957)

[9、解决冲突 23](#_Toc472805958)

[10、TortoiseGitMegar 27](#_Toc472805959)

[11、查看本地文件与基础文件的差异 30](#_Toc472805960)

[12、查看别人的修改 36](#_Toc472805961)

[13、查看文件的所有修改者 39](#_Toc472805962)

[14、导出某一版本的文件 40](#_Toc472805963)

[15、慎用Clean Up 43](#_Toc472805964)

[16、.gitignore和不需提交的文件 44](#_Toc472805965)

[17、附：删除windows凭据 45](#_Toc472805966)

1、关于版本控制

版本控制是管理信息修改的艺术，它一直是程序员最重要的工具，程序员经常会花时间作出小的修改，然后又在某一天取消了这些修改，想象一下一个开发者并行工作的团队 - 或许是同时工作在同一个文件！- 你就会明白为什么一个好的系统需要管理潜在的混乱。这是来自TortoiseSVN文档的一段话，总结得很精辟。

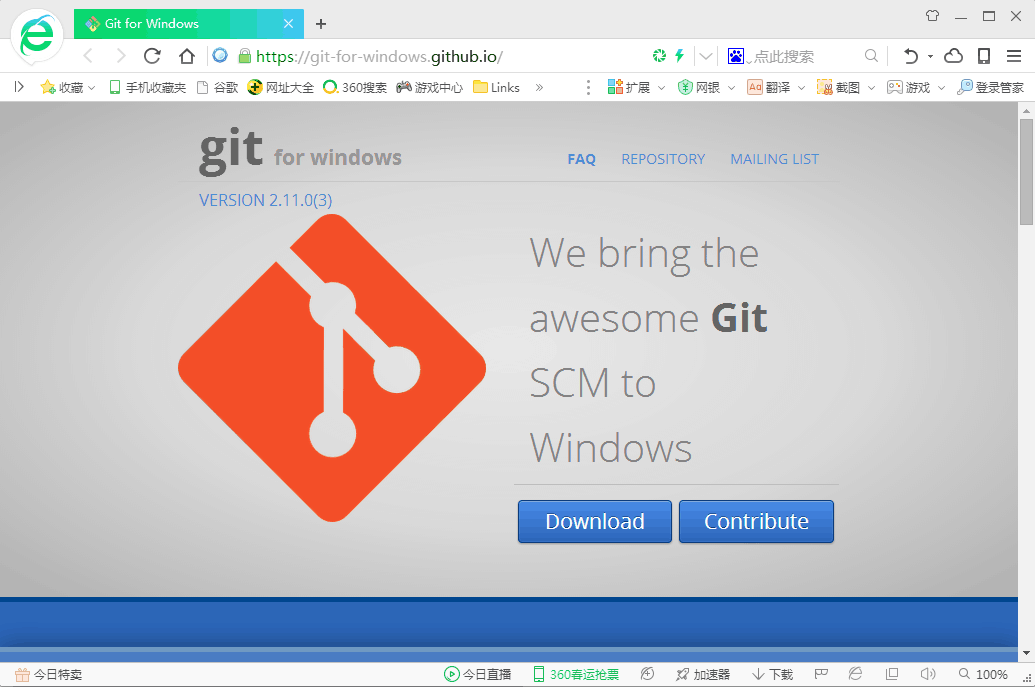
TortoiseGit 是一个 Windows 下的版本控制系统Git的客户端工具,正如其名字一样，小乌龟外面有一个壳，TortoiseGit也只是Git一个壳，使您更加方便的使用Git。Git是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。版本库就像一个常见的文件服务器，除了它保存你对文件和目录所有的改变。这一特性使得你可以恢复文件的旧版本并查看历史-谁在什么时间如何进行的修改。因此也可以说版本控制系统是一种“时间机器”。

TortoiseGit现在有多语言包可以下载，本文档使用英文版进行说明。

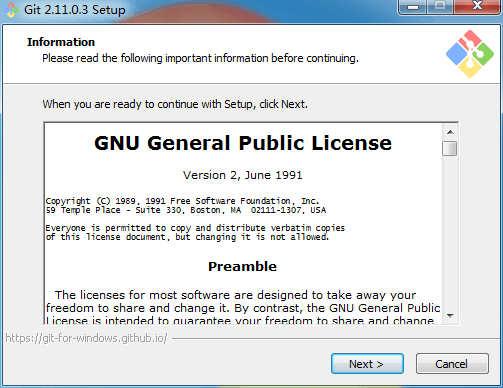
本文档是以个人经验参考官方文档整理出来的，可能会存在很多问题，如有理解不正确的地方希望能够指出，大家共同学习吧。

2、TortoiseGit安装：

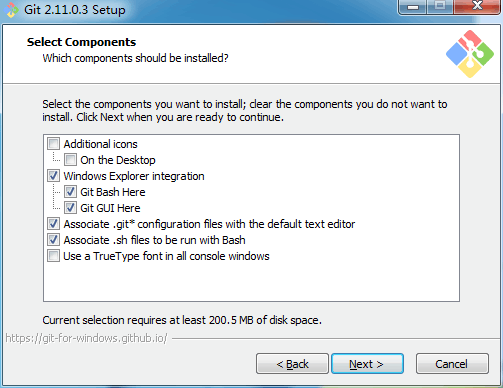
首先安装git for windows，下载地址：<https://git-for-windows.github.io/>



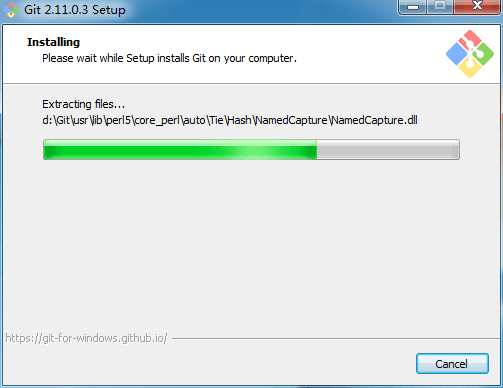
下载完成后安装：

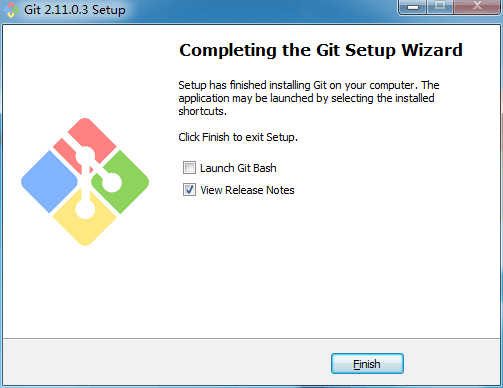


默认设置即可：



一路“next”直到安装完成：



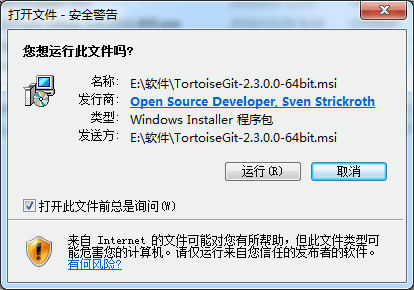


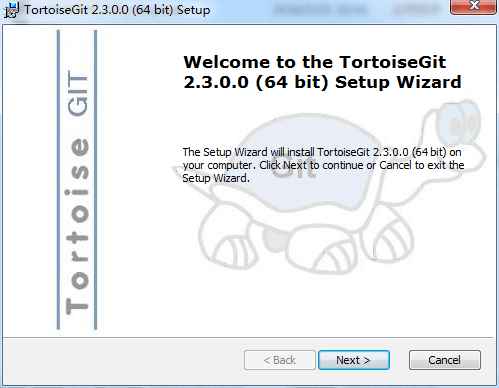
安装tortoisegit：

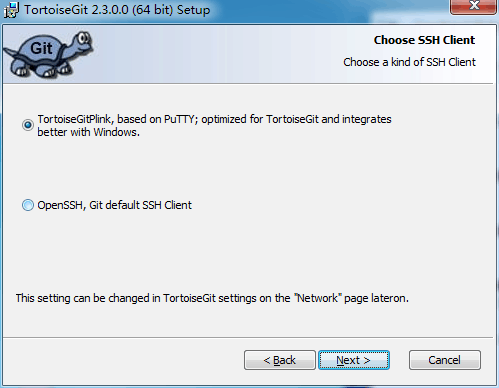
从网站https://tortoisegit.org/download/上下载安装程序。

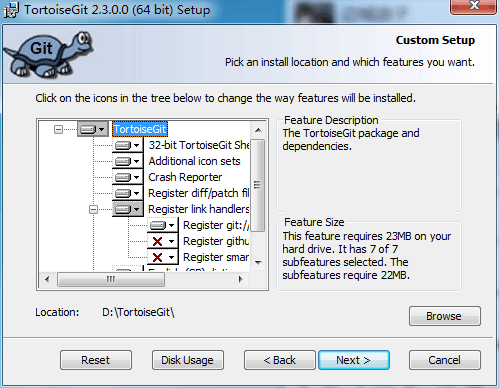
E:\Tencent Files\175016513\Image\C2C\T0BTLK05@F_I_U2{Z0PZF%6.jpg

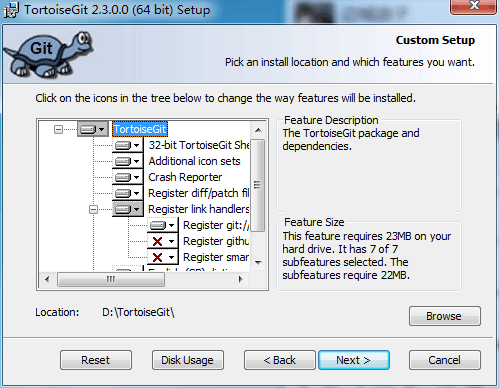
双击安装文件进行安装：

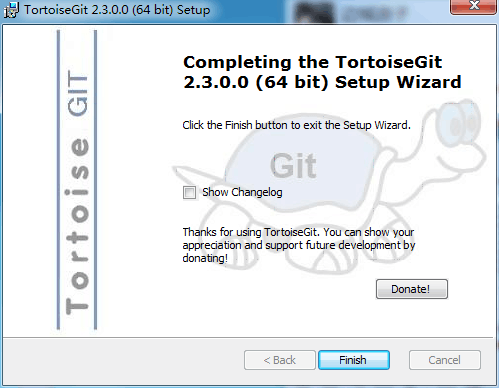












2、TortoiseGit文件夹图标说明：

正常文件，从上次与远程库同步后本地没有更改。

本地更改过的文件，只要文件一编辑，就会出现这个图标。

有冲突的文件。

执行“git update-index”命令后显示该图标，表示文件即将提交，但还没有提交。

表示文件已经删除或丢失。

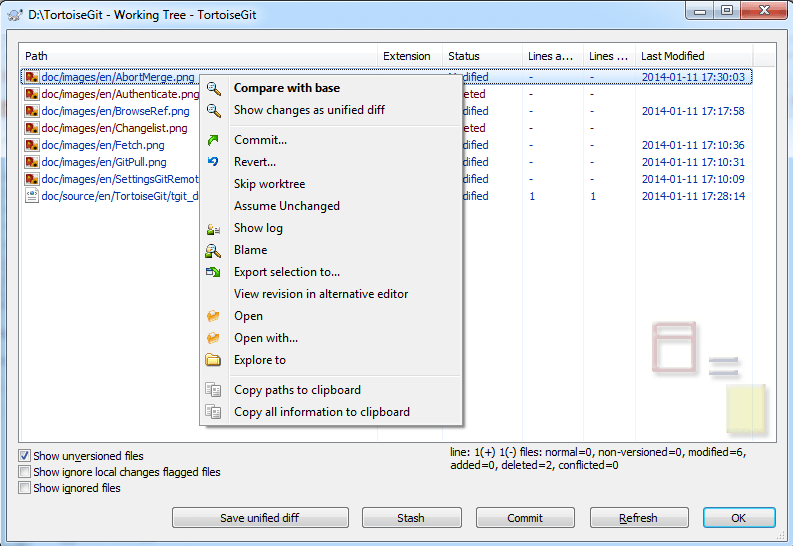
不受版本控制的文件，这种文件不会被提交。



* 新添加的文件，还没有加到版本控制里。

 新添加的文件，已经添加到版本控制里，但还没有提交。

借用官方文档上的一张图说明一下文件列表中各文件状态用颜色的表示：



蓝色：本地修改了的文件。

紫色：新增加的文件，已经添加到版本控制里，但还没有提交。

深红：删除或丢失的文件。

绿色：本地和远程修改了的文件，更新时可能产生冲突或合并。

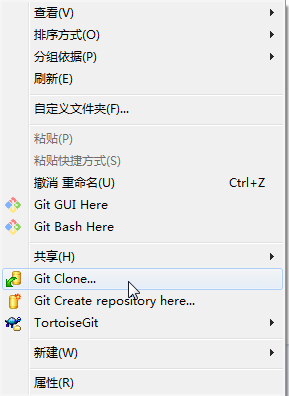
红色：本地库删除了文件而远程进行了修改或者相反，更新时可能产生冲突。

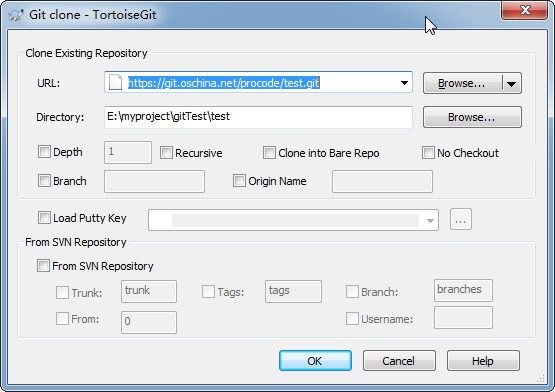
黑色：未修改或未加入版本控制的文件。

3、克隆远程库

从现在开始要正式使用git来做版本控制了，首先我们从git远程库将项目克隆（clone）到本地：

(1)在要建立项目的文件夹点击右键，选择“git clone…”：

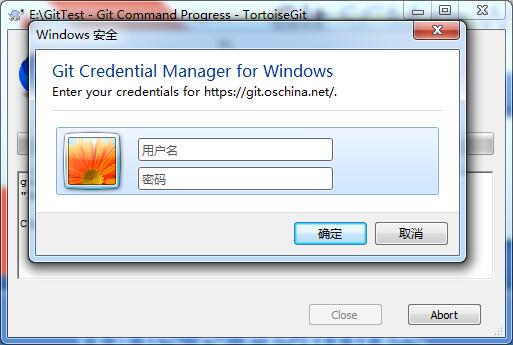




URL：填写git远程库的地址。

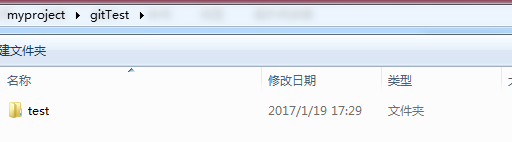
Directory：克隆目的地，默认为当前文件夹，不用修改。

完成后点击“ok”，这时可能需要你添加用户名和密码，如下图：

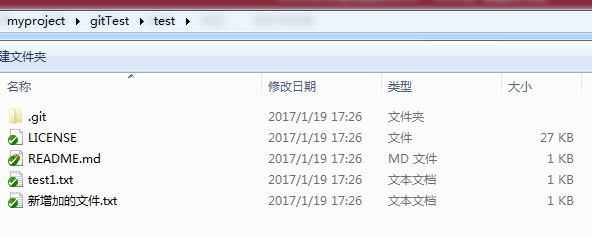


Windows会自动记住用户名和密码信息，如需要修改和删除请看附录删除windows凭据部分。

点击确定后git会将远程库的文件clone到本地，如下图：



打开文件夹进入项目文件夹内：

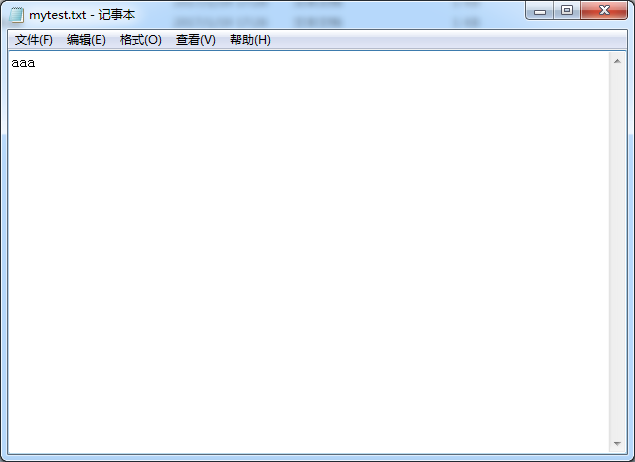


这里的标记表示文件是正常的，没有进行过修改。

4、对文件进行修改

Clone下来后就可以对文件进行正常编辑等操作了。

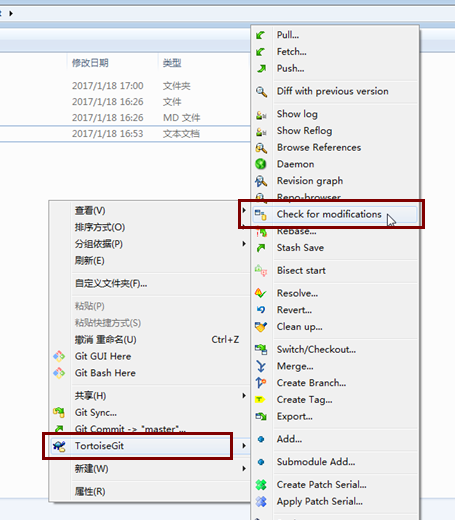
先新建一个文本文件“mytest.txt”，输入内容“aaa”，

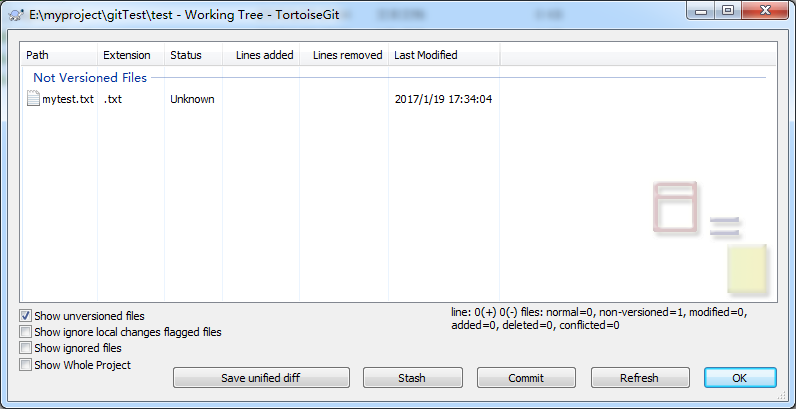


编辑完成后保存。

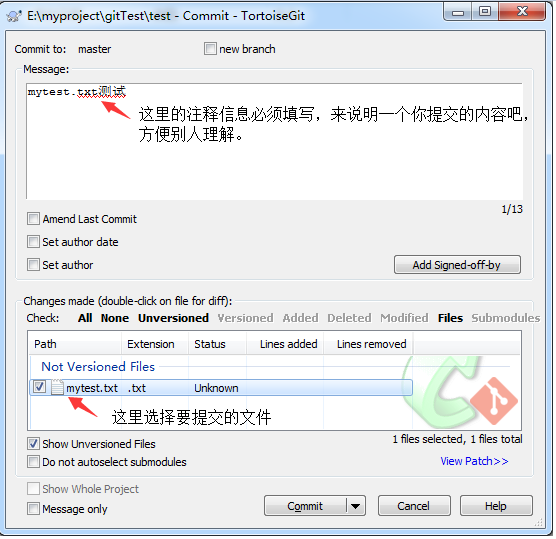
5、提交更改。

提交你的更改到工作目录树称为“提交”。提交时先使用Check for Modifications功能查看一下都有哪些文件发生了变化，如下图：





如果没有冲突，可以点击“Commit”按钮进行提交：



注意：

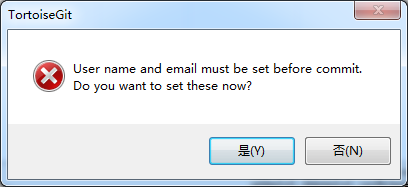
(1)message必须填写。

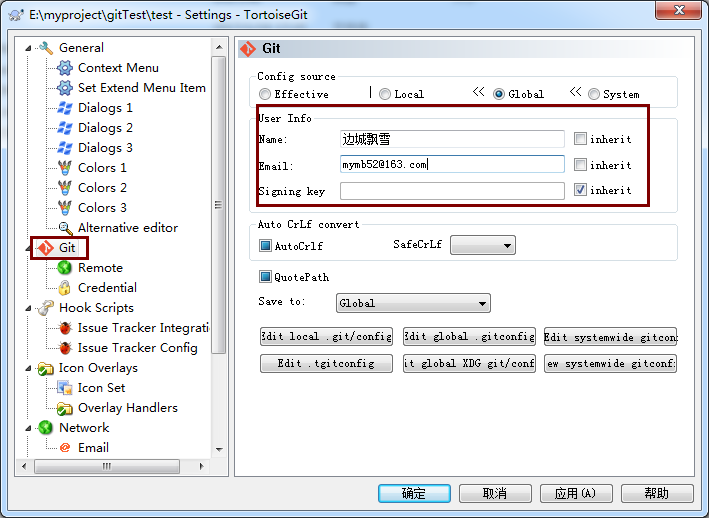
(2)在下面选择要提交的文件。

上图中下面的表格里列出了所有改变的文件和状态，如“Modified”表示文件修改过，双击这个文件，可以打开外部对比编辑器查看文件的改变内容。

Git要求注释必须要填写，这点和svn有点不同了，懒人们要发怒了哈哈！

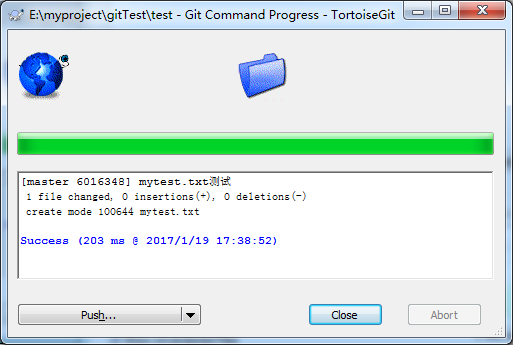
提交时如果提示email没有设置可以进行设置：





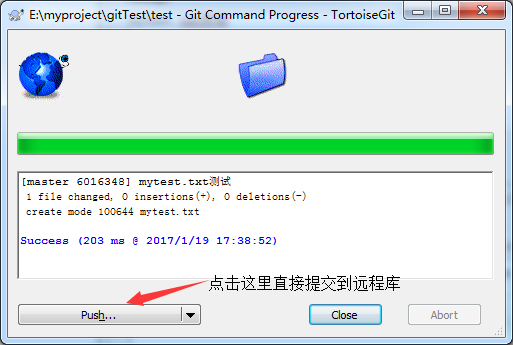
这里设置的信息将用来标识提交者。

提交成功后如下图所示：

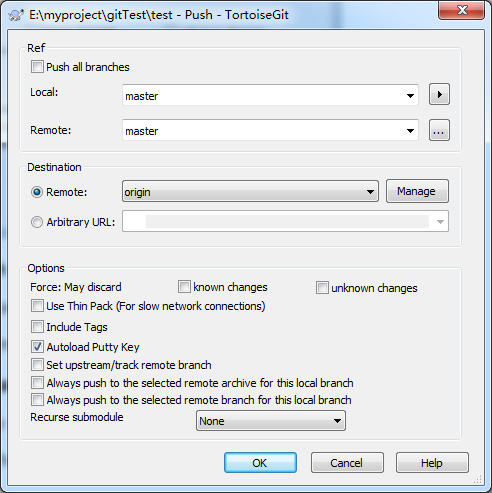


6、push推送变更到库

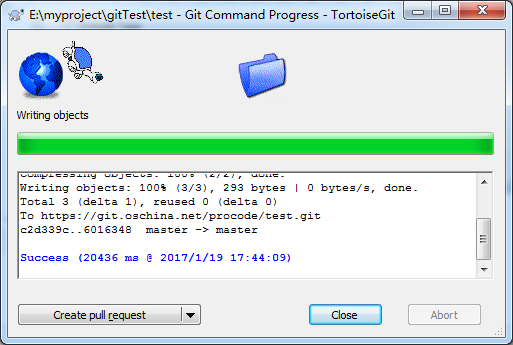
提交完成后可以直接提交到远程库：



推送界面如下：



Push完成：



7、从远程库更新

Pull 和 Fetch操作，这两个词不知道该翻译成什么好，先英文的呆着吧。

Pull 和 Fetch都是从库里获取数据，不同之处在于

Pull会获取远程修改并且与本地库合并。

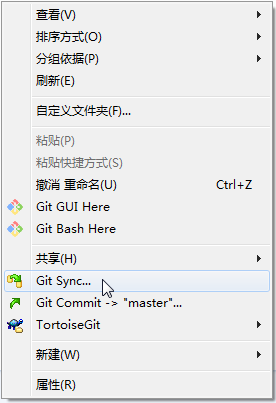
Fetch仅仅获取远程数据，而不执行合并操作。

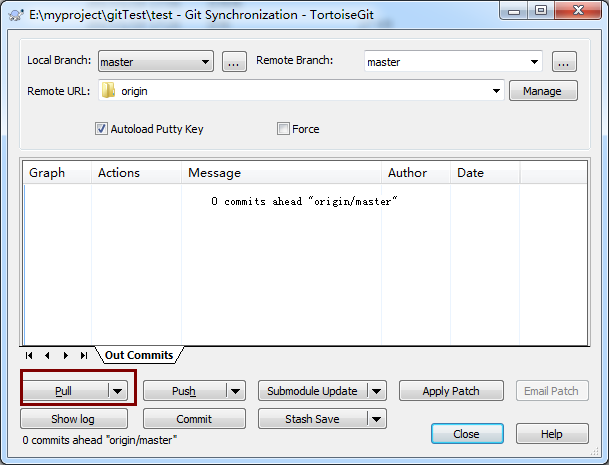
因此比较起来还是用Fetch更加安全点。

8、与远程库同步

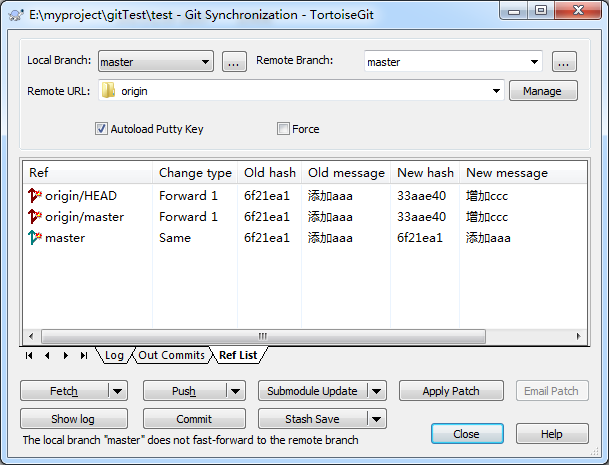
同远程库更新之前先与远程库进行一下同步是一个不错的习惯，可以先了解一下远程库和本地的所有改动。

在文件夹空白处点击右键，选择“Git Sync…”：

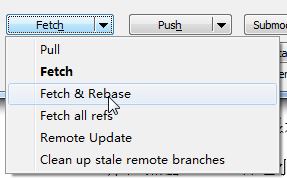




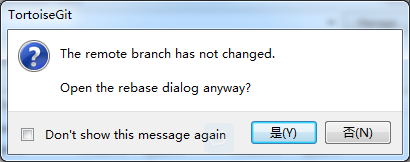
在同步窗口中点击“Pull”按钮边上的小三角，选择fetch用来获取远程数据，完成后如下图：



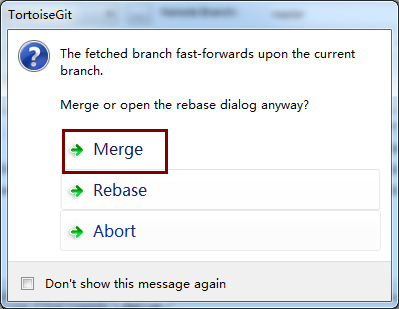
第一行和第二行显示远程库已经有了一次提交，你需要更新下来，点击fetch右边小三角，选择“Fetch&Rebase”，在接下来的窗口中直接点击“是”：



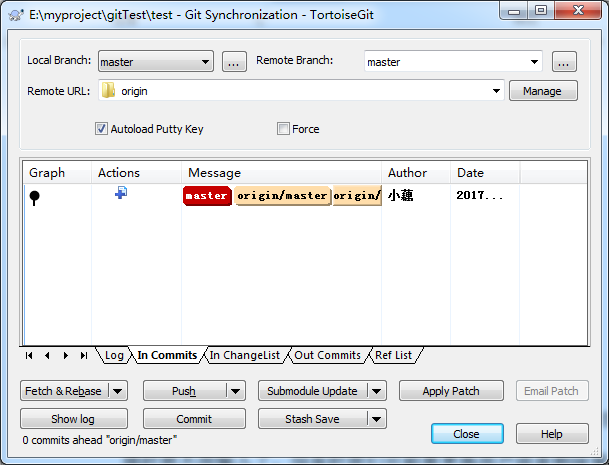
点击“是”：



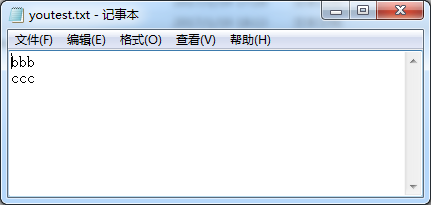
选择合并：



合并完成：



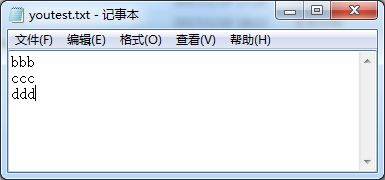
打开本地文件，发现对方的修改已经同步到本地：



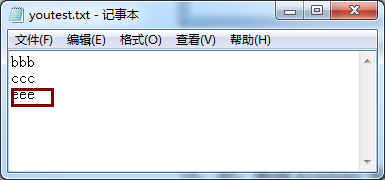
9、解决冲突

如果有幸两个人修改了同个文件，后提交的用户提交时将会产生冲突，冲突需要手动解决。

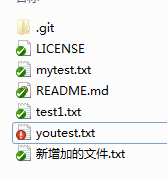
我们使用另一个用户在“yourtest.txt”增加“ddd”字符并提交到远程库：



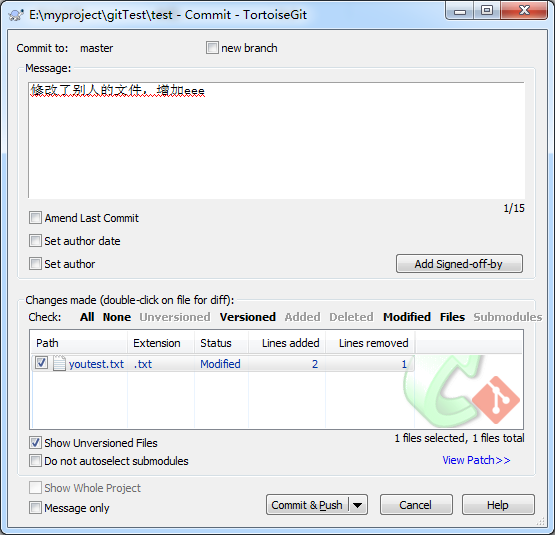
然后自己在文件中增加“eee”字符：



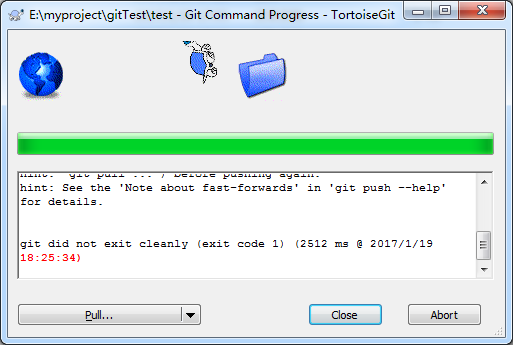
修改完成后本地文件图标变成下在的样子：



然后提交到本地库：

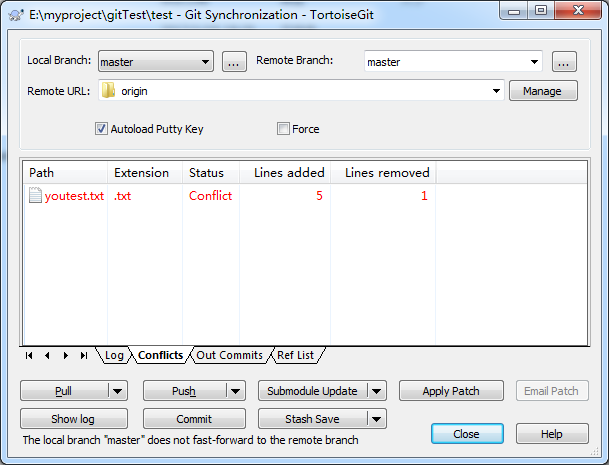


当我们提交到远程库时会出现错误：



点击Pull，将远程库下载到本地库，可在同步界面中点击“pull”按钮。

下图列出来了所有的冲突文件，双击冲突文件，可以对冲突进行编辑。



10、TortoiseGitMegar

解决冲突与对比文件使用TortoiseGitMegar工具，该工具可以完成以下工作：

（1）察看两个文件的差异，合并它们，或者删除不想要的修改。

（2）更新后，编辑本地修改和版本库修改之间的冲突。

（3）应用补丁或查看它们。

TortoiseMerge 有三种主要视图方式：单窗格，两窗格和三窗格。单窗格/两窗格视图用于察看修改，三窗格用来解决冲突。

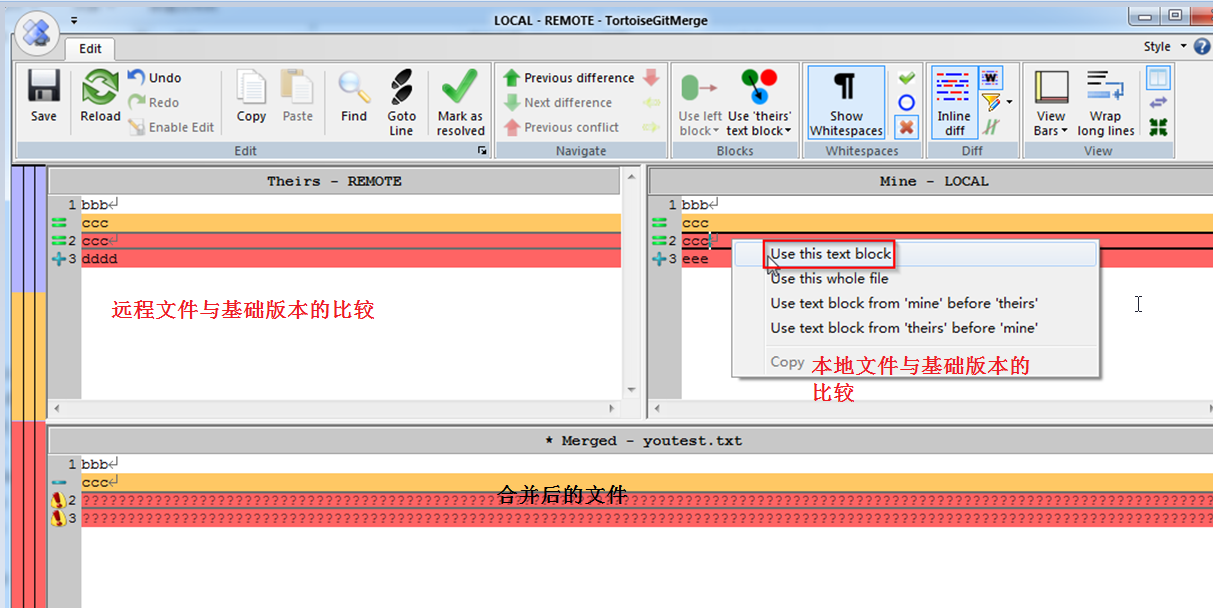
解决冲突界面分三个窗口，左窗口显示他人的文件与基础文件的差异，右窗口显示我的文件与基础文件的差异。底部窗口显示基础文件，我的文件，与 他人的文件 的合并结果，以及可能的冲突。

编辑时可以点击右键，在菜单中选择使用远程内容还是本工作区内容，也可以选择直接使用远程文件或本地工作区文件，还可以使用所有组合，不是他人的在我的之前，就是我的在他人的之前。

窗口中左侧显示行状态图标，已经修改的行用图标指示发生了什么类型的修改，图标意义如下：

* 已经增加一行。
* 已经删除一行。
* 此行曾经修改过，但当前和基础内容一样（已经通过恢复到原始内容而撤销修改）。
* 此行只有空白的修改。当有几行连续被标记时，此段可能被重新排版，从而造成修改。
* 此行已经使用 TortoiseMerge 作为文本编辑器手工修改。
* 此行处于冲突状态。
* 此行处于冲突状态，但是被空白或行结束符的设置掩盖。

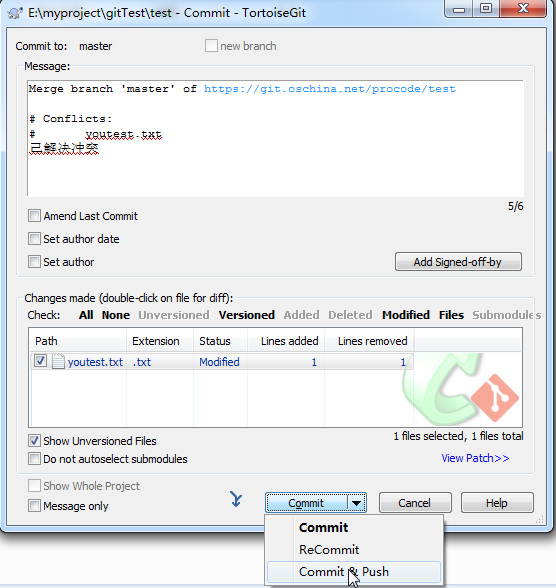
https://tortoisesvn.net/docs/nightly/TortoiseMerge_zh_CN/images/TMerge_linemoved.png 检测到该行已经移到其它位置或是从其它位置移过来（This line is detected as being moved from / to another location）。



编辑完毕后点击保存按钮进行保存，自动弹出是否解决冲突对话框：



点击标记为解决，将冲突标记为解决。

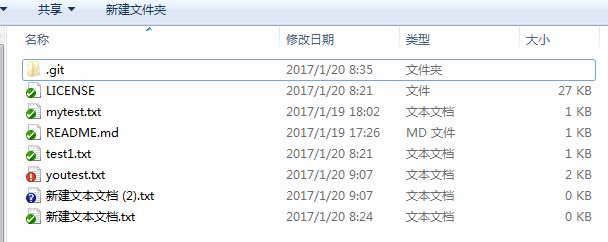


再次进行提交。

11、查看本地文件与基础文件的差异

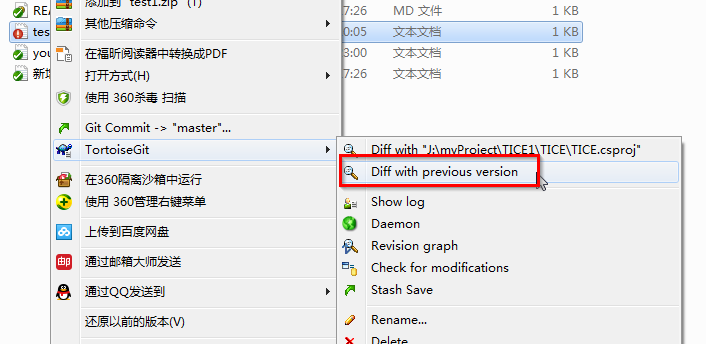
有时我们改了好多文件，但有一部分改错了想改回来或者想查看一下都作了哪些更改，可以使用此方法查看。

首先我们可以从文件（夹）图标上简单看出哪些文件进行了更改，如下图：

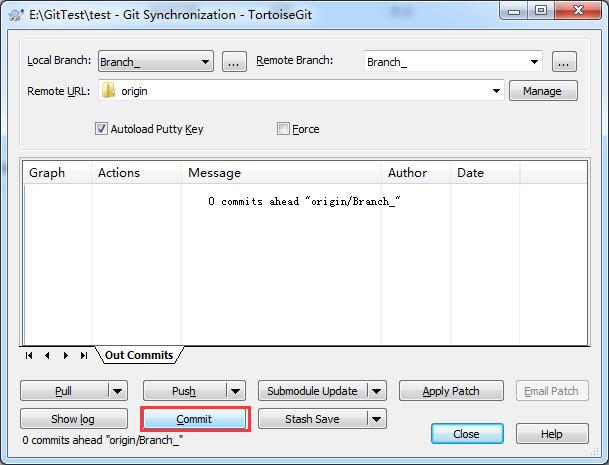


根据前面说的文件夹图标中的说明，可以看出“youtest.txt”文件进行了修改，“新建文本文档(2).txt”是一个新增的文件，其它的文件没有变化。

要查看单个文件的变更情况可以在文件上点击右键，在菜单中选择“Diff with previous version”以上一个版本进行比较：

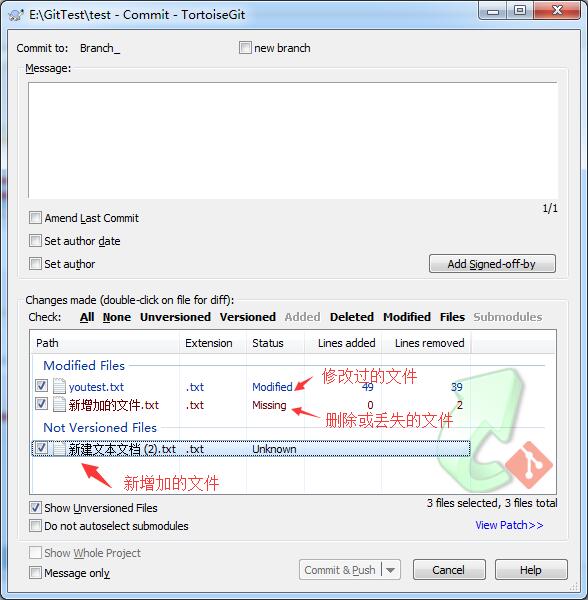


要查看更多文件变更的详细情况可以在文件夹空白处点击右键，在弹出的菜单中选择“Git Sync…”，弹出git同步窗口，如下图：



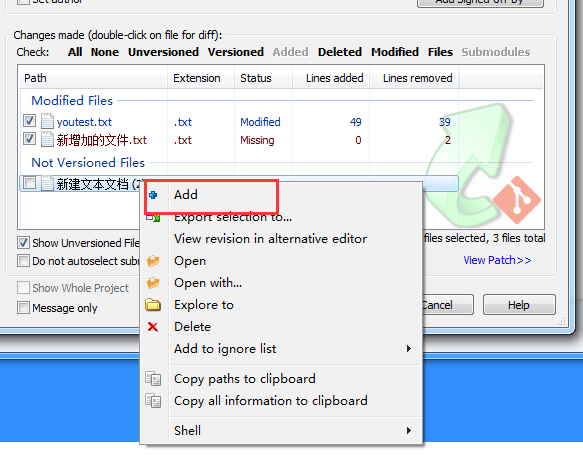
只比较工作区中修改的文件与本地库的差别可以直接点击“Commit”按钮，如果想比较与远程库的差异可以先进行Pull操作。

点击“Commit”按钮，进入提交界面，如下图，提交界面列出了所有文件的更改：

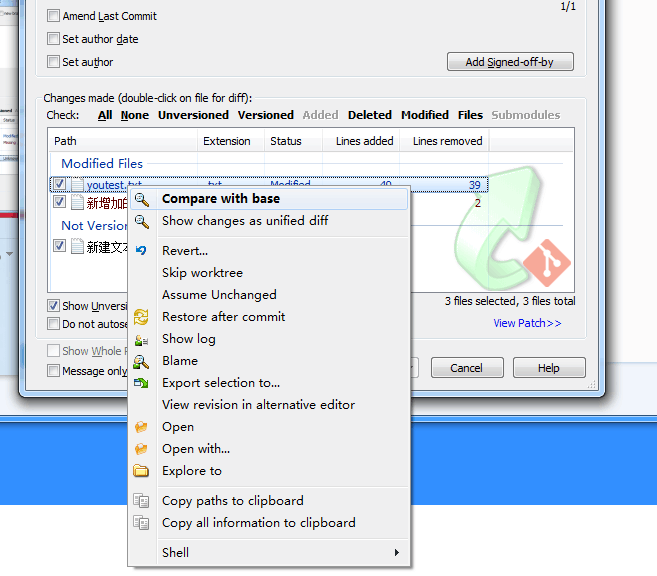


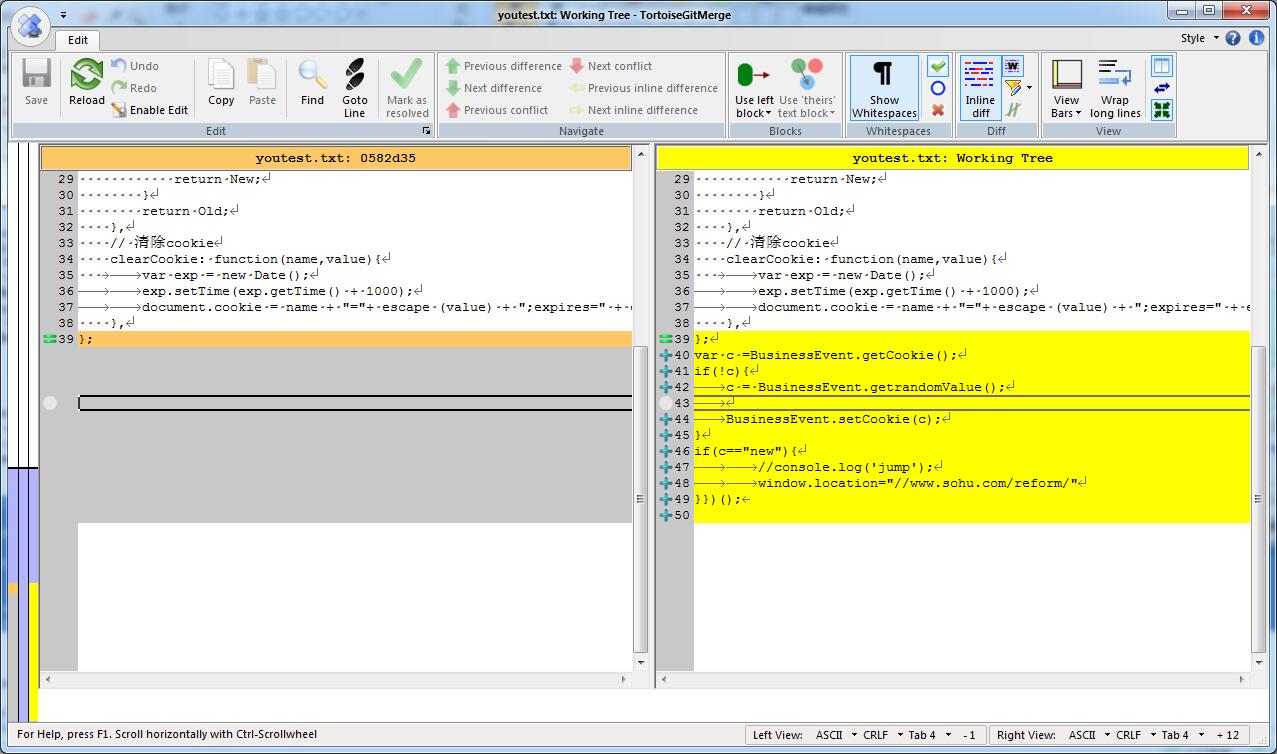
可以看到上面两行是修改过的文件，包括一个删除的文件（这个删除的文件直接在文件夹中查看不到，因此也无法用图标显示，但这里可以看到）。

将新增加的文件添加到版本库中可以在新文件上点击右键，选择“Add”，如下图：

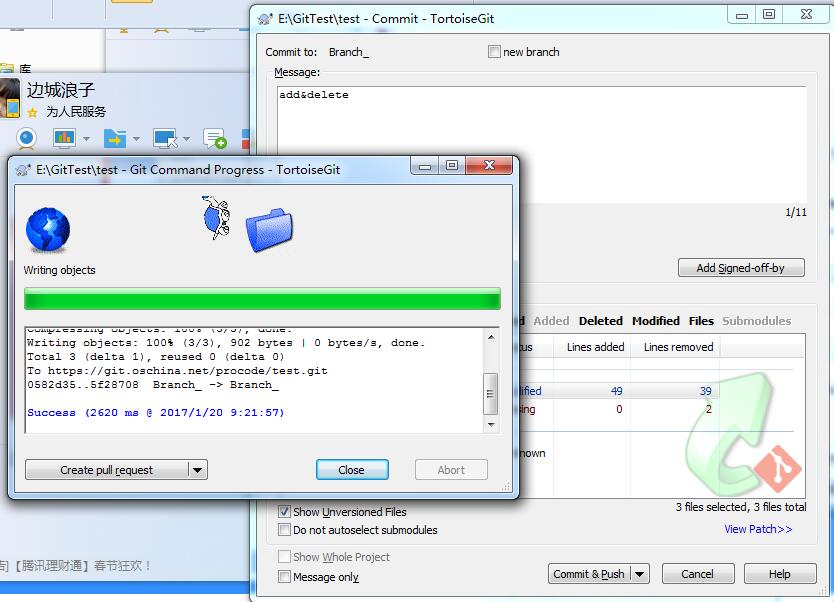


要查看新旧文件的差异，可以双击文件行或在相关文件行上点击右键，选择“Compare with base”，将打开TortoiseGitMegar工具，如下图：



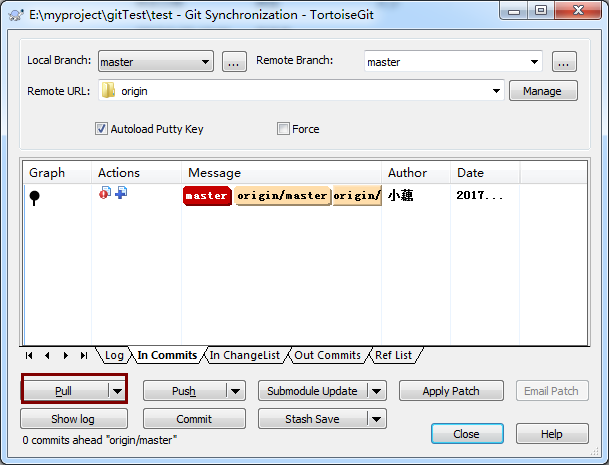


从这个程序中可以看出新增加了哪些内容。如果确认可提交的话可以直接点击下方的“Commit”或“Commit&Push”。

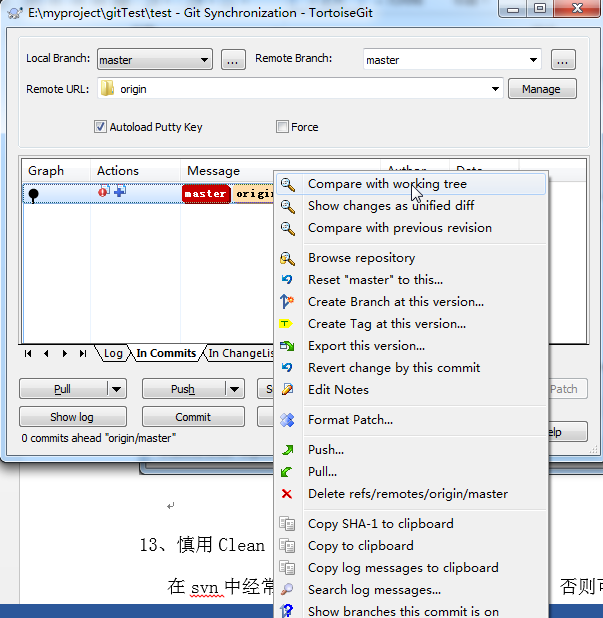


12、查看别人的修改

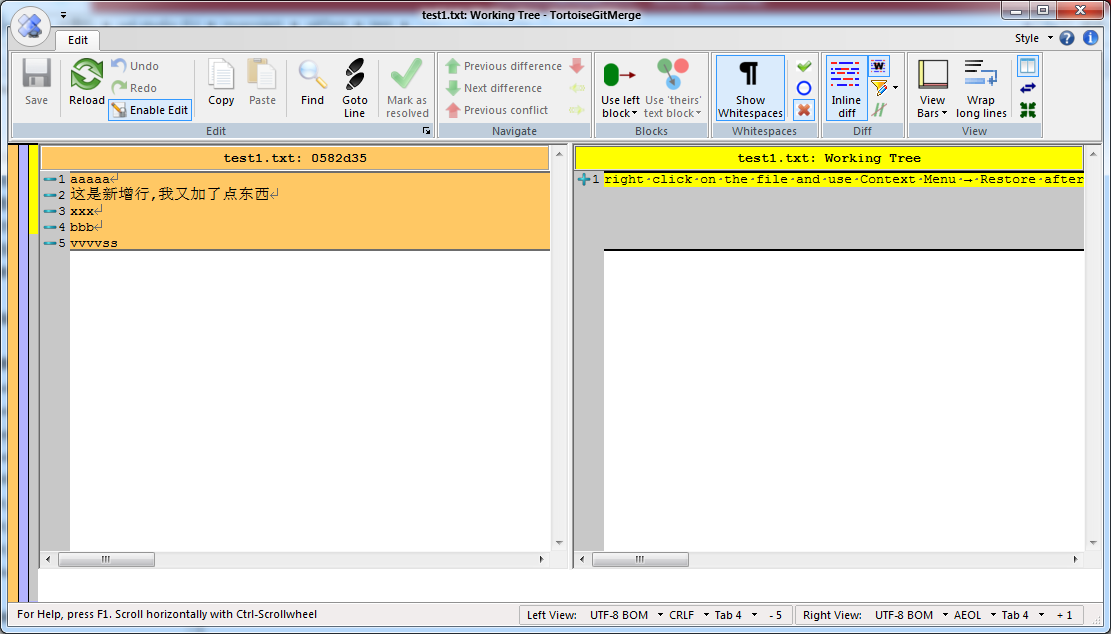
在同步窗口中点击“pull”按钮，将远程仓库的信息拉取到本地库，完成后界面如下：



在“inCommit”中，右键点击提交记录，选择“Compare with working tree”，进入比较界面。



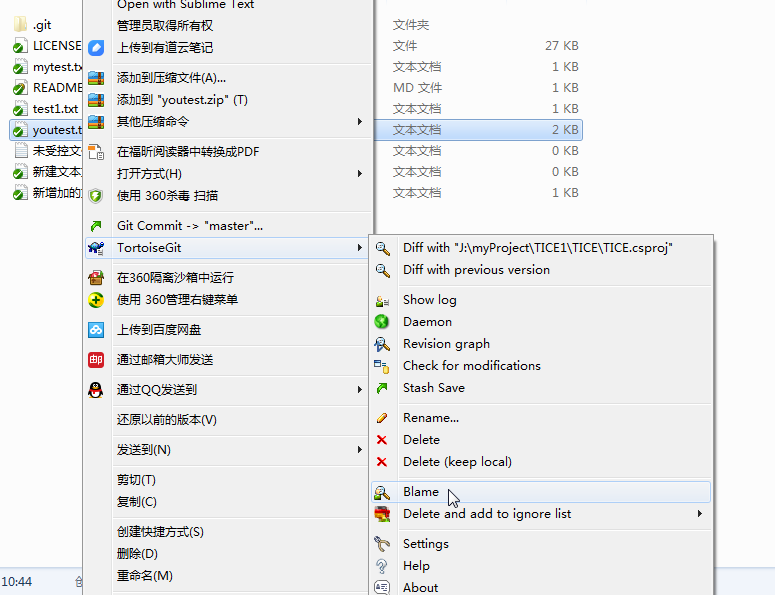
在比较界面中可以看到远程库中别人的修改，如下图（左侧为别人修改的，右侧为自己的文件）：

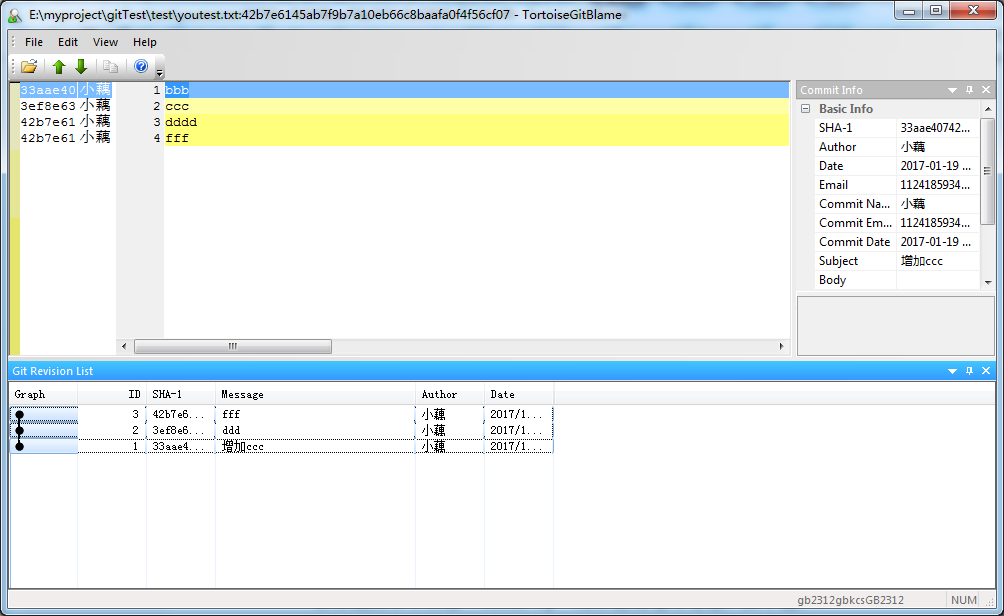


13、查看文件的所有修改者

根据需要，我们可以查看一个文件的所有更改历史。

在文件上右键菜单中点击“Blame”，弹出如下对话框：

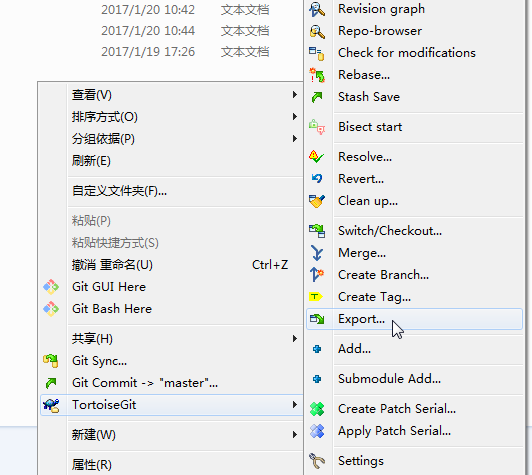


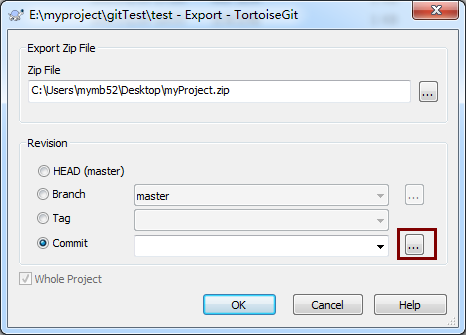


在这里不但可以看到修改了哪一行，而且可以看到谁修改的，是不是管理起来很方便。

14、导出某一版本的文件

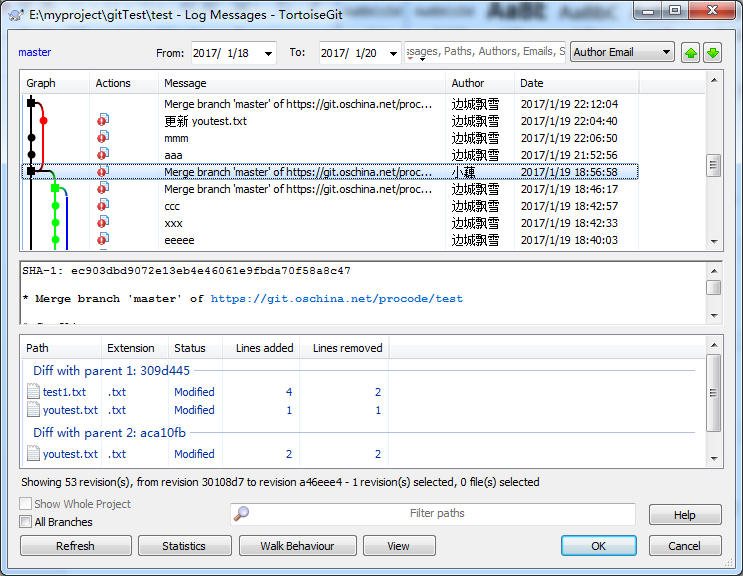
点击右键，在菜单中选择TortoiseGit → Export....：





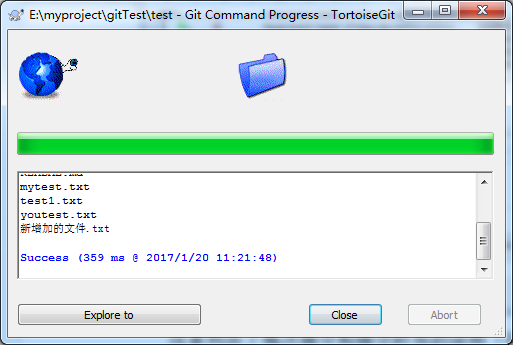
在这里你可以选择导出最新版本（主分支），也可以选择导出某一子分支，也可以导出打标记时的版本或某一次提交时的版本。

这里我们选择导出某一次提交时的版本，点击“commit”后面的“…”，弹出选择对话框：

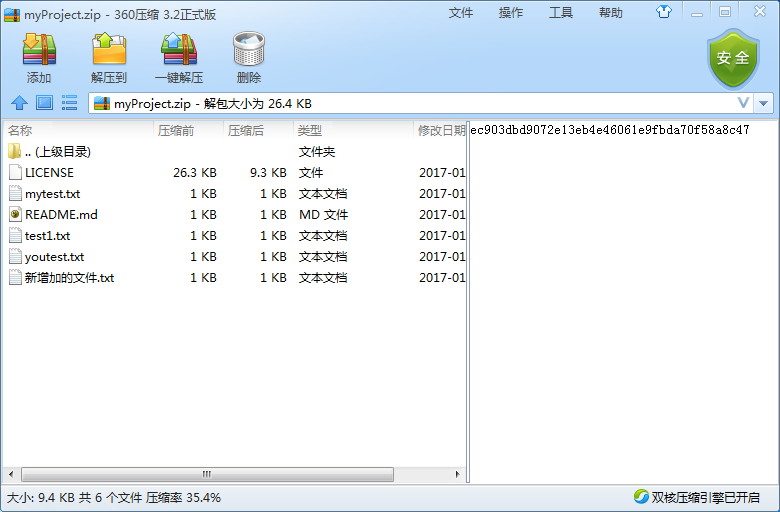


这里列出了每次提交和提交时写的说明（现在知道说明的重要性了吧），下面窗口则显示了本次提交涉及的文件。

选择一个版本，点击“ok”，点击上一界面的“ok”按钮，将该本文件导出。



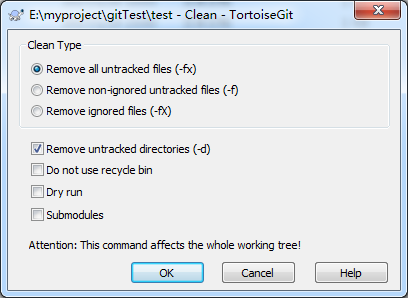
打开这个文件，是不是发现有一种回到了从前的感觉。



15、慎用Clean Up

在svn中经常用到的清理功能在git中要慎用，否则可能删除你还要保留的文件。

在右键菜单中点击“Clean Up”将弹出清理对话框，如下图：



请注意上面的三个选项：

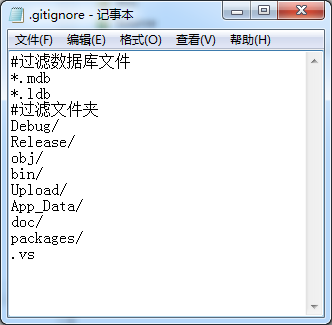
1. 移除所有未受版本控制的文件。
2. 移除所有未爱版本控制的文件（标记为ignore的除外）。
3. 移除标记为ignore的文件。

这三个选项都会删除相应的文件，请小心使用。

16、.gitignore和不需提交的文件

.gitignore 配置文件用于配置不需要加入版本控制的文件，配置好该文件可以为我们的版本管理带来很大的便利。

.开头的文件在windows下认为只有扩展名而没有文件名，所以不能直接建立该文件，可以使用记事本新建一个文件，另存为“.gitignore”文件，文件内容如下：



这是一个.net开发环境下的一个配置，表示所有的mdb文件和ldb文件不加入版本库；所有Debug文件夹下的文件、Release文件夹下的文件、obj文件夹下的、bin文件夹下的、Upload文件夹下的、App\_Data文件夹下的、doc文件夹下的、packages文件夹下的以及.vs文件夹不加入版本控制

配置语法：

以#号开头为注释。

　以斜杠“/”开头表示目录。

　以星号“\*”通配多个字符。

　以问号“?”通配单个字符。

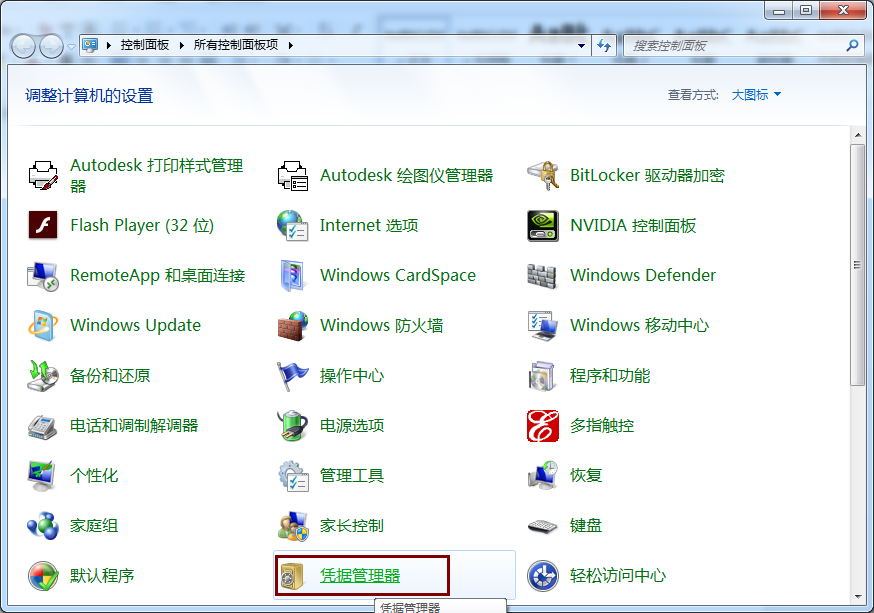
　以方括号“[]”包含单个字符的匹配列表。

以叹号“!”表示不忽略(跟踪)匹配到的文件或目录；

17、附：删除windows凭据

我们在第一次连接git库的时候输入的用户名和密码，后面再连接时就不用输入了，但有时我们可能要更换用户或者删除曾经使用过的用户名或密码该怎么操作呢？

在控制面板中点击“凭据管理”：



该文档为本人原创文档，转载请注明！

边城浪子 二O一七年元月二十日