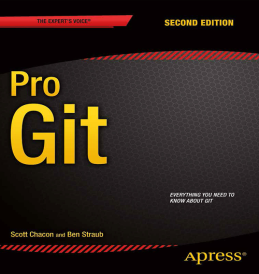
Git 免费书籍Pro Git **中文版** <https://git-scm.com/book/zh/v1>



Windows上安装 git

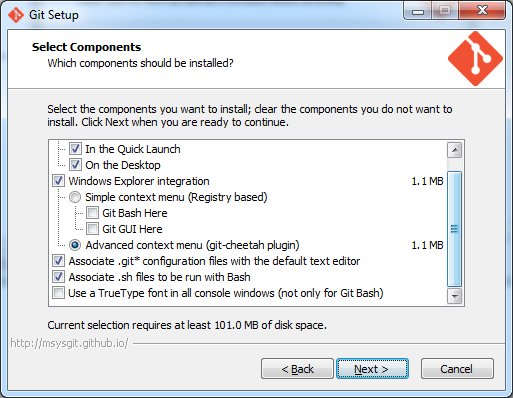
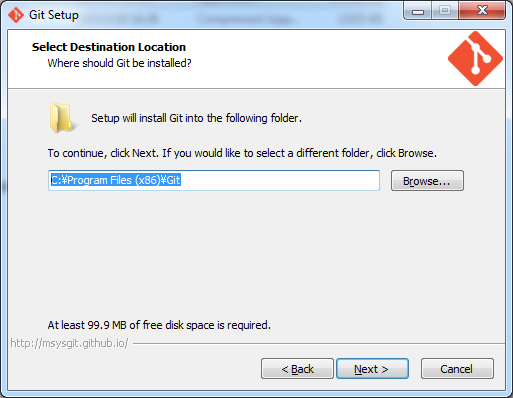
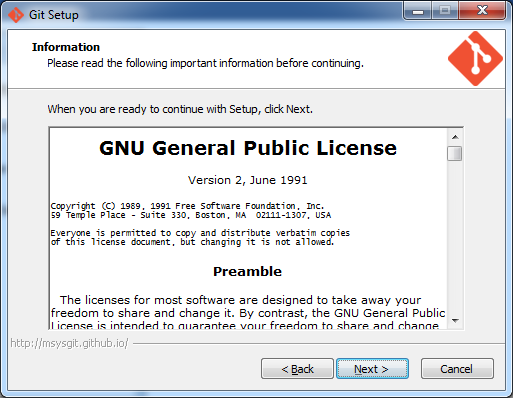
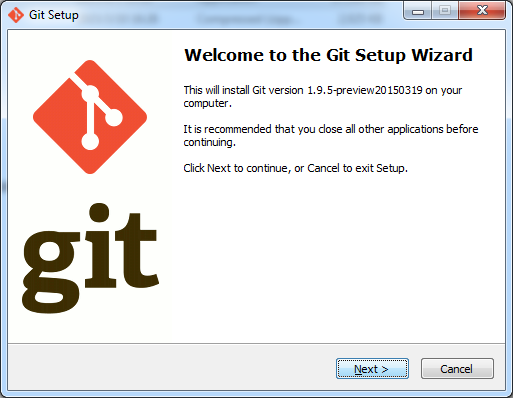
从<http://msysgit.github.io/> 下载Git-1.9.5-preview20150319.exe git安装包 用管理员身份去安装

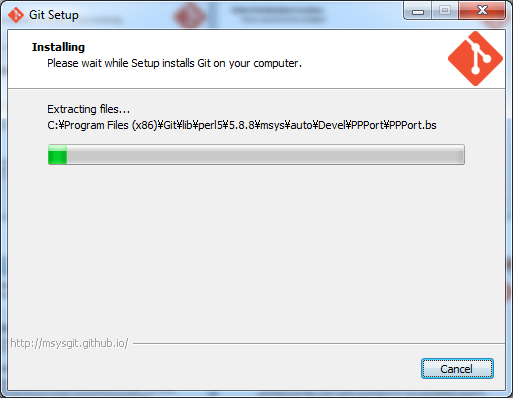
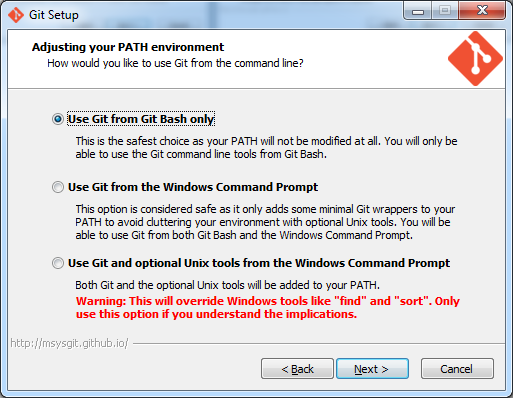
msysGit自带命令行工具以及图形界面工具。

或者从<https://www.sourcetreeapp.com/> 下载 SourceTreeSetup\_1.6.14.exe， 这是一个免费的git图形界面工具。

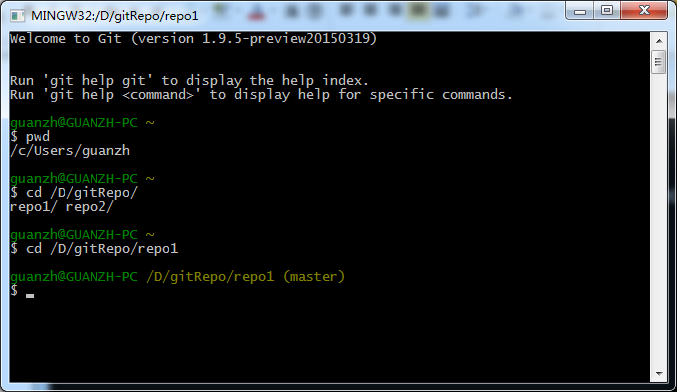
使用内嵌的Git安装包 或者可以指定其他版本Git的安装包。





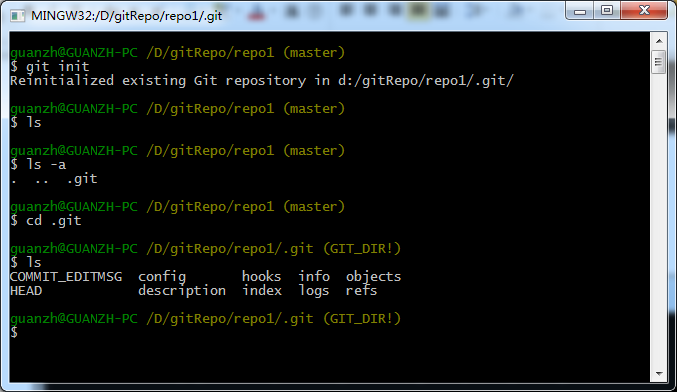


在桌面或者开始菜单中打开 然后移动至这次的一个工作目录D:\gitrepo\repo1



**先看一些简单的命令。稍微熟悉一下Git**

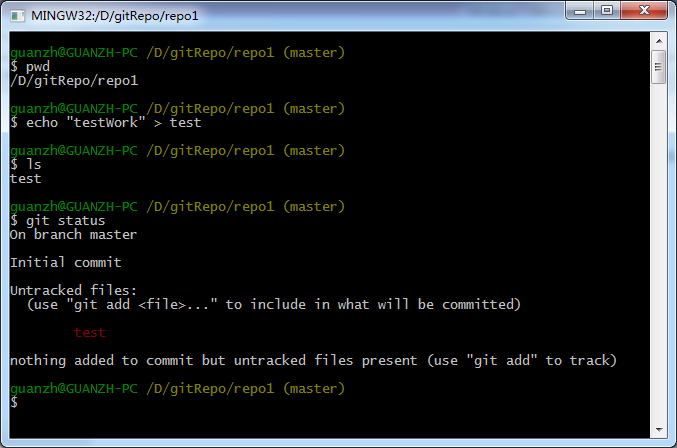
**初始化Git仓库命令 git init**



初始化完了以后，我们在这个目录下就建立了一个Git的本地仓库，这个目录中的文件就可以做版本管理了。

版本管理的信息会放在这个目录中的隐藏目录.git下。

创建一个新文件。此文件还没有被git管理。就类似在使用SVN的时候的新建文件。



查看Git仓库的状态 命令 git status

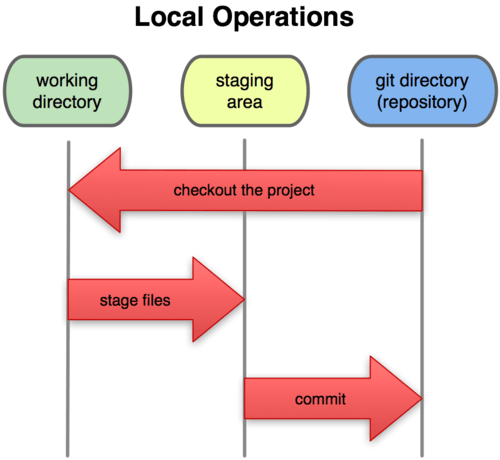
上面的操作创建了一个文件，然后查看了一下这个文件的状态为 【未跟踪】（untracked）

然后可以通过一些其他的命定进行commit，diff ，reset等操作。

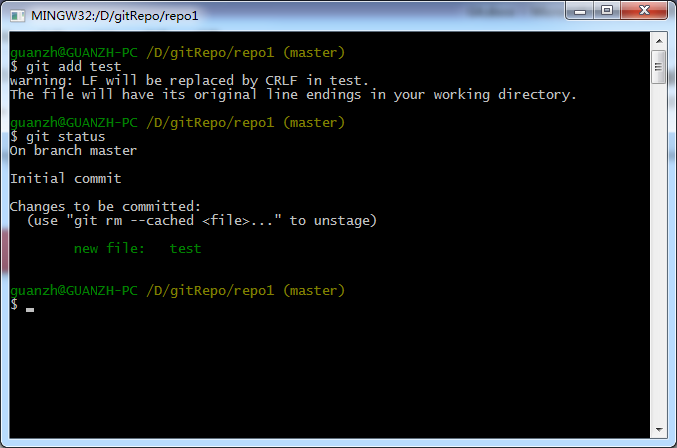
在继续之前，需要介绍几个Git的基本概念

一个GIT仓库分为 3 个区，分别为：【工作区】，【暂存区】，【本地仓库】。

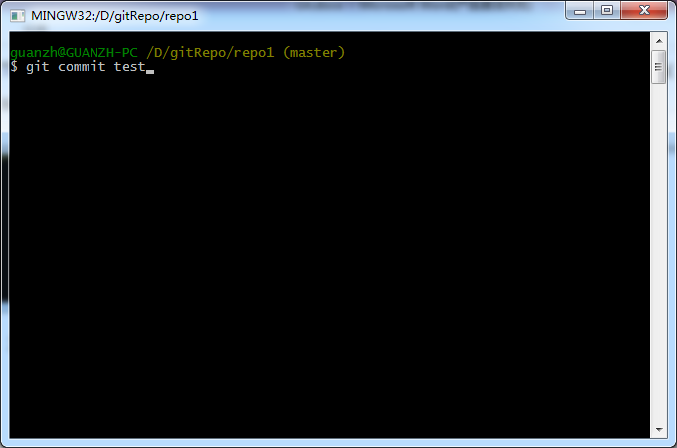
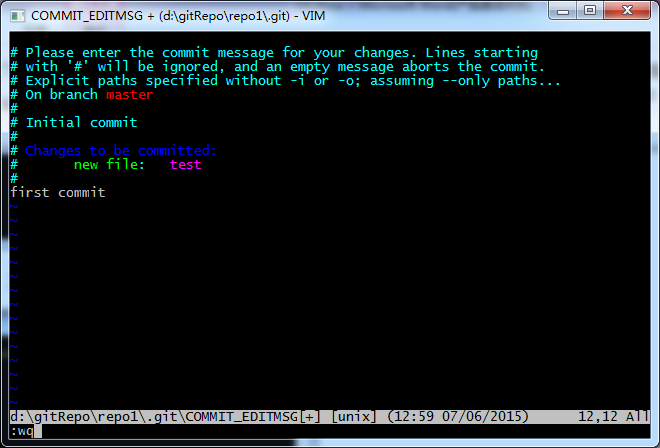
暂存区目的是为了在提交前做一次最后确认的区域。

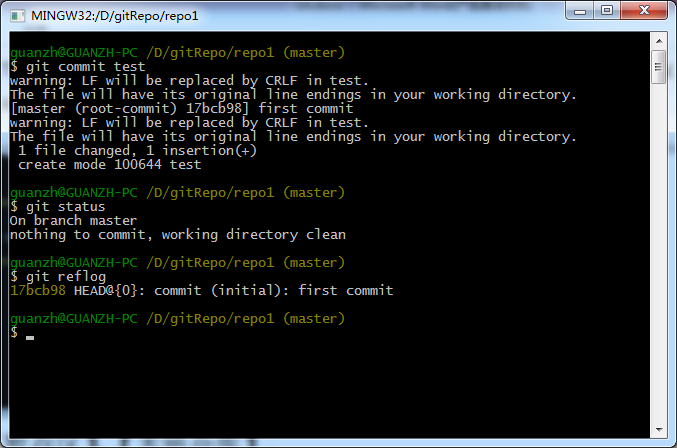


使用 git add <file> 将文件从【工作区】保存到【暂存区】



使用 git commit <file> 将文件从【暂存区】保存到【本地仓库】

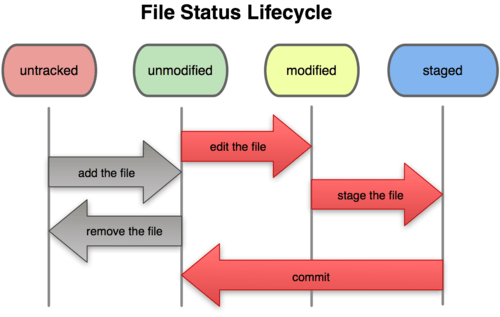


这时候，【工作区】，【暂存区】，【本地仓库】的文件都一样。

可以通过git reflog 查看提交的版本号。

在文件版本管理的过程中，文件会有一些状态，结合官方给出的图做一个说明

这个图个人感觉有点晦涩。



如果你觉得不想太费劲可以不用看下面的内容，直接跳到红字部分看结论

【未跟踪】（untracked）特指【工作区】里新建的文件的状态。

【已跟踪】（tracked）指【未跟踪】以外的所有状态的统称。

这两个状态了解了以后，后面我们将不再使用这两个状态来描述。免得状态太多太繁琐。

对于【工作区】里的文件，有【未暂存】【已暂存】【未修改】【已修改】状态

【未暂存】 意思是【工作区】里新建了一个文件。或者【工作区】里的文件和【暂存区】里的文件不一致。

【已暂存】 意思是【工作区】里的文件已经保存到【暂存区】了。并且两边的文件一致。

【未修改】 意思是【工作区】里的文件和【暂存区】里的文件一致。

【已修改】 意思是【工作区】里的文件和【暂存区】里的文件不一致。

因为【未修改】已经被包含在【已暂存】中了，【已修改】已经被包含在【未暂存】中了。

所以【工作区】里的文件，我们只保留【未暂存】和【已暂存】两个状态来描述。

对于【暂存区】里的文件，有【未提交】【已提交】状态

【未提交】 意思是【暂存区】里的文件还一次还未提交到【本地存储区】或者和【本地存储区】里的文件不一致。

【已提交】 意思是【暂存区】里的文件和【本地存储区】里的文件一致。

所以简化后，我们讨论的状态就是

【未暂存】，【已暂存】，【未提交】，【已提交】

情况A

1. 一个新建的文件test.txt的状态叫【未暂存】（unstaged），位置在【工作区】,目前并没有做版本管理
2. 通过add命令可以将test.txt变为【已暂存】（staged）

此文件将在【工作区】和【暂存区】同时存在。两处的文件一样

1. 通过commit命令可以将【暂存区】test.txt【未提交】（uncommitted）提交到【本地存储区】变为【已提交】（committed）

此文件将在【工作区】和【暂存区】和【本地存储区】同时存在。三处的文件一样

情况B

1. 同情况A
2. 同情况A
3. 对test.txt进行修改，这时候【工作区】和【暂存区】的文件内容已经不一样。工作区的test.txt状态变为

【未暂存】（unstaged）

1. 通过commit命令可以将【暂存区】test.txt【未提交】（uncommitted）提交到【本地存储区】变为【已提交】（committed）

此文件将在【工作区】和【暂存区】和【本地存储区】同时存在。

【暂存区】和【本地存储区】的文件一样。

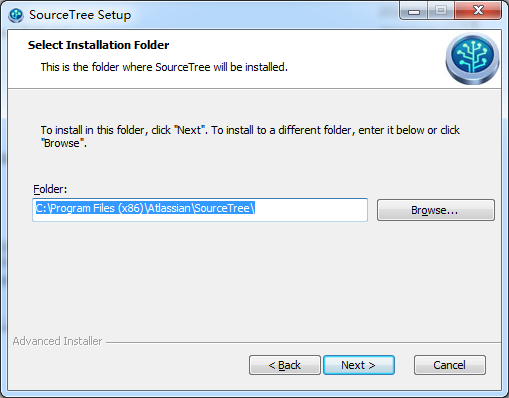
【工作区】的文件还是【未暂存】（unstaged）

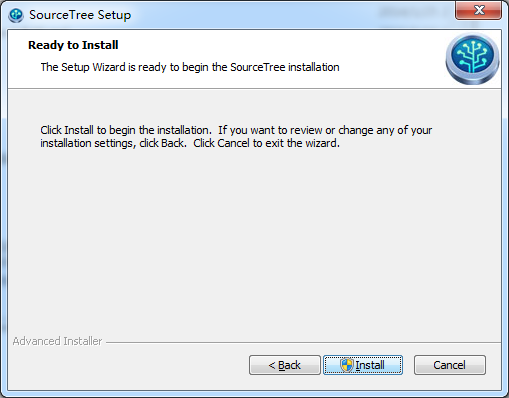
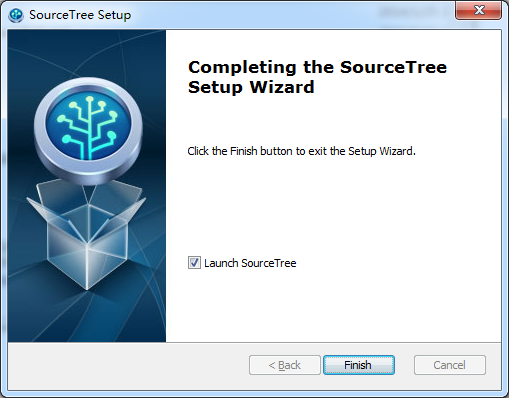
至此，我们已经对Git的命令行有了一个简单的了解。

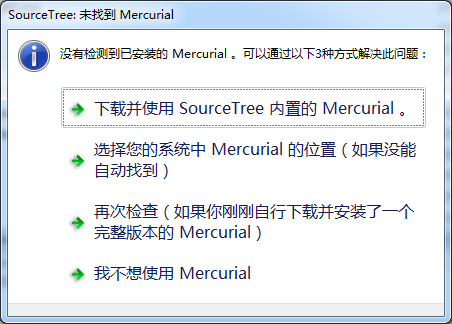
Git 命令很多，而且每个命令后面有很多参数可以选择。对于普通用户来说有点过于复杂。

所以对于普通用户来说，使用图形管理工具会更方便一些。

安装 SourceTree

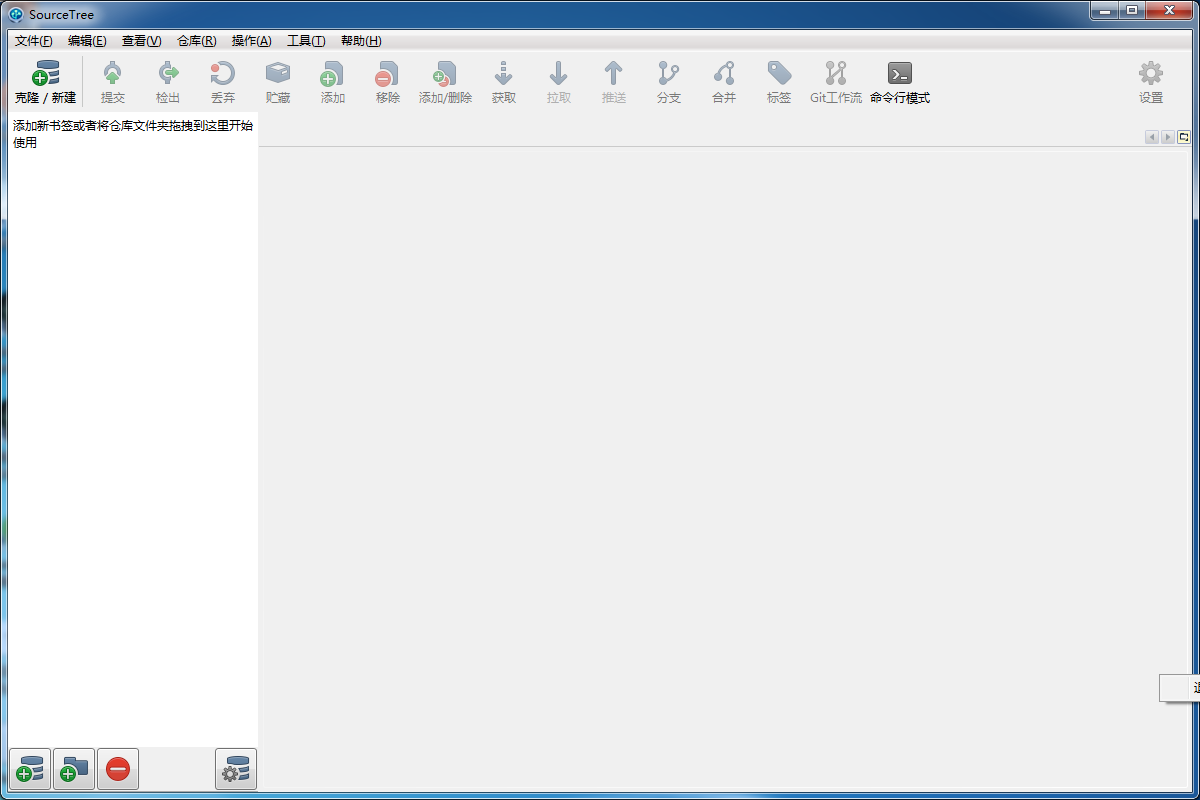
 

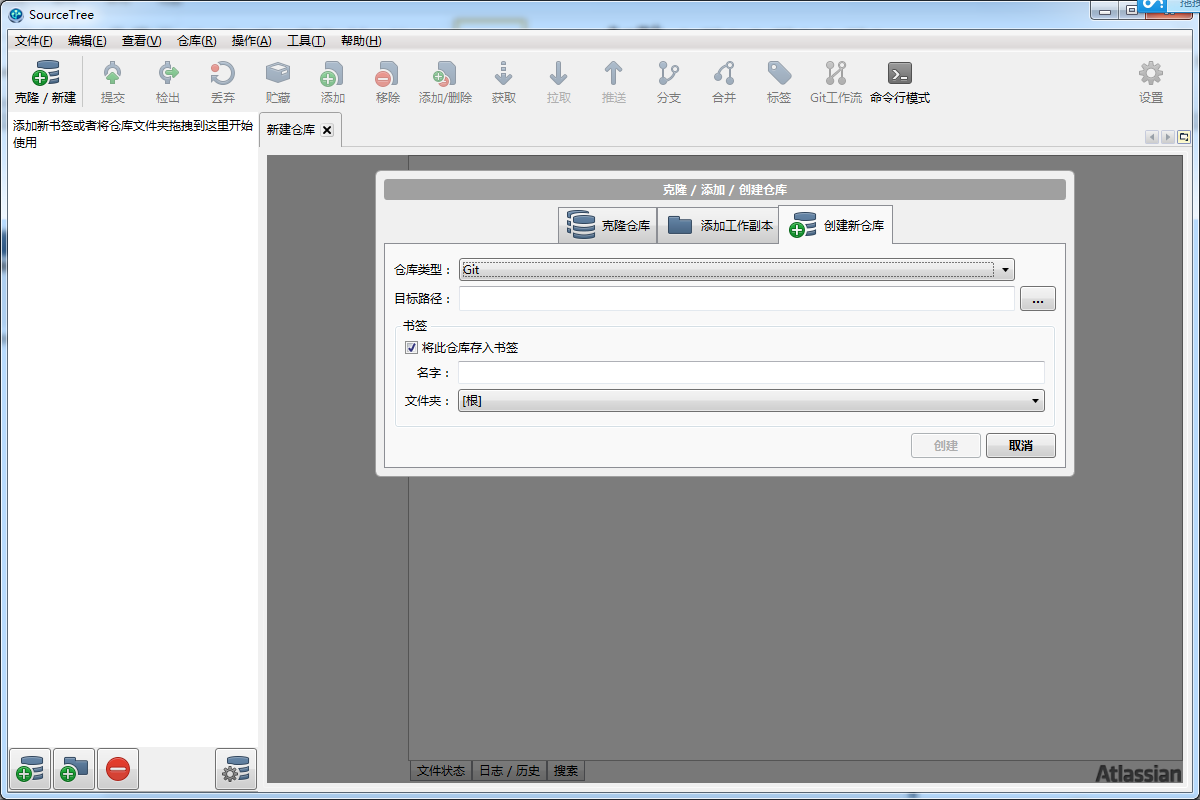


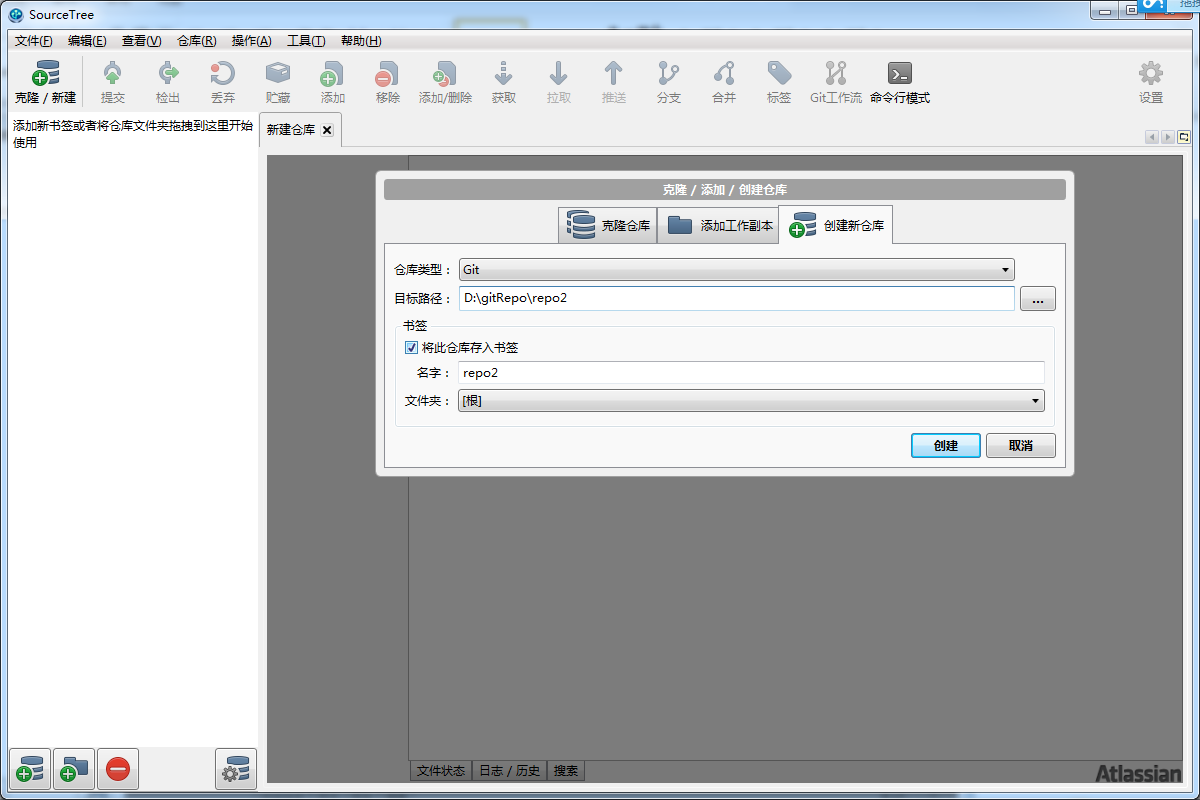
Mercurial是和Git类似的一个版本管理工具，我们这次使用Git，不使用Mercurial所以选最后一个。

因为我们已经安装过msysGit,所以sourceTree会自动监测到。如果没有安装过任意版本的Git，那么也会弹出一个没有监测到已安装Git的对话框。



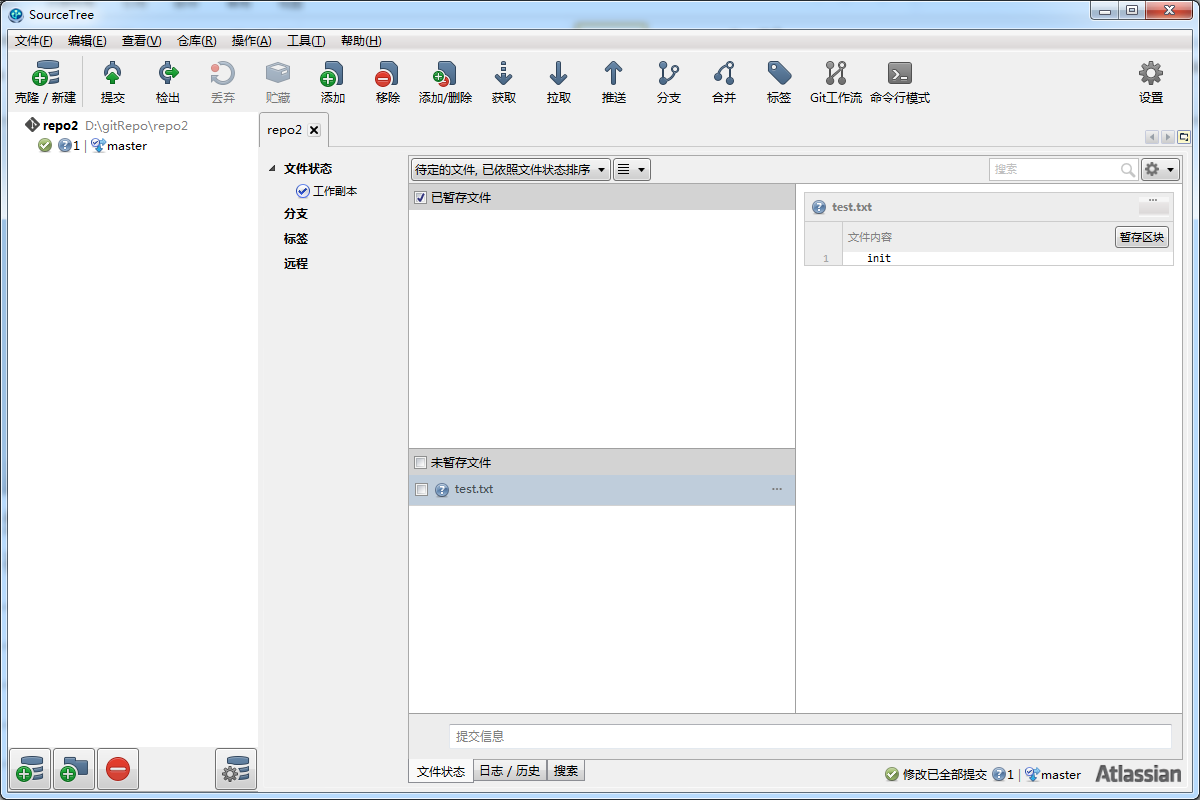
新建





然后创建，这实际上就相当于在命令行执行了git init

在 D:\gitRepo\repo2 下创建一个文件。Test.txt 内容为 init, 创建完成后可以在sourceTree中看到效果。



点击提交

提交文件

点击checkbox

取消暂存

点击checkbox

取消暂存

已暂存文件内容与本地存储区的内容比较

点击按钮

完成暂存

点击checkbox

完成暂存

1个未暂存文件

没有已暂存文件

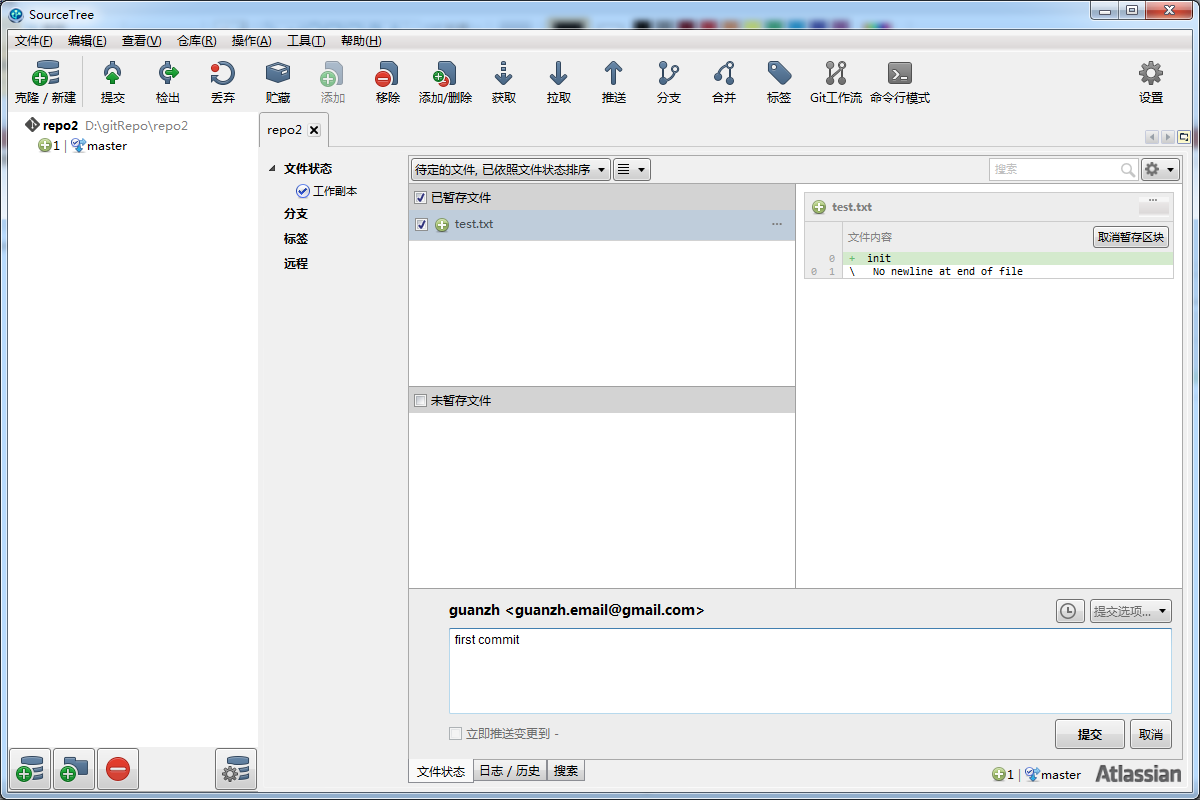
未暂存文件内容



仓库名称，仓库位置



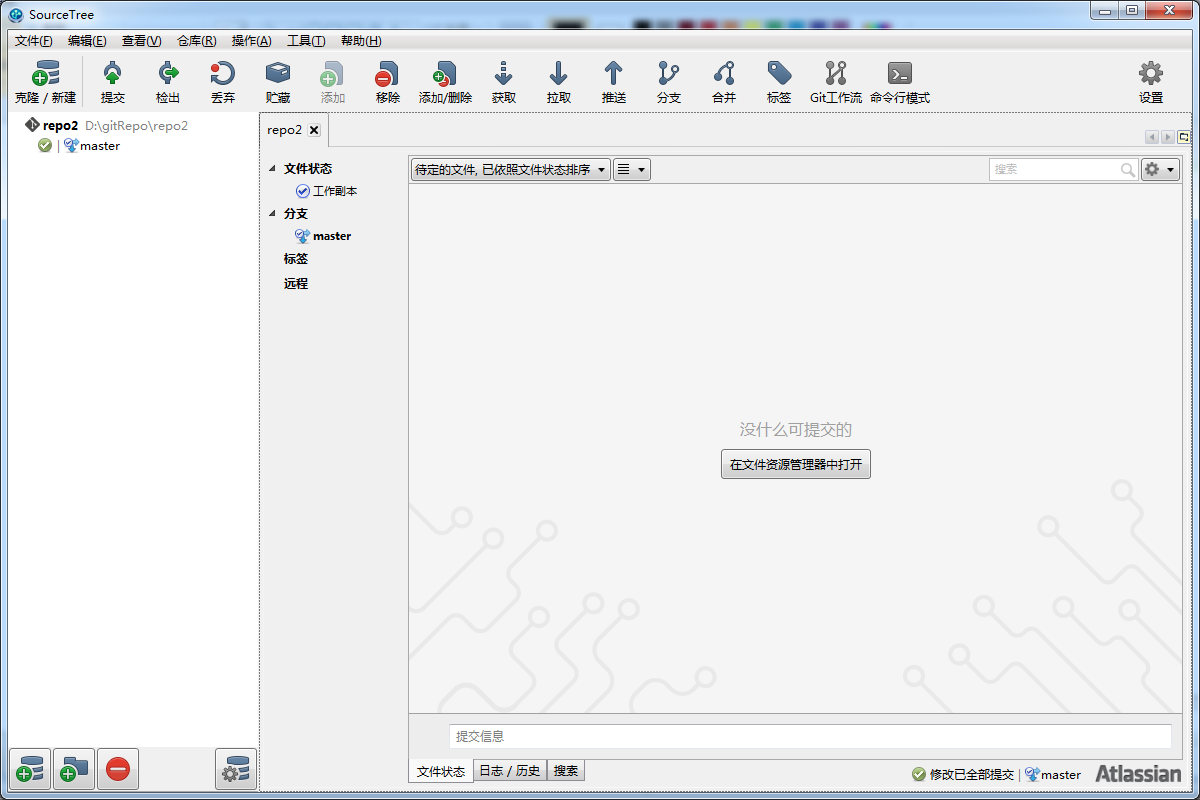
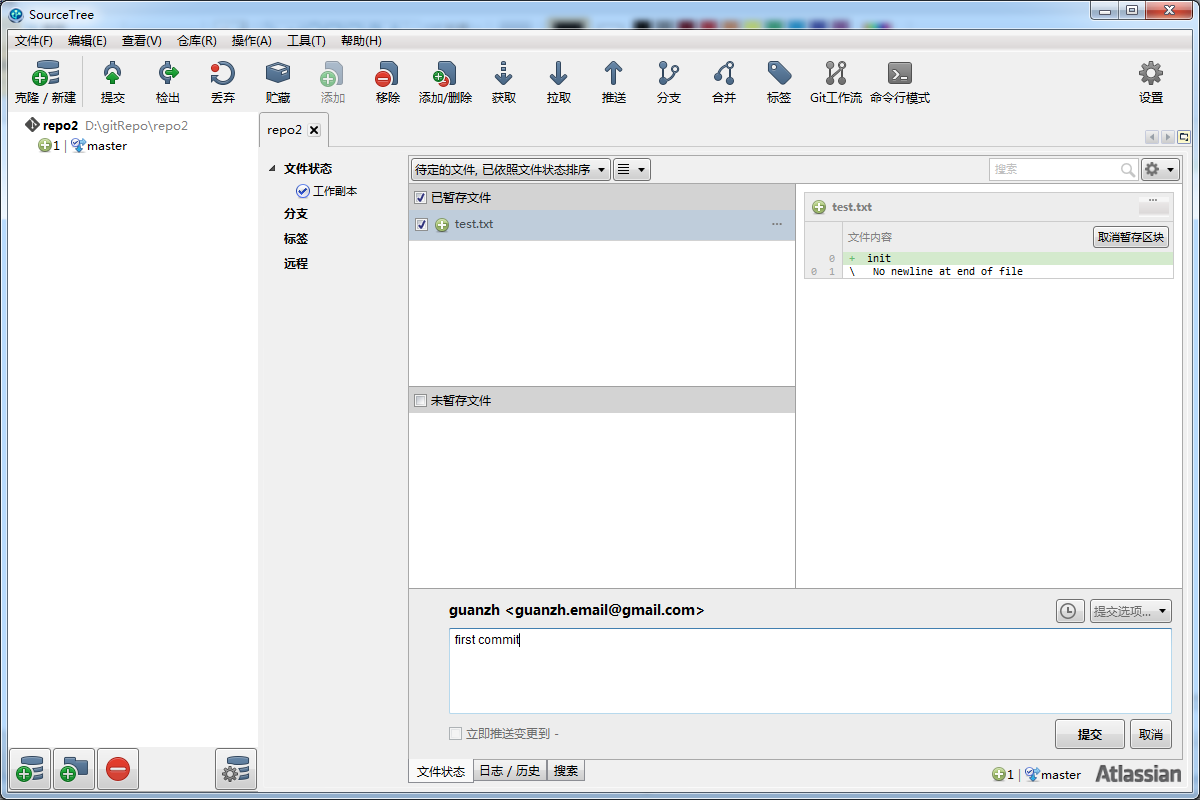
当前的分支为Master ， 1个未暂存文件



1个已暂存文件

没有未暂存文件

1个已暂存文件



点击提交

提交文件

填写提交信息

点击提交

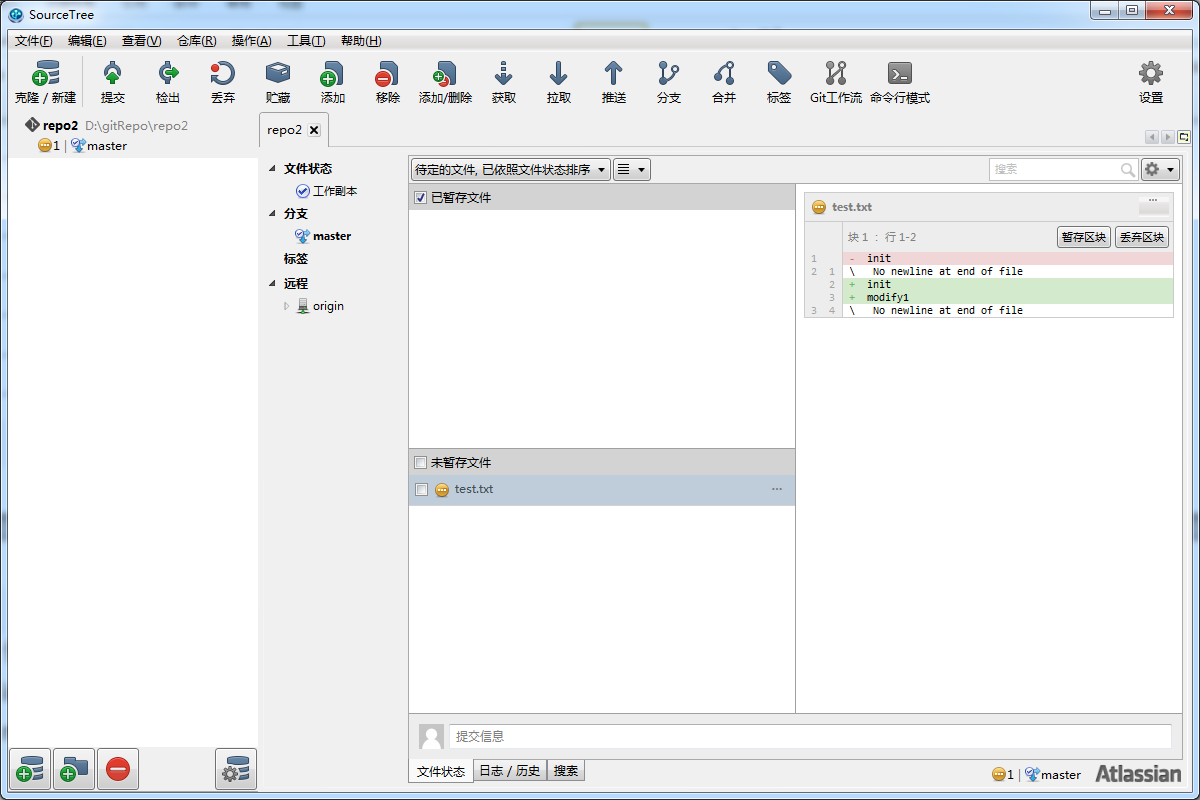
提交文件

提交完毕

修改本地文件

init

modify1

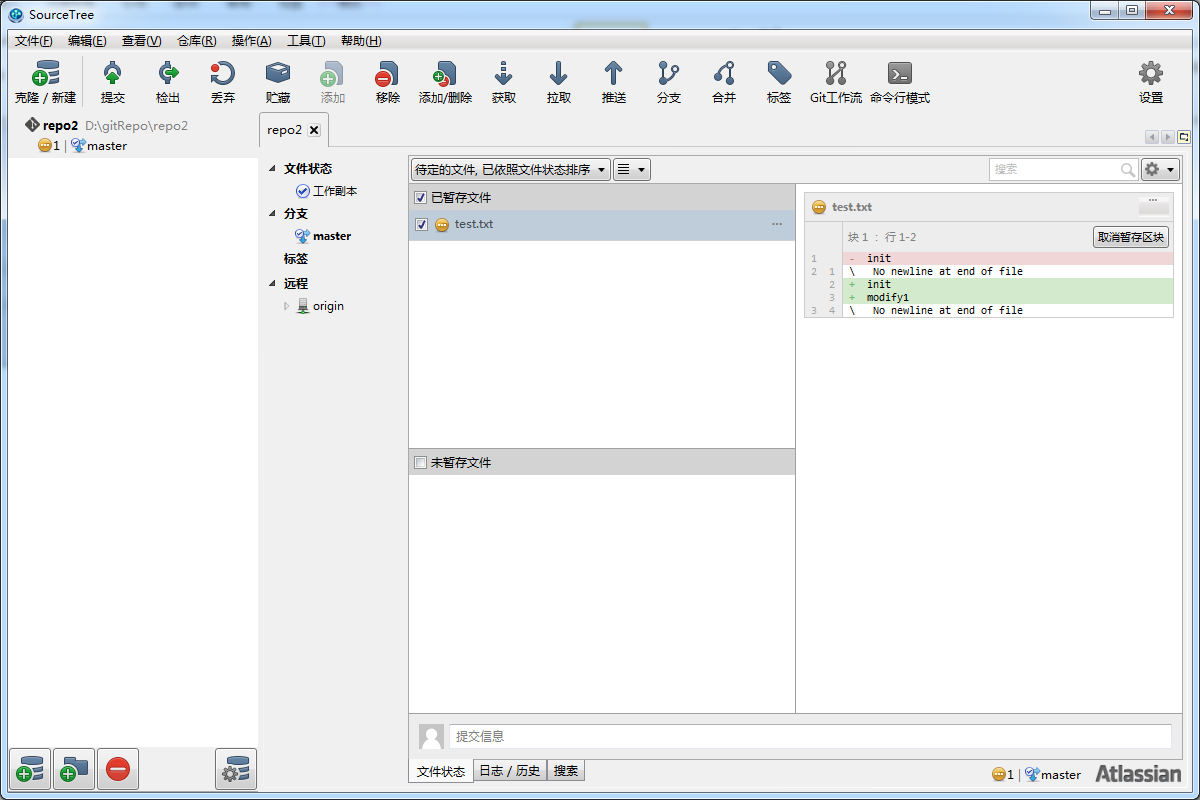


工作区文件和已暂存文件

内容比较

工作区1个文件已修改

1个文件已修改

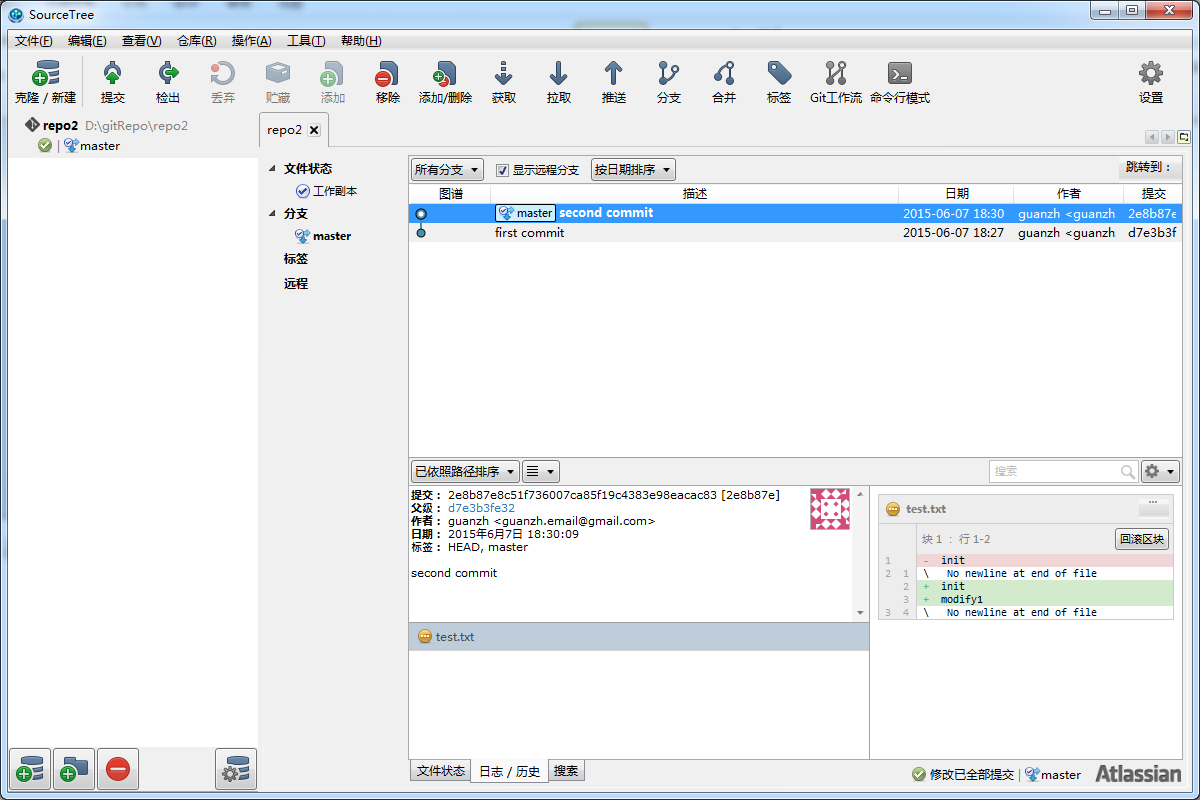


暂存文件和本地存储区文件

内容比较

暂存区1个文件已修改

再次commit, 查看履历



HEAD表示当前处理对象

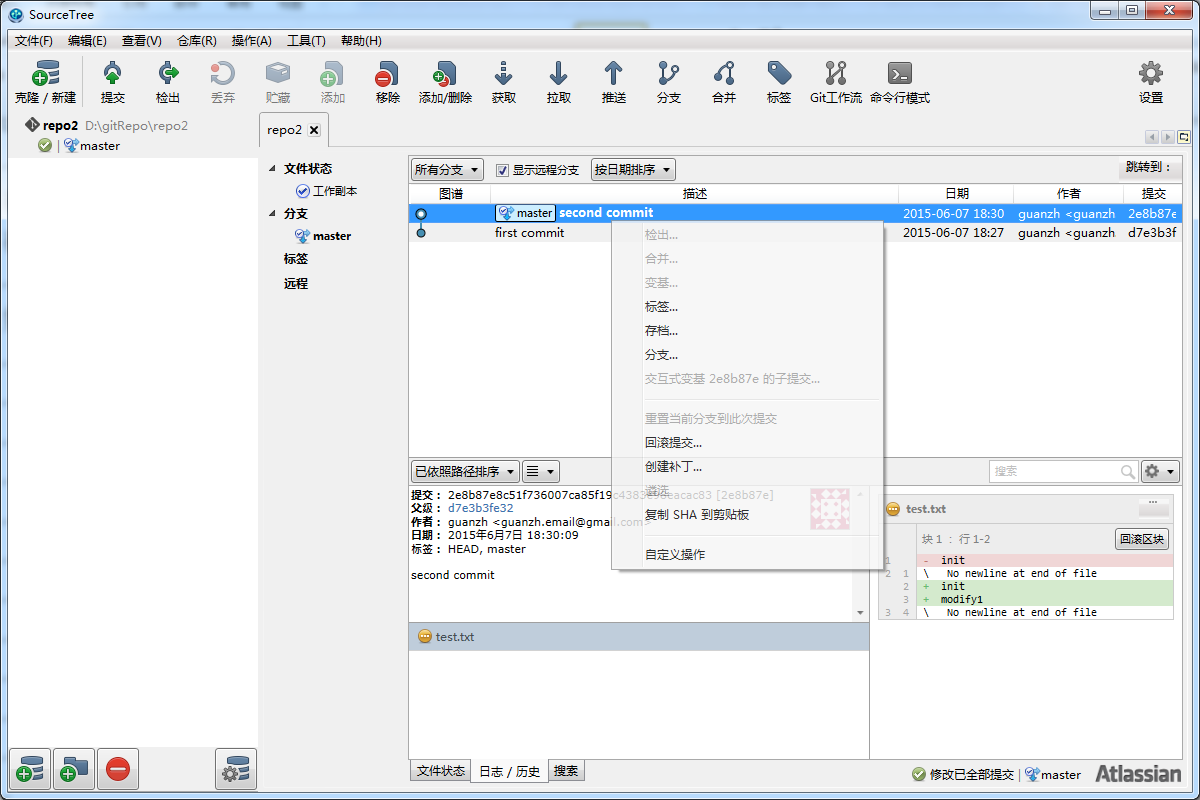
有点类似revision No

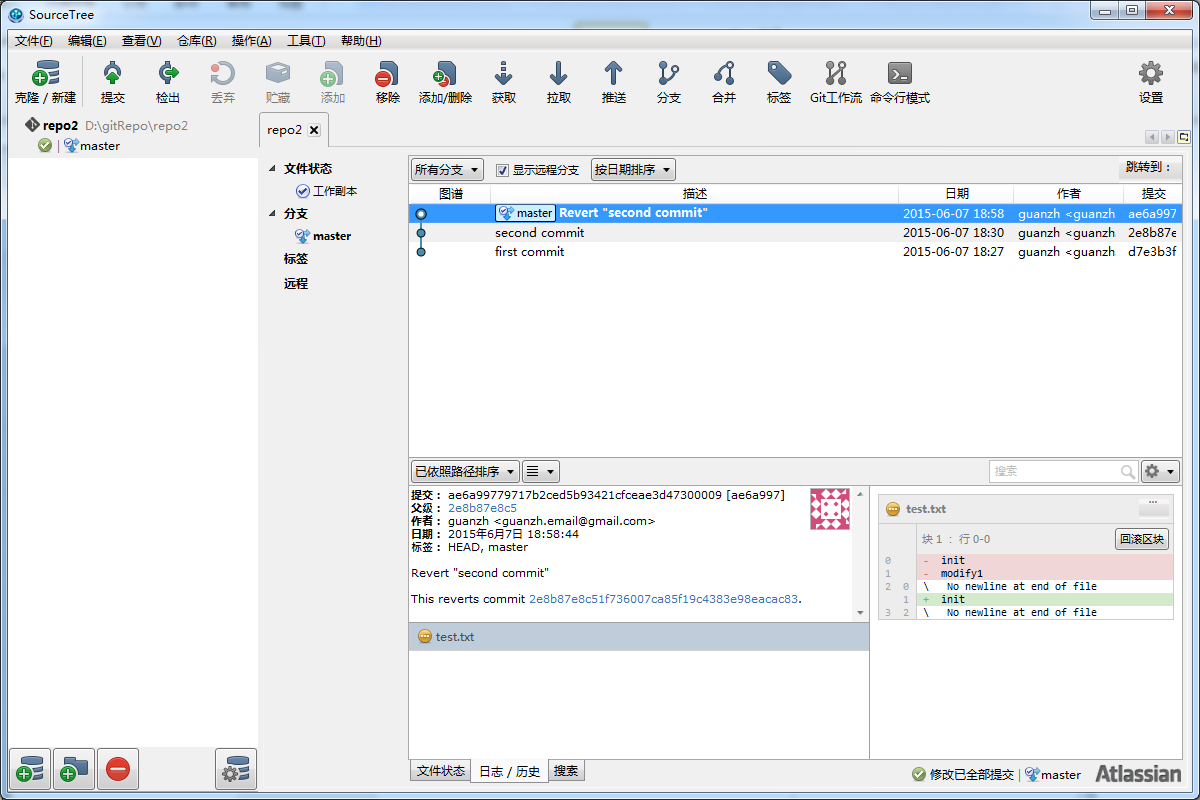
提交履历

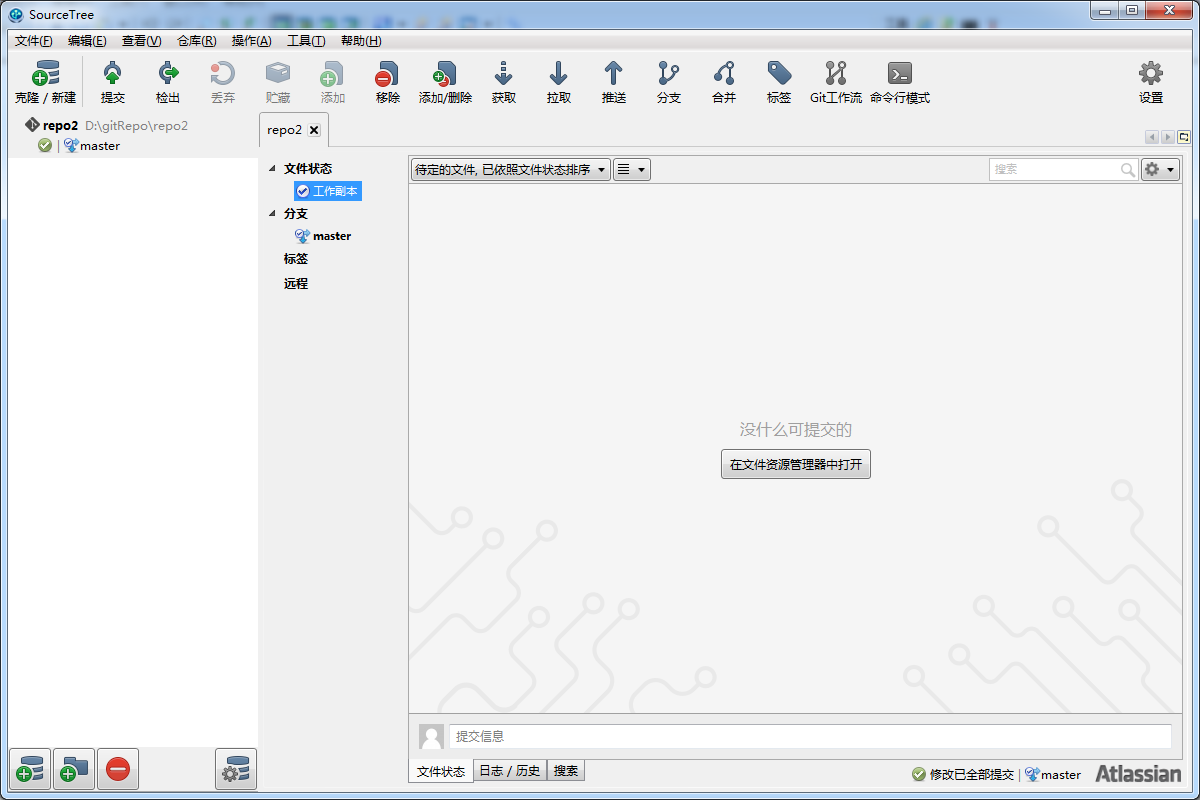
在版本管理中遇到比较多的情况就是恢复到上一次commit的结果，或者说取消最近一次的提交。

在git 中有两个方法来完成这个处理，一个是reset 一个是revert

先看revert



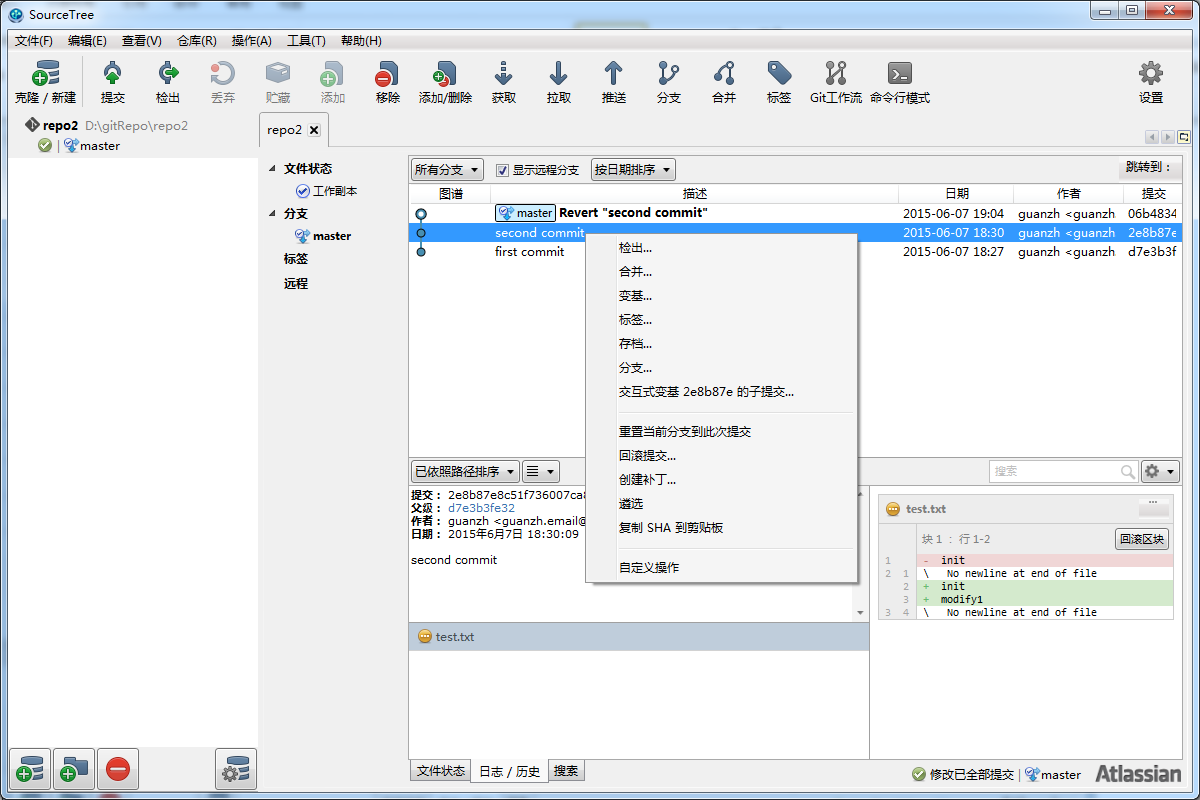


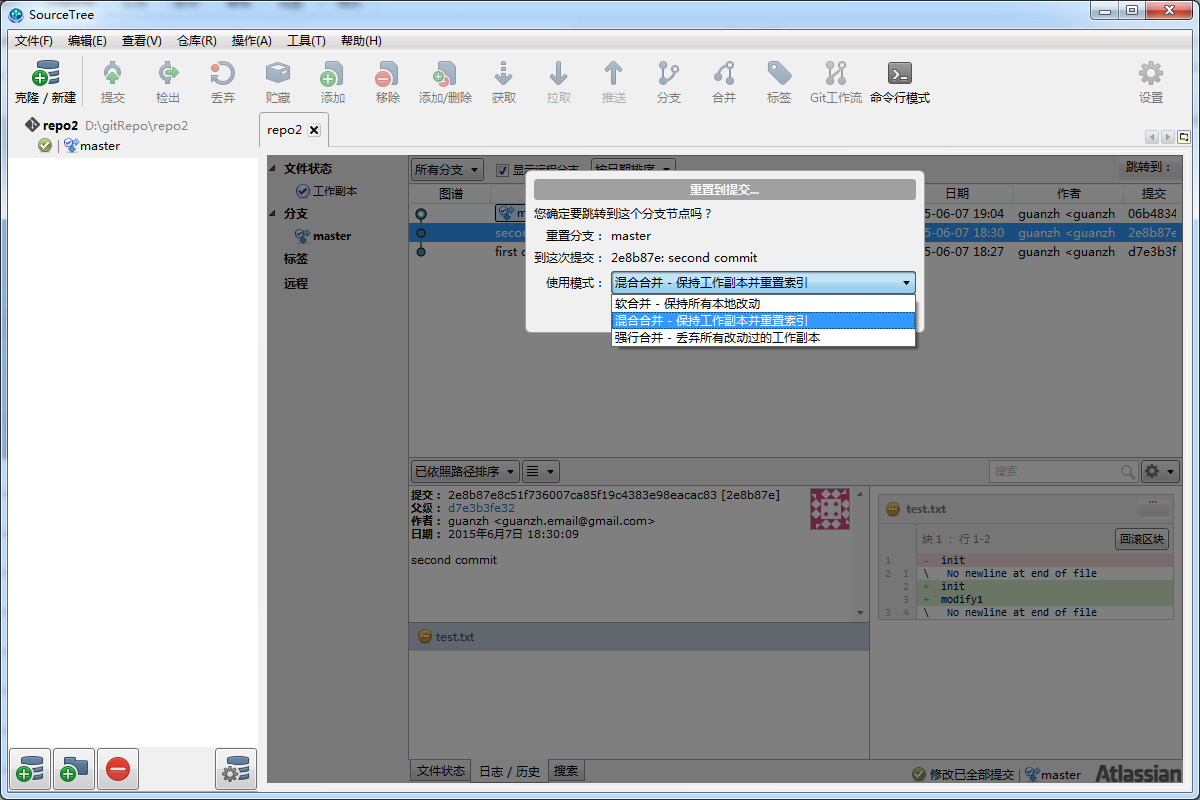


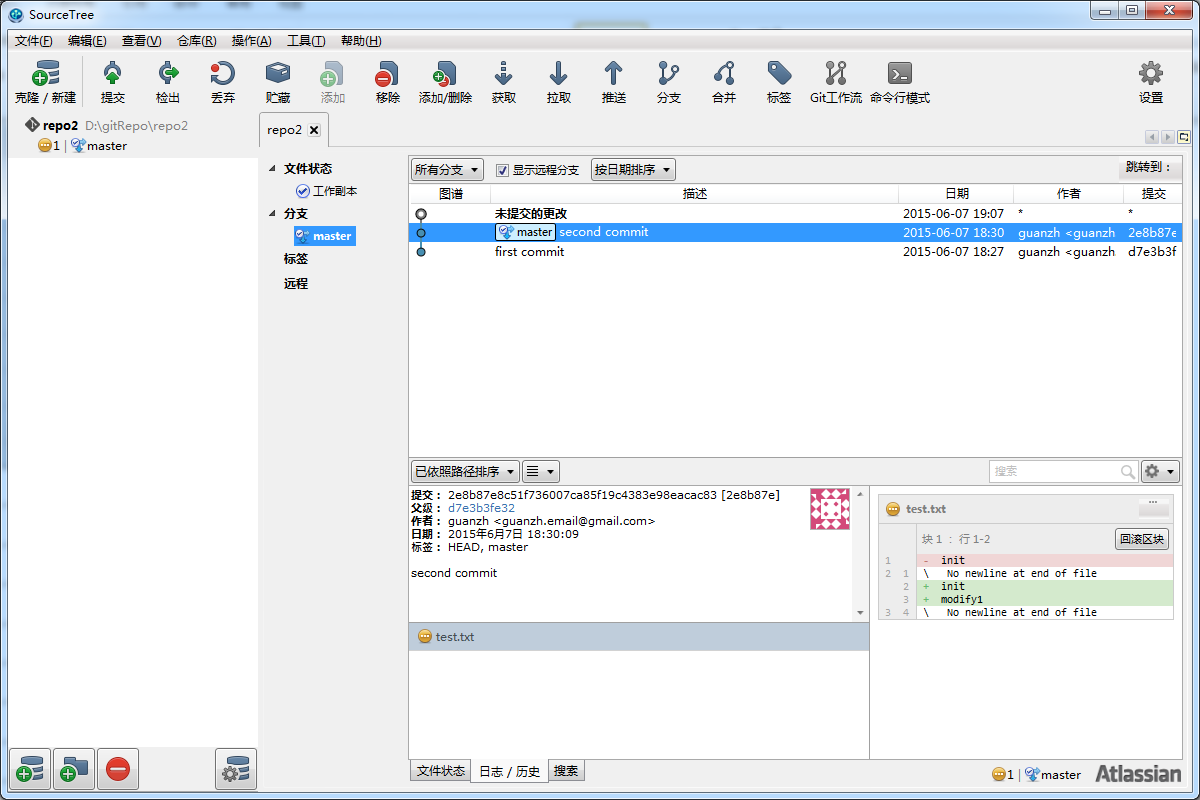
Revert的意思就是把当前这次Commit的内容取消重新再做一次提交。

工作区和暂存区的内容都回复成新的提交后的内容

再看reset







Reset的意思是，选中的提交之后的提交全部彻底删除。

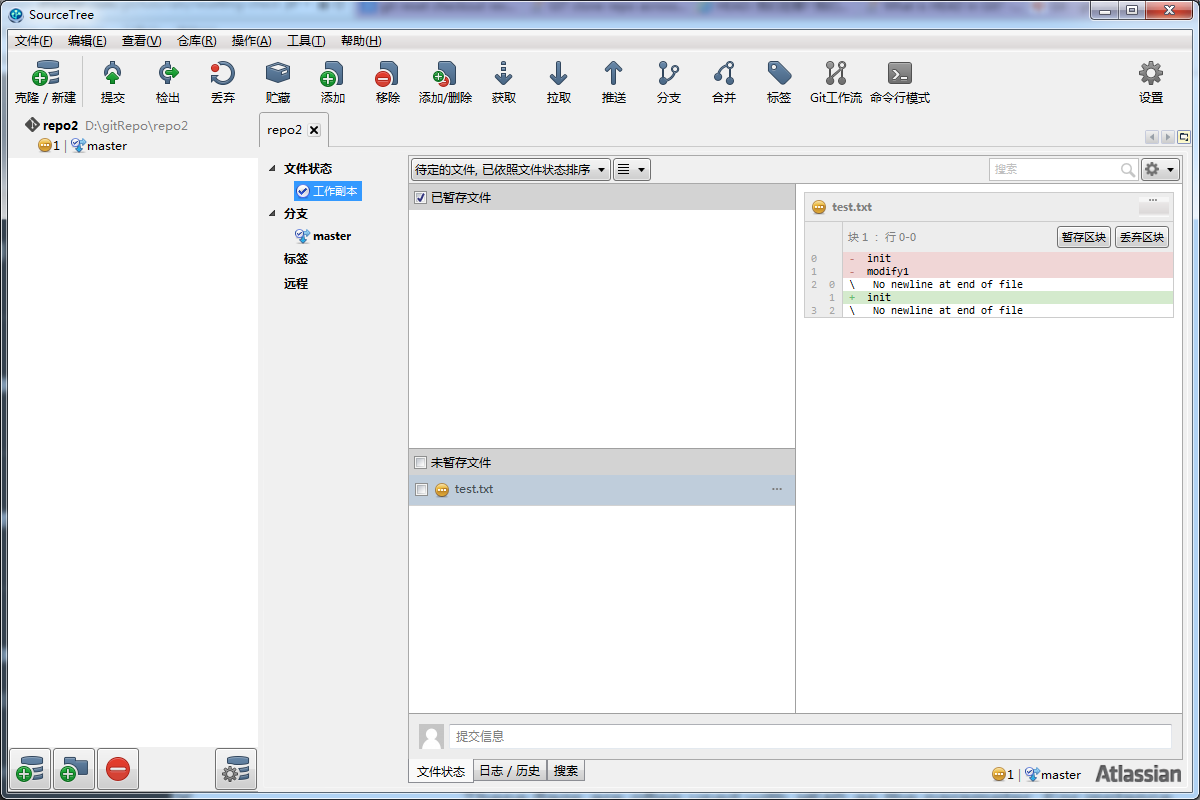
其中软合并 soft ， 混合合并 mix， 强行合并 hard的意思是。

soft的意思是只把 本地存储区的文件恢复。暂存区和工作区的文件还是保持上次的提交内容。

mix的意思是只把 本地存储区和暂存区的文件恢复。工作区的文件还是保持上次的提交内容。

hard的意思是把本地存储区和暂存区和工作区的文件都恢复。

例子中使用的是混合合并所以工作区的文件和暂存区和本地存储区的内容不一样。

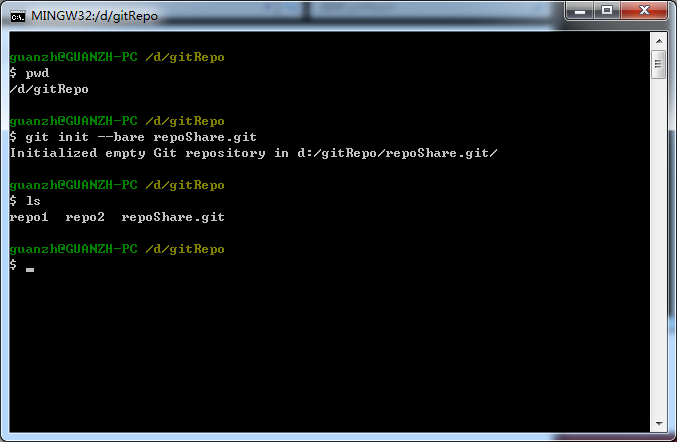


如果你使用Git作为本地一些文件的版本管理。那么上面的一些基本应用可能已经足够了。

在多人合作工作的场景下。还需要了解远程存储区。分支等概念。

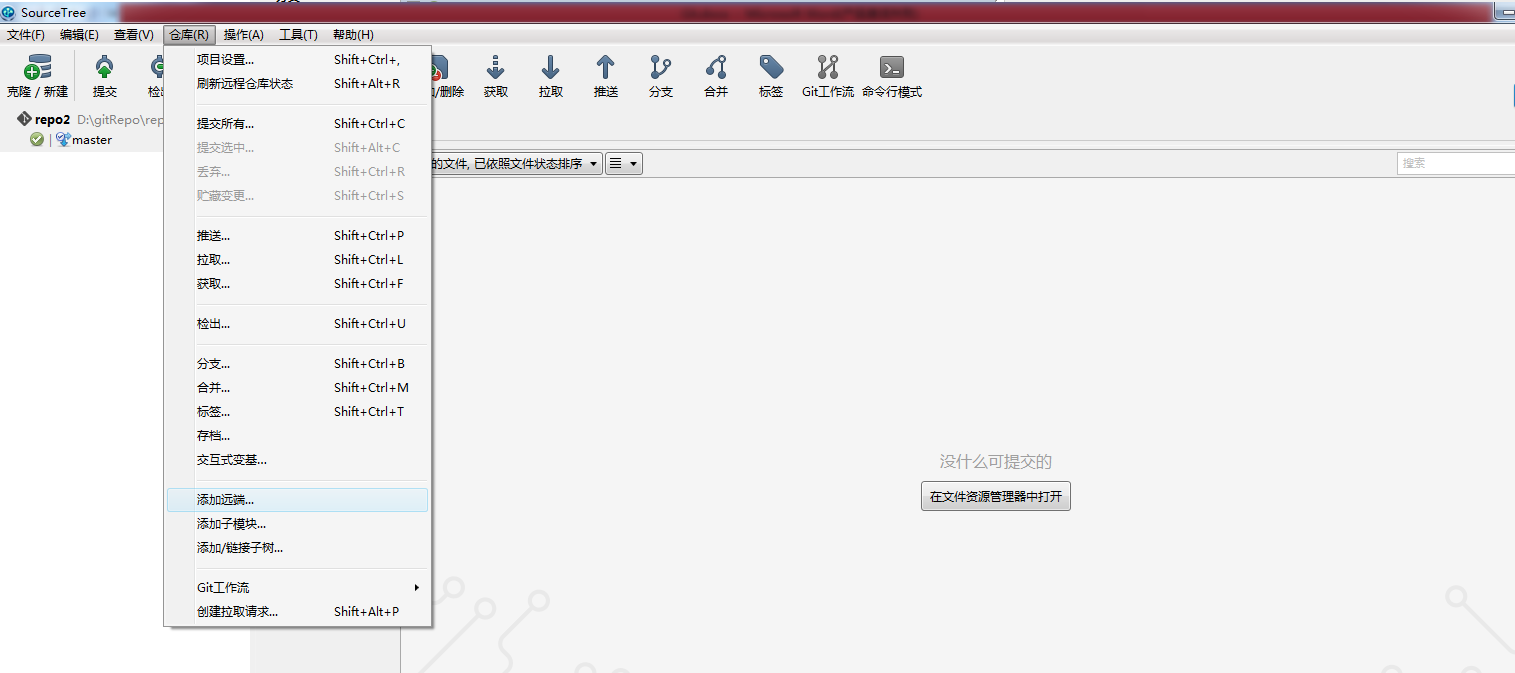
本地存储可以把自己的工作做版本管理，但最后如果需要让其他人也看到。需要把本地的内容反映到远程存储区共享给其他人。GitHub就是一个非常流行的远程存储区。为了简单的描述。本文将在本地创建一个存储区模拟远程存储。

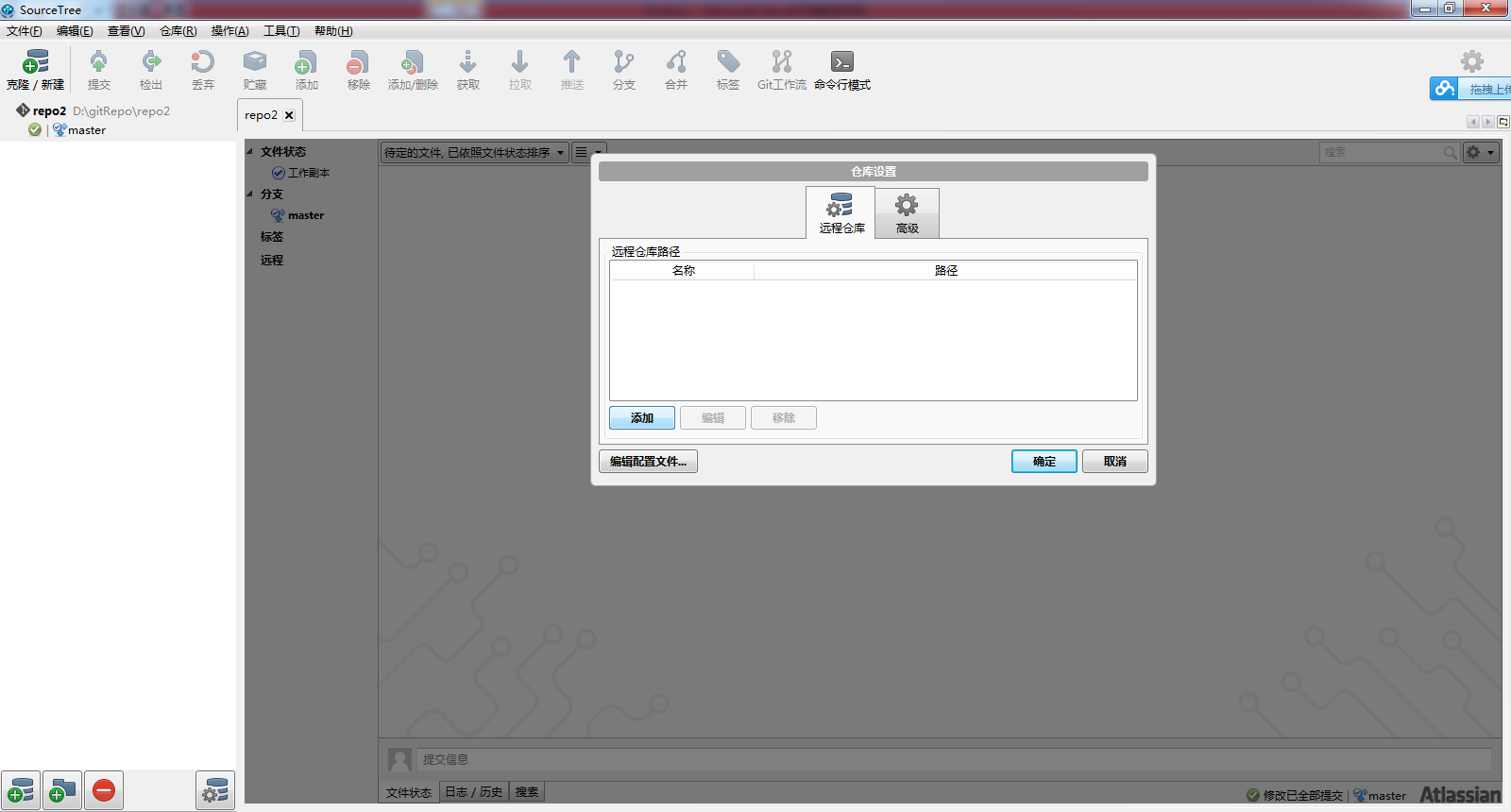
在 D:\gitRepo 使用 git init –bare repoShare.git

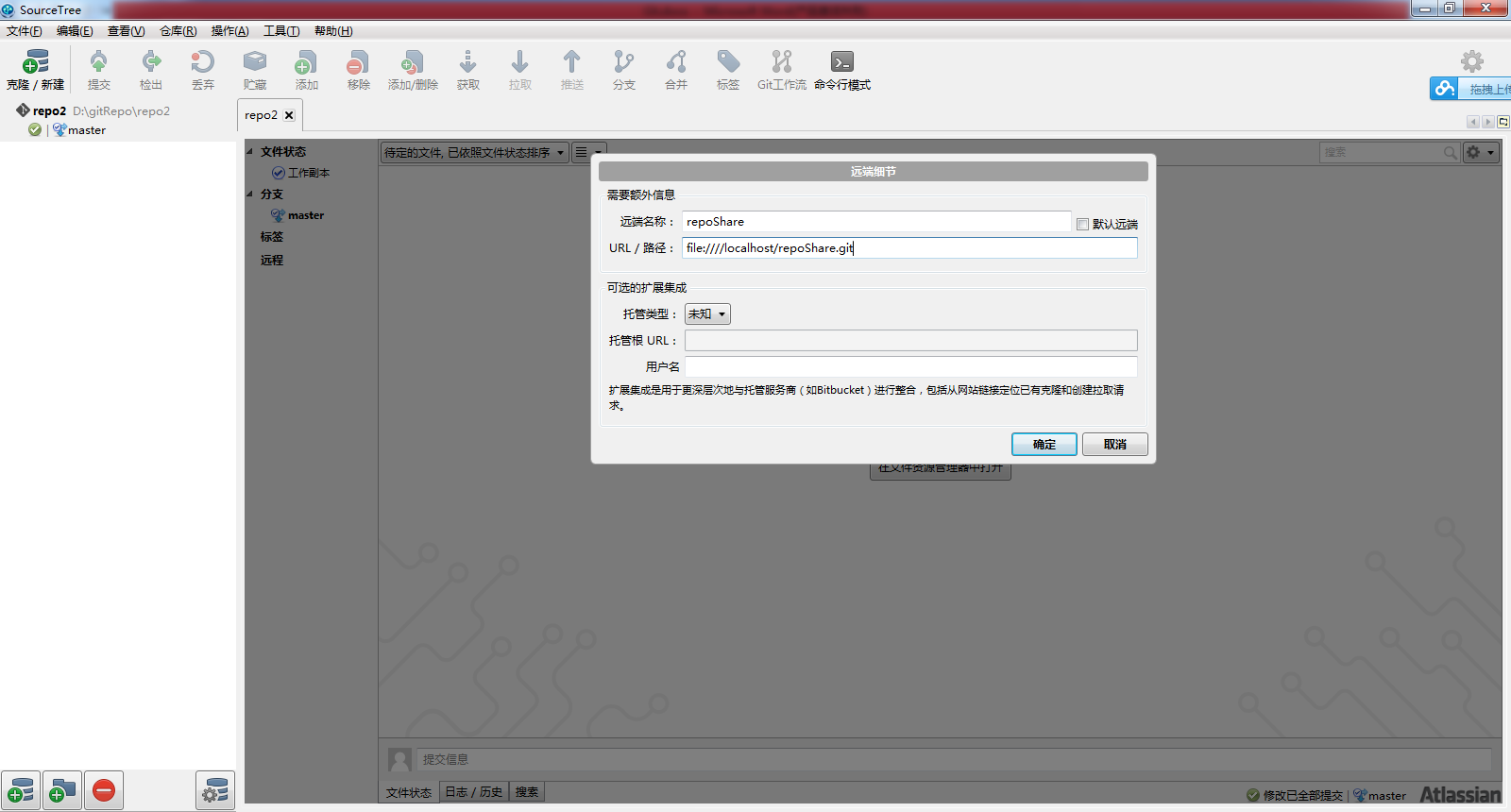


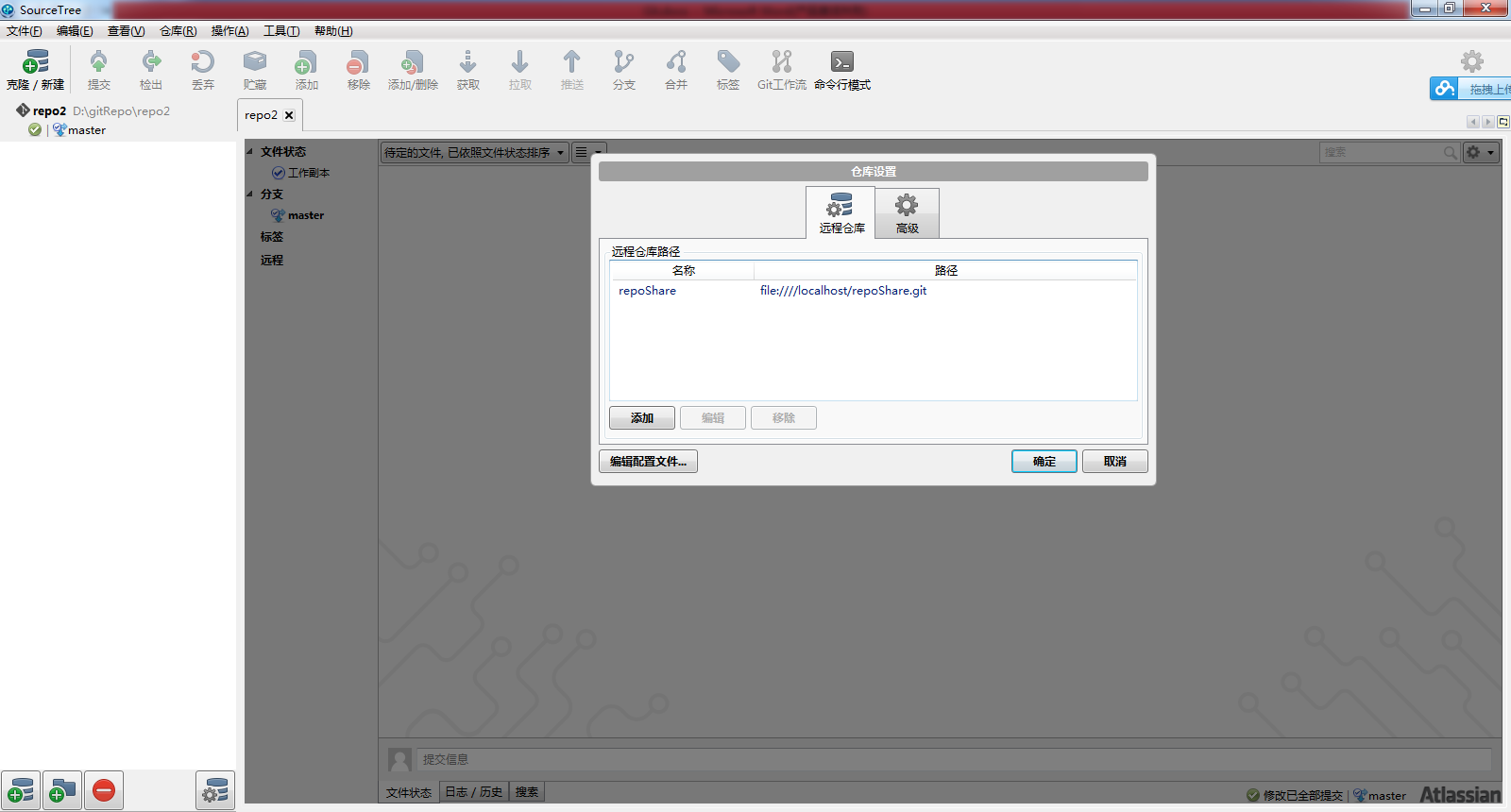
然后将这个repoShare.git目录共享。一个远程仓库就建立好了。

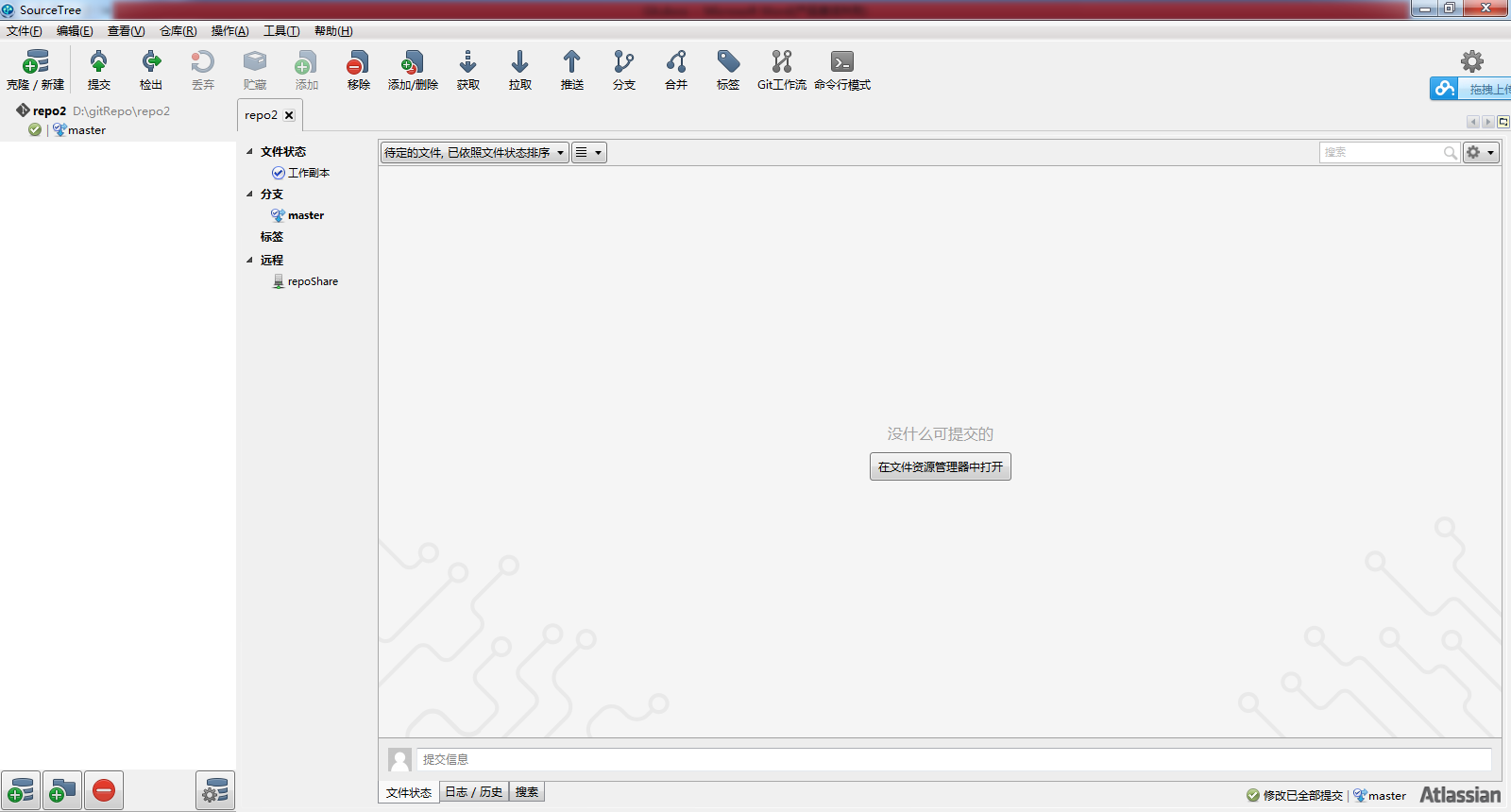
然后在SourceTree里添加这个远程仓库



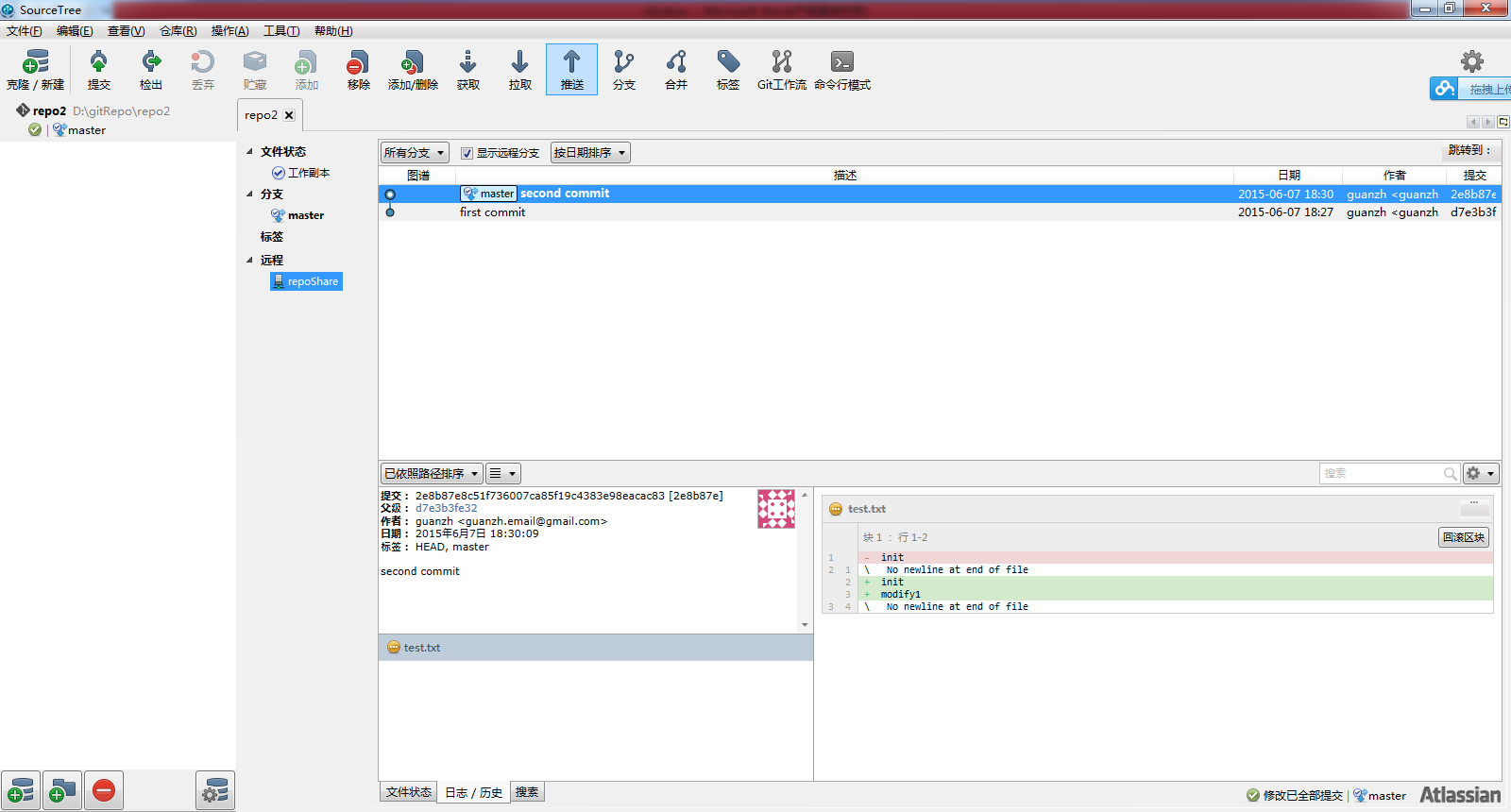






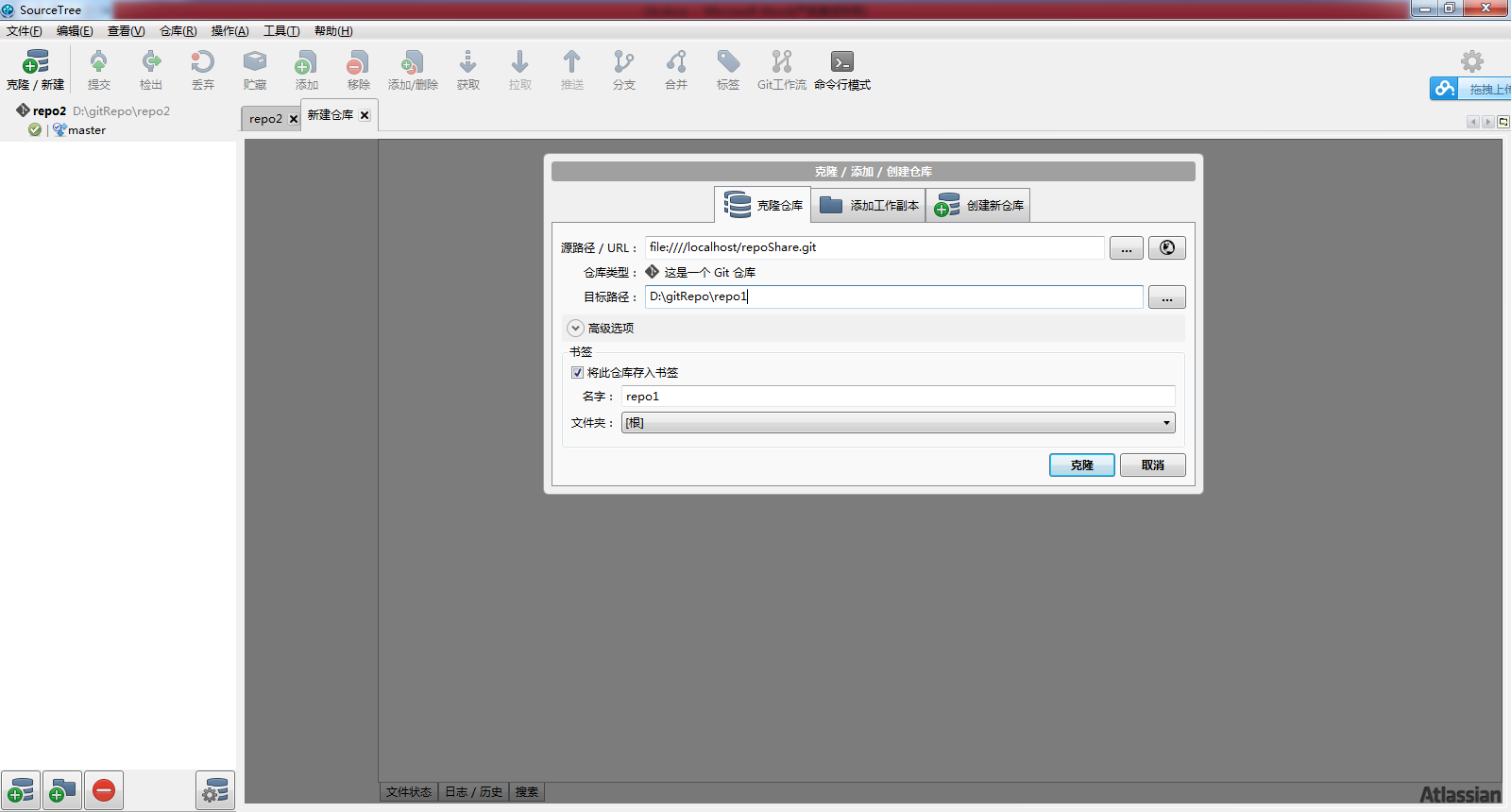


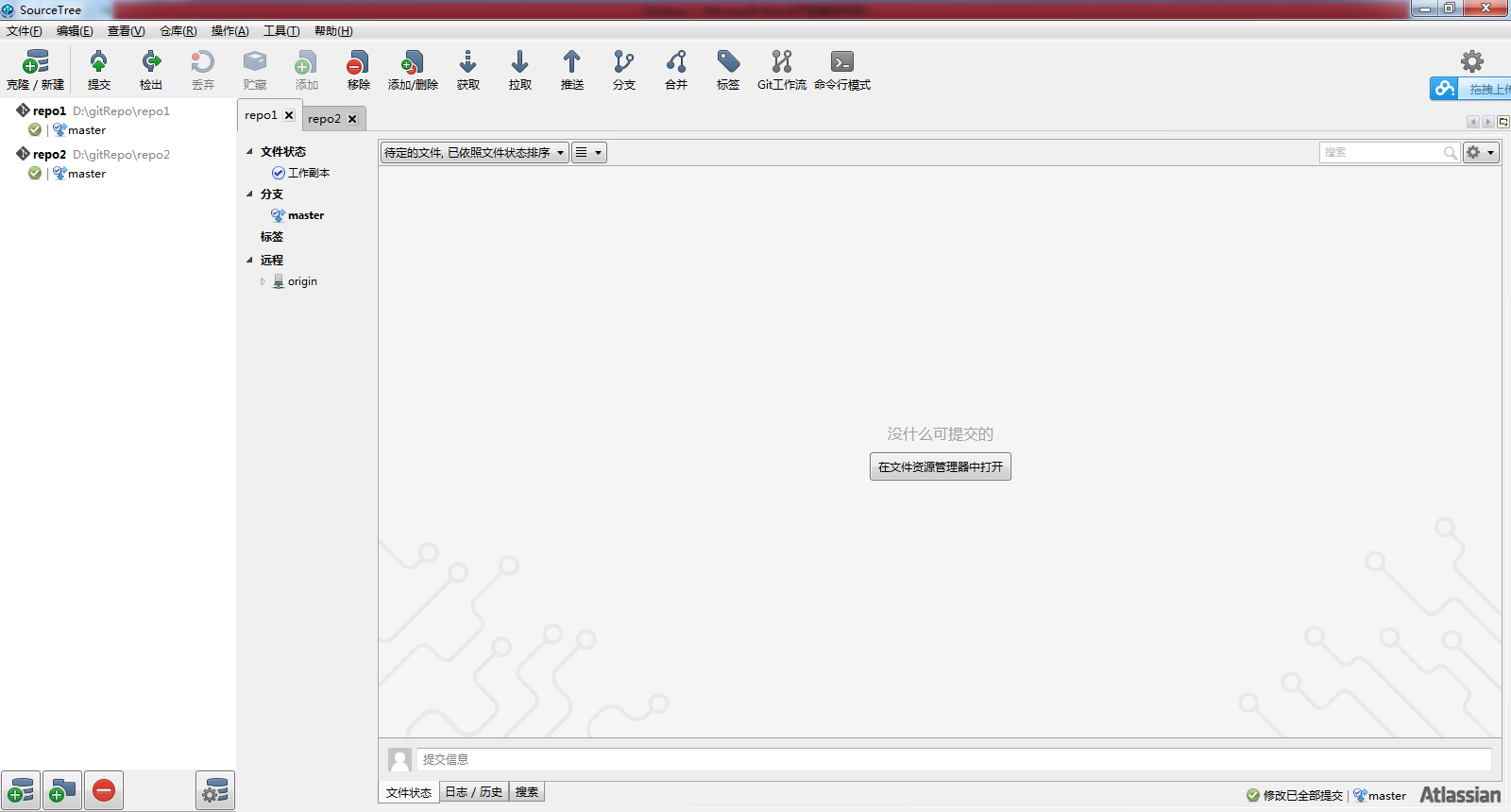
然后将Repo2的内容上传到RepoShare





然后从新建一个本地文件夹Repo1，从repoShare clone 文件





这时候本地仓库repo1 和repo2的内容都一样，并且和远程仓库repoShare一样。内容是

init

modify1

repo1的用户负责给init 修改成 modify. 变成

modify

modify1

假设repo2的用户负责加上modify2. 变成

init

modify1

modify2

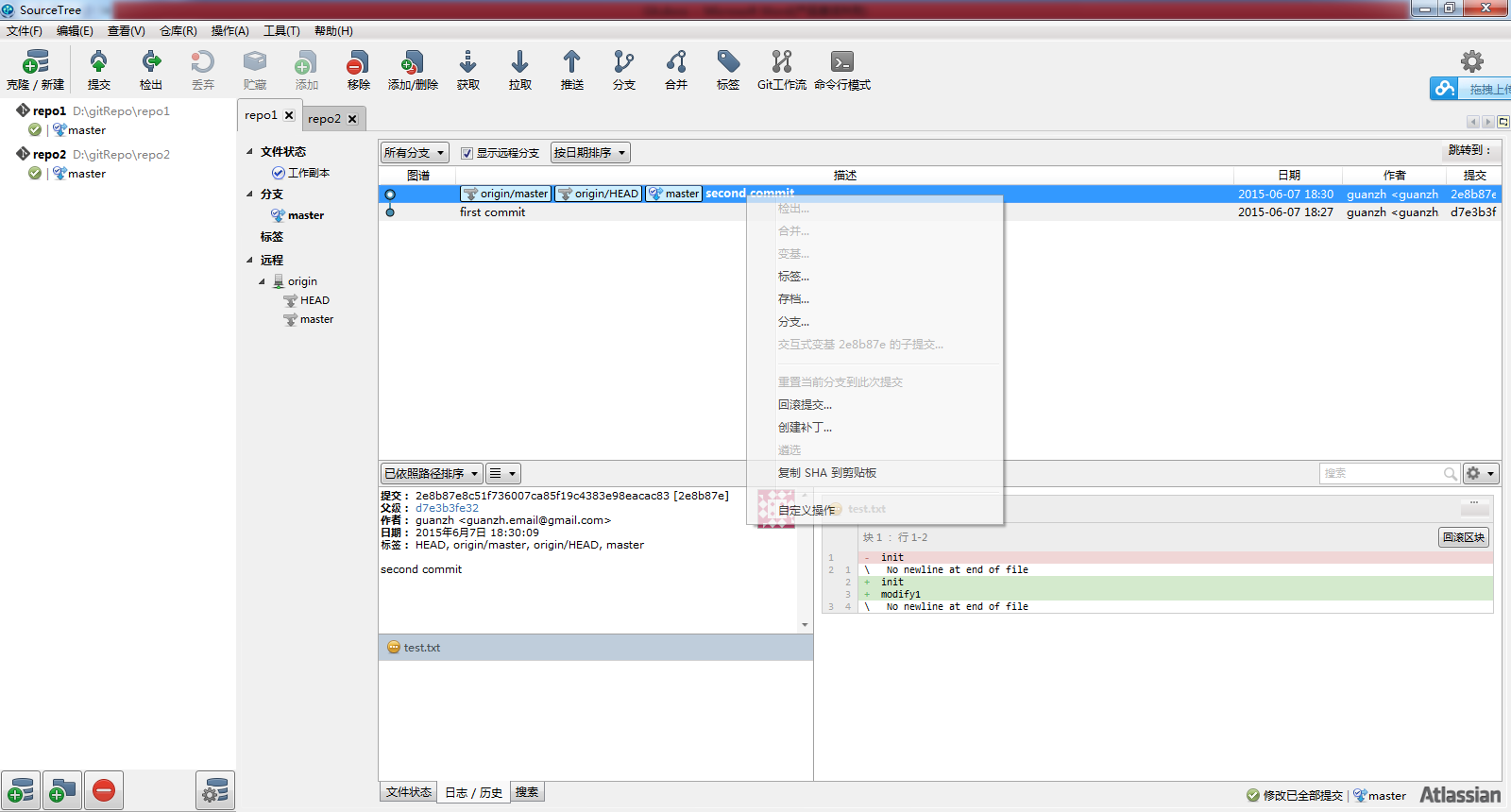
由于Git的Branch功能很强大。所以Git也推荐我们多使用Git的分支功能。

假设现在Reposhare中的是一个稳定版本。Repo1的用户需要增加机能，尔Repo2的用户需要修改机能

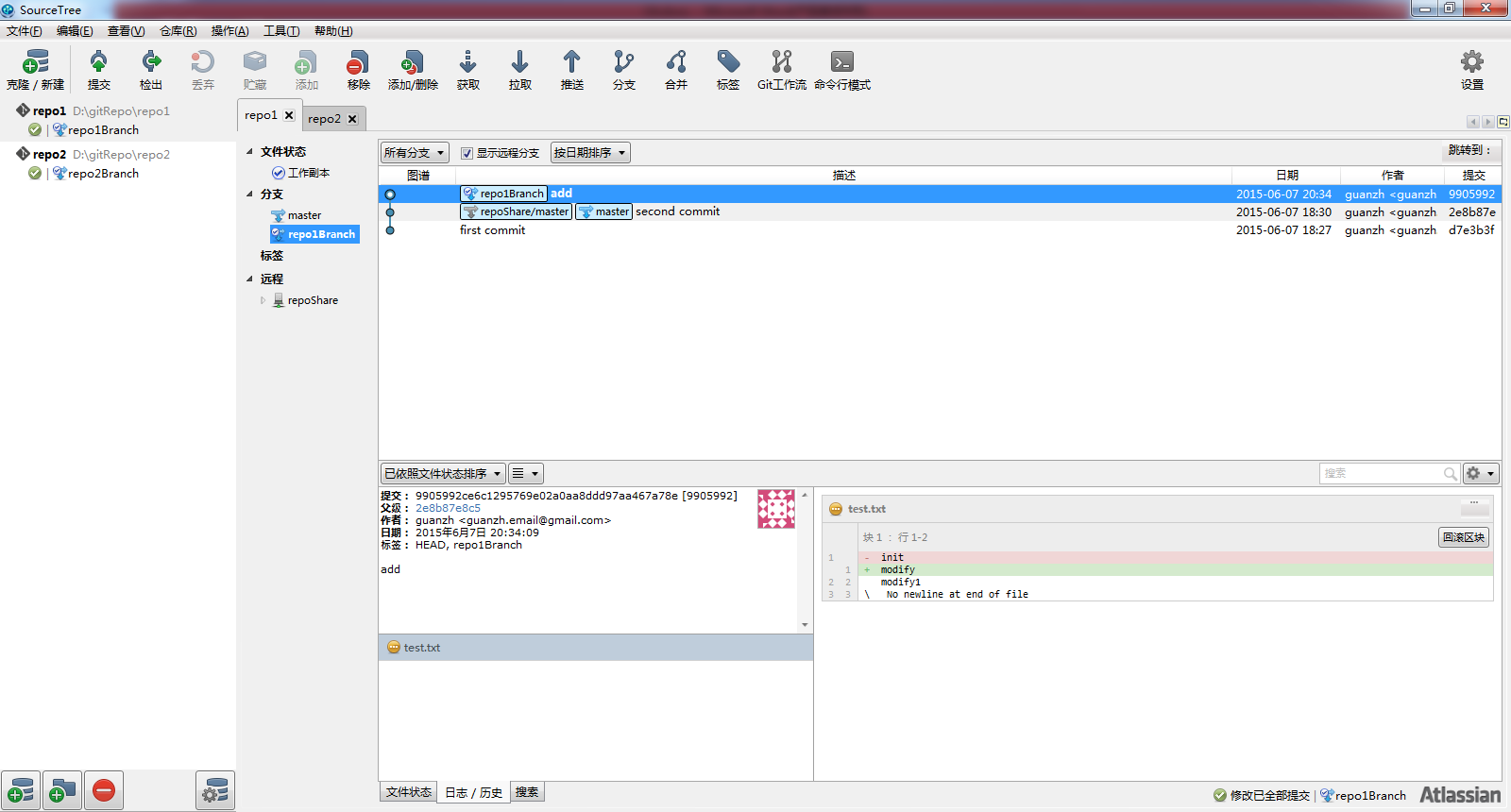
Repo1 的Master 和 Repo2的Master目前都跟踪这远程的Master。

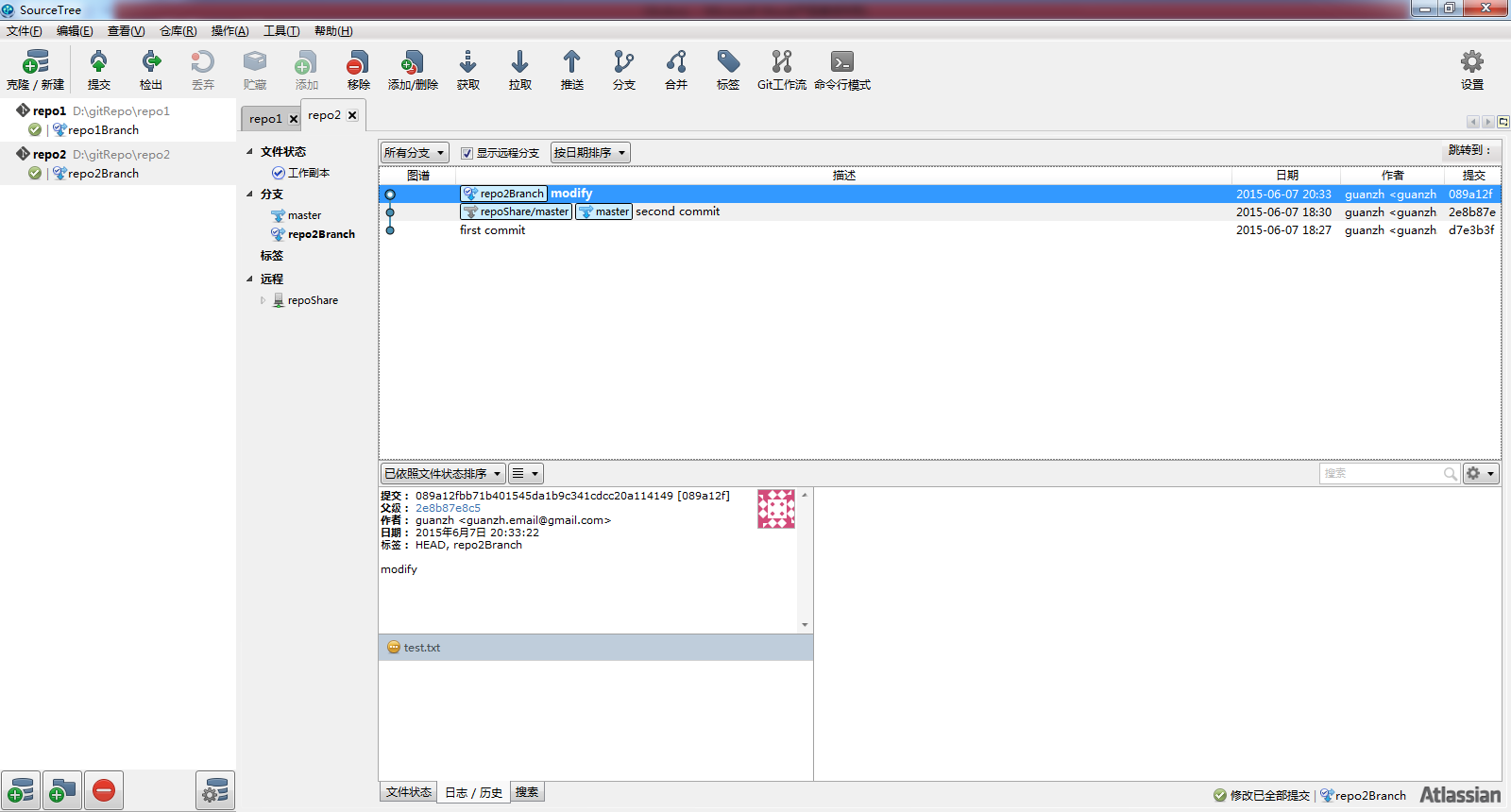
Repo1在本地新建一个branch叫repo1Branch

Repo2在本地新建一个branch叫repo2Branch

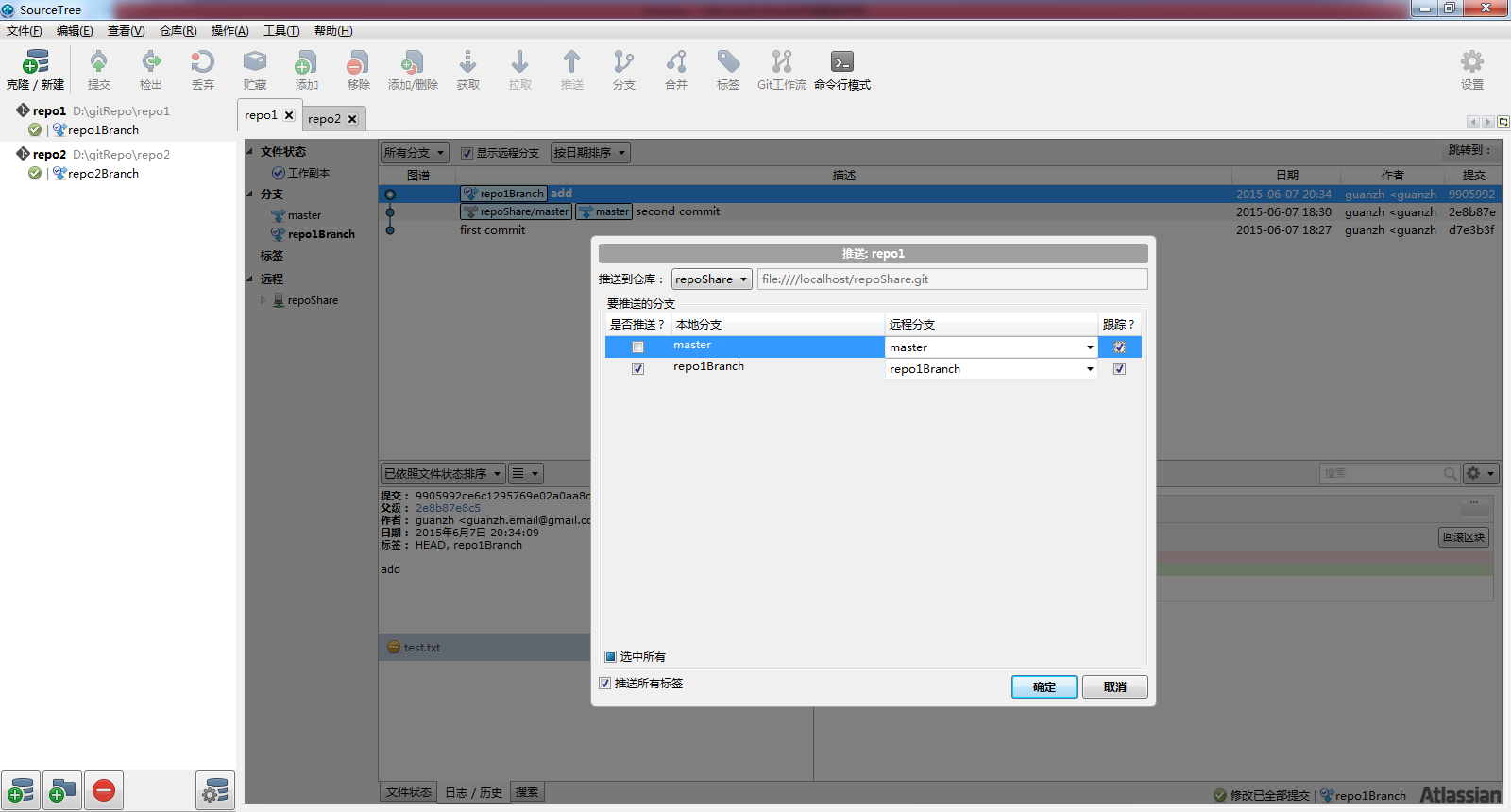


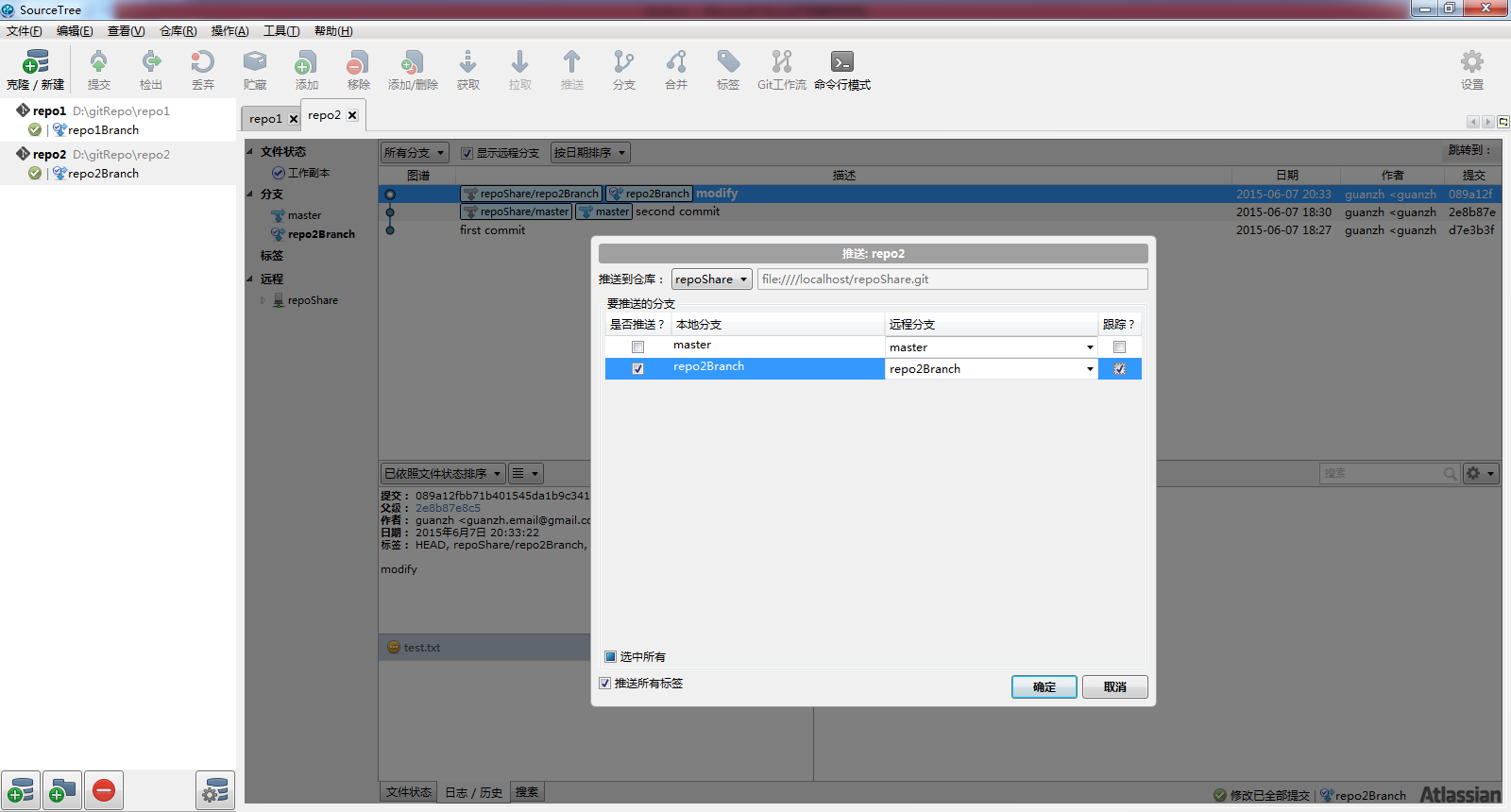
在新建后的分支上做了追加修改后commit





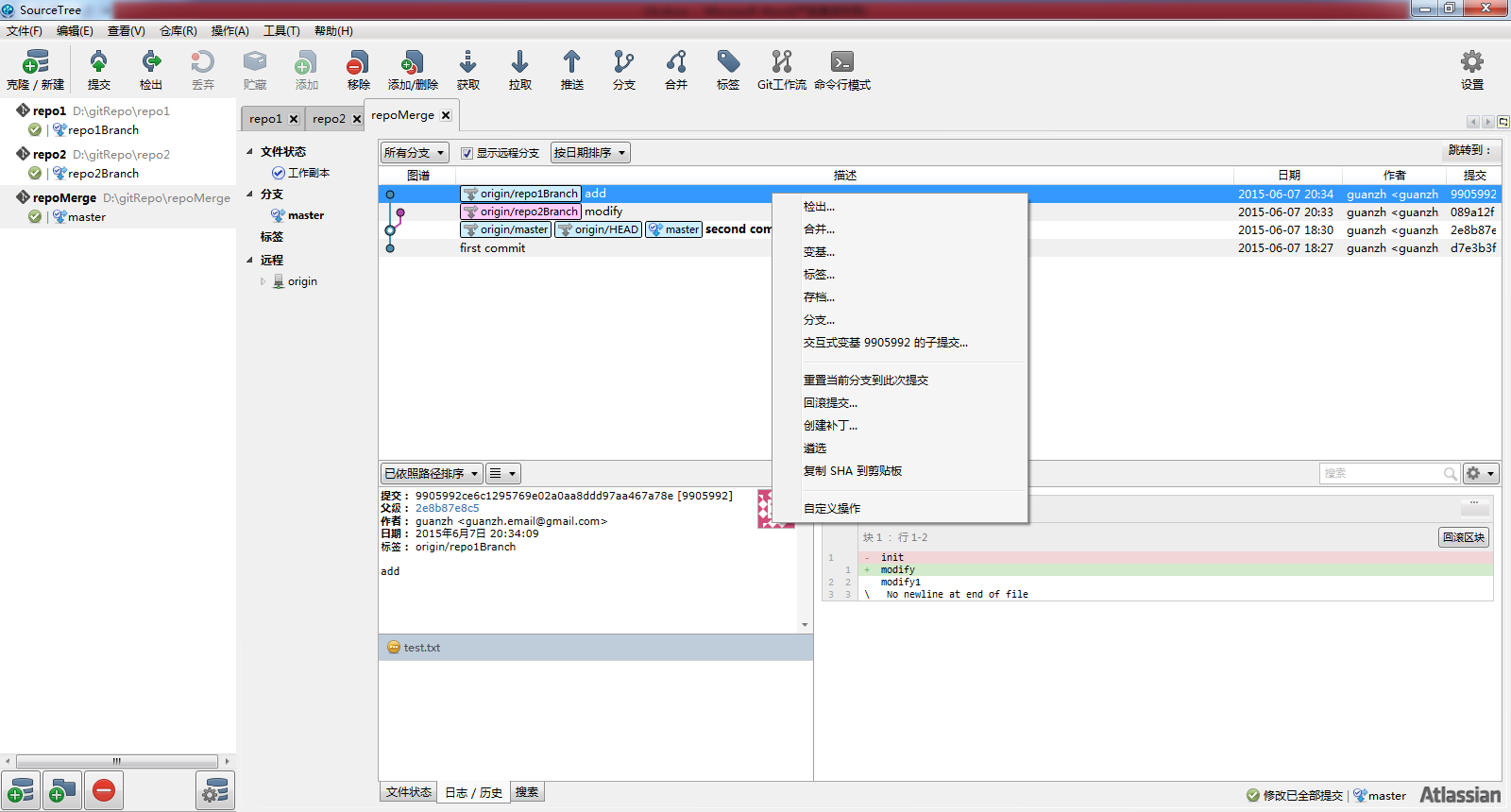
两个用户个将两个Branch 推送到远程仓库

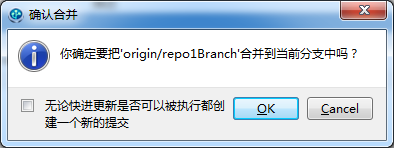




测试人员从RepoShare克隆repo1Branch和repo2Branch分别做了两个测试版本。经过测试后。两个修改都正确。

最后配置管理人员从RepoShare克隆出所有的Branch（master，repo1Branch， repo2Branch）进行Merge

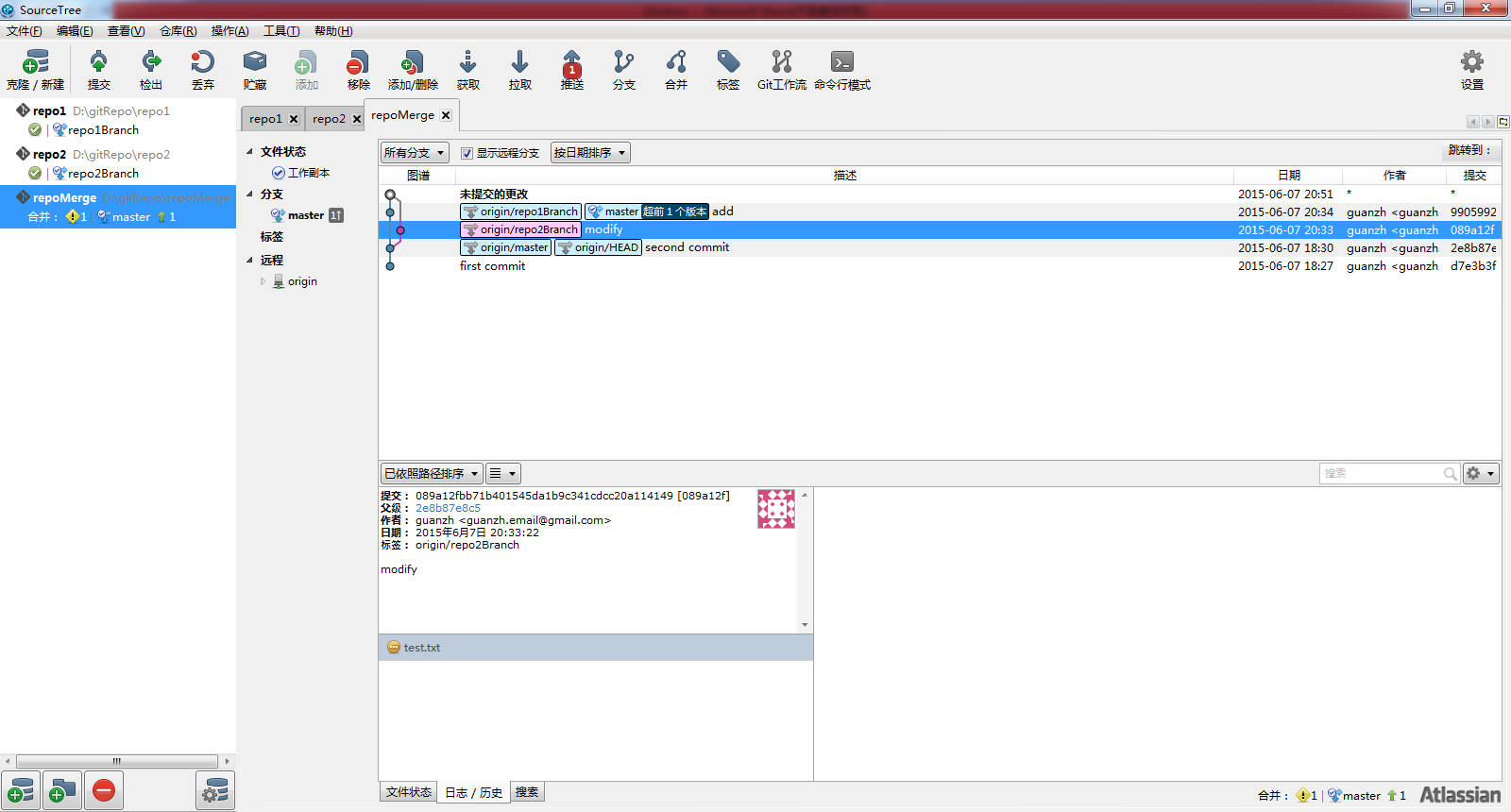




注意当前分支现在是master



再合并第二个



发生冲突后

可以先把冲突取消，然后把合并一次的Master push到远程仓库，然后通知Repo1，repo2用户

告知冲突。Repo2重新取得Master，重新修改Repo2Branch，再一次Push Repo2Branch，直到没有冲突。