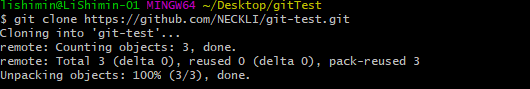
# 基础命令：

## Git clone 克隆仓库代码：

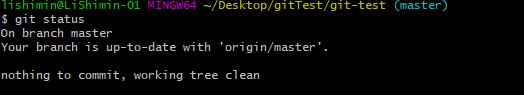
git clone <https://github.com/NECKLI/git-test.git>（仓库地址）

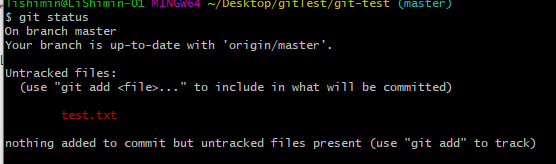




## git status查看本地工作区状态

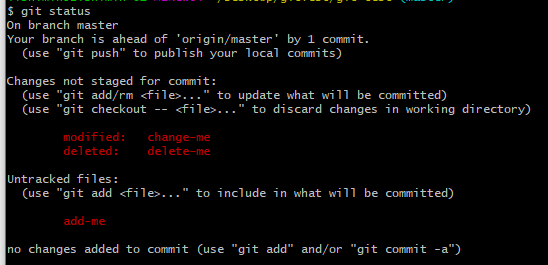
git status

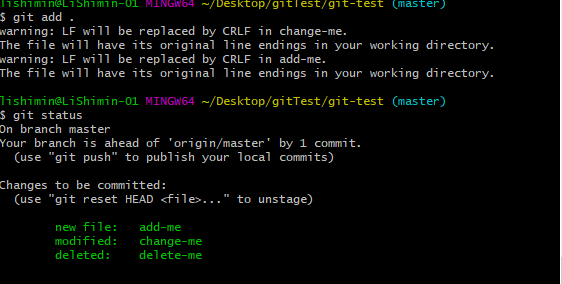




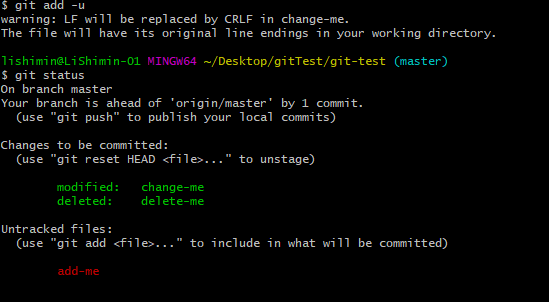
## git add 把修改/新增等提交暂存区

* Git add . ： 提交新文件(new)和被修改(modified)文件，不包括被删除(deleted)文件;--仅针对git版本为1.X的，目前2.X的含义是提交所有变化；

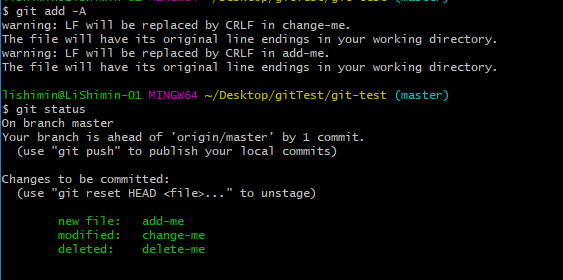




* Git add –u :  提交被修改(modified)和被删除(deleted)文件，不包括新文件(new);



* Git add –A : 提交所有变化;



Ps：在未commit命令之前，如果想撤销add , 可以用命令：git reset；

## Git commit 将暂存区的改动提交到本地仓库

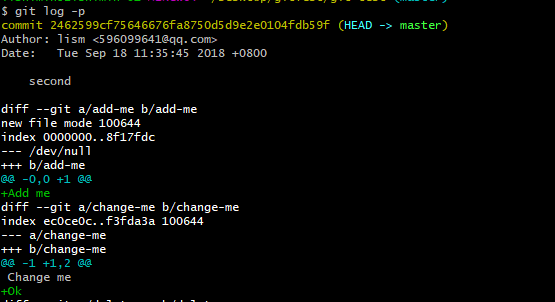
* 每次使用git commit 命令我们都会在本地版本库生成一个40位的哈希值，这个哈希值也叫commit-id，commit-id在版本回退的时候是非常有用的，它相当于一个快照,可以在未来的任何时候通过与git reset的组合命令回到这里；
* git commit –amend (原来提交基础上提交，不是新的提交)

## Git log 查看提交历史

* Git log : 默认不用任何参数的话，git log 会按提交时间列出所有的更新，最近的更新排在最上面。每次更新都有一个 SHA-1 校验和、作者的名字和电子邮件地址、提交时间，最后缩进一个段落显示提交说明—commit –m 里的message。

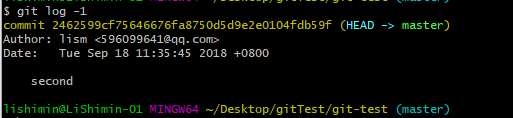


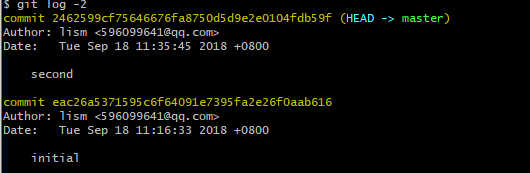
* Git log –p: 选项展开显示每次提交的内容差异



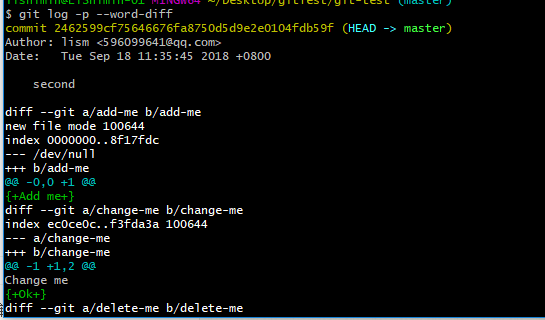
PS:退出 :q

* Git log -1 : 后面加数字，显示最近的几次提交





* Git log –p –word-diff : 单词层面上的对比



## Git pull : 将本地仓更新至远程仓的最新状态，获取代码并自动合并

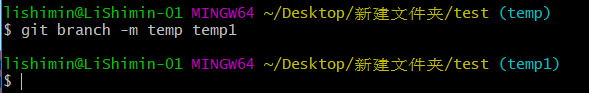


## Git push : 将本地分支的更新，推送到远程仓库；

* 如果当前分支和远程分支之前存在追踪关系，则git push origin 的含义是：将当前分支推送到origin主机的对应分支上；

## git branch –vv:查看本地当前分支与远程分支的追踪关系;

* 
* 更改分支名称：git branch –m 改之前的分支名称 改之后的分支名称



## Git checkout : 切换分支

* Git checkout dev : 切换到dev 分支；
* Git checkout –b dev :以当前分支为源创建分支dev；

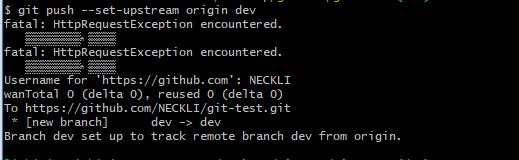
Ps: 是 git branch dev ; git checkout dev 两条命令的简写



如果本地已经存在dev分支，则会提示；



* 新创建的分支，是没有与远程关联的，需要先建立关联，把本地分支与远程建立追踪关系；git push --set-upstream origin dev



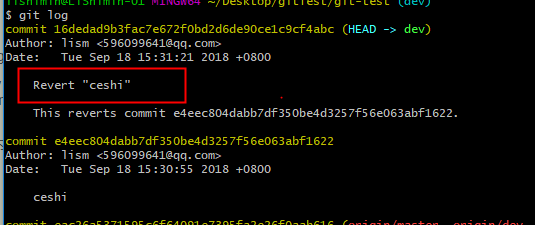


* Git checkout --filename :该命令适用于：已经add后，后来又修改了文件，想要恢复之前add 的内容，则就可以把暂存区的内容替换到工作区了；

或者没add 之前，我想把修改的内容恢复，也可以用git checkout filename;

## Git revert :撤销某次操作，并把此次撤销作为新的提交—谨慎操作

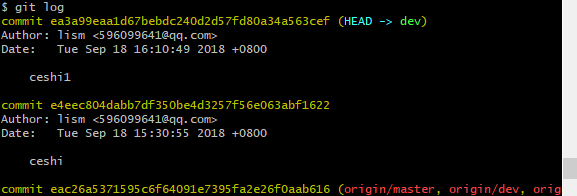
* Git log 里会新增一条记录



* Git revert HEAD ：撤销前一次的commit ；
* Git revert HEAD^ :撤销前前一次的commit;
* Git revert commit-id :撤销指定版本的提交；
* 可以在命令中加上--no-edit参数。另一个重要的参数是-n或者--no-commit，应用这个参数会让revert 改动只限于程序员的本地仓库，而不自动进行commit

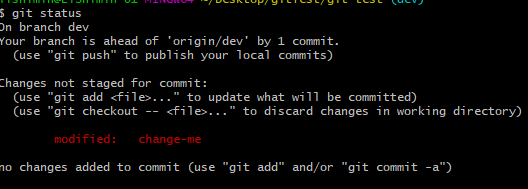
## Git reset : 用来将当前branch重置到另外一个commit的—谨慎操作

* Git reset HEAD : 重置当前分支到最近的一次提交；（HEAD这是当前分支版本顶端的别名，也就是在当前分支你最近的一个提交）
* Git reset HEAD~1 : 将HEAD从顶端的commit往下移动一个commit；从暂存区取出，替换到工作区；









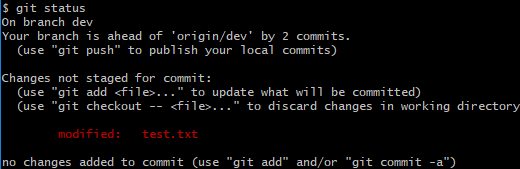
* 参数：--soft : 重置HEAD到另外一个commit，但也到此为止 , 所有变更都集中到暂存区，工作区；

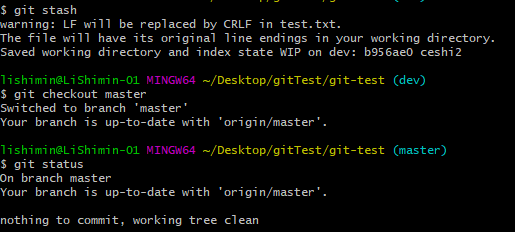
--hard ：危险操作，所有变更全部丢失，不管本地仓，暂存区，工作区；

--mixed : 所有变更保存到工作区，本地仓，暂存区都丢失；默认参数；

## Git stash : 储藏变更；

* 使用场景：当前工作区有变更，且作业没完成，但是需要切换到其他分支进行其他作业，需要干净的工作区间；

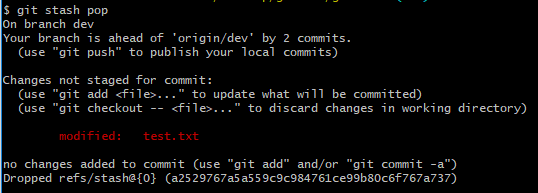




* 查看现有的所有储存：git stash list



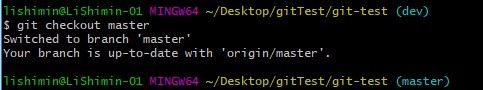
* 其他分支作业完成后，返回之前分支，继续之前作业：git stash pop



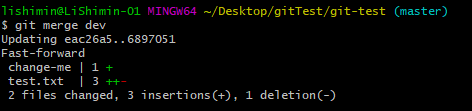
* 删除没用的储存：git stash drop stash@{0} / git stash clear

## Git merge : 合并分支到当前分支

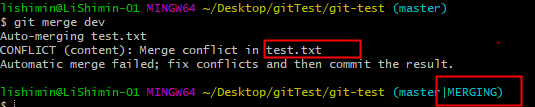
* 首先切换到要合并的目标分支，如把dev分支合并到master分支上，先切到master分支；

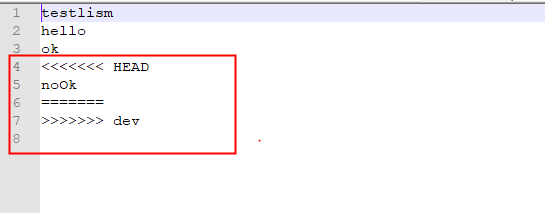


* Git merge dev :把dev 分支合并到当前分支



* 如果没有冲突的话，merge完成。有冲突的话，git会提示那个文件中有冲突，比如有如下冲突：

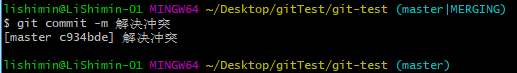




可以看到 ======= 隔开的上半部分，是 master 分支冲突的内容，下半部分是在 dev分支冲突的内容。

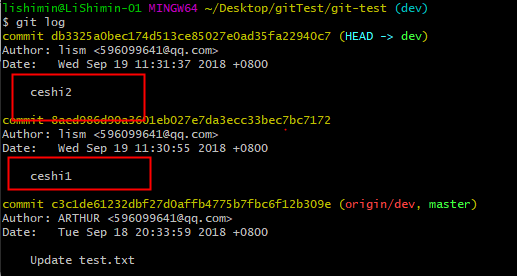
* 解决后：git add ; git commit

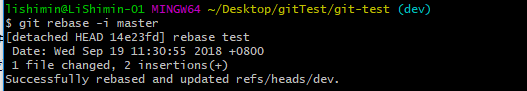


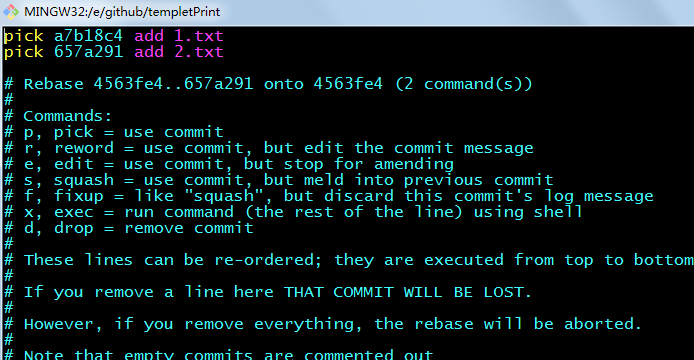


## Git rebase : 对于分支做‘变基’操作，可以理解为‘重新设置基线’

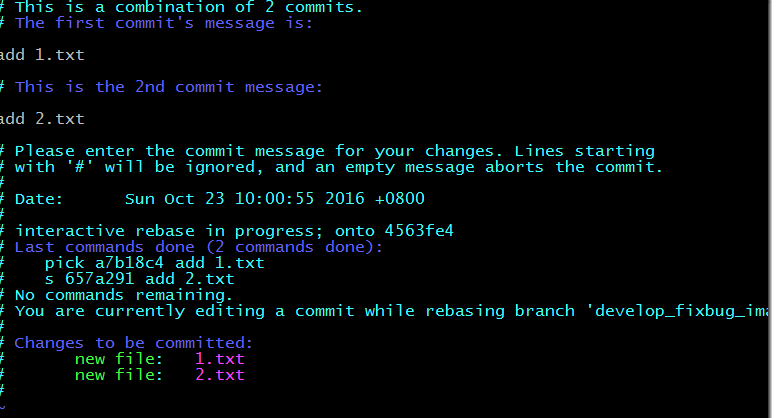
* 该命令适应于自己本地私有分支，远程分支慎用；
* 使用场景1：当你本地试图commit ,push 到一个remote时而因tracking branch过于陈旧而被拒绝的时候（原因是自从之前与orgin同步后，有其他人员提交了很多commit），如果强行push，会覆盖远程仓，把其他人员的commit给丢掉--谨慎，谨慎，谨慎；
* 回到rebase之前的状态：git rebase --abort
* 使用场景2：本地开发多次提交合并，便于管理跟踪，比如：dev 分支从master分支上创建，然后dev分支上自己提交了多个commit ,

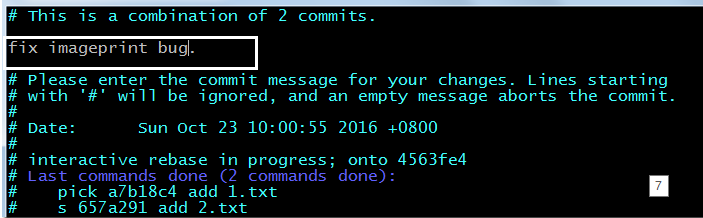




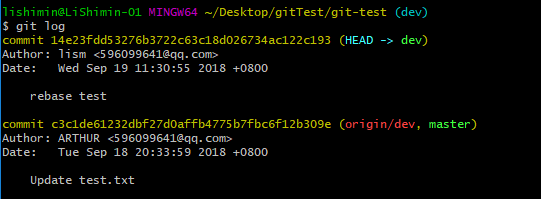


我们设置第二个”pick 657a291 add 2.txt” 为” s 657a291 add 2.txt”这里的s就是squash命令的简写。





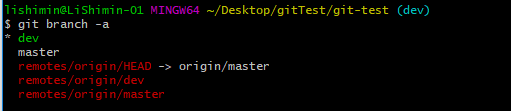
删除之前的两条message(ESC dd)，设置一总的message 然后保存退出。(ESC wq)



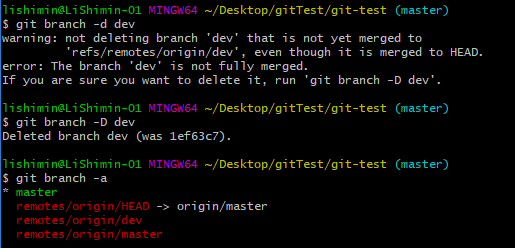
## Git fsck --lost-found : 找回git add 过，但是已经删除的文件内容；--没用过

## Git branch : 查看分支情况

* 查看项目的目前存在的分支情况，包括本地和远程 : git branch -a



* 删除本地分支 ：git branch –D dev (-d , 删除分支； -D ,强制删除分支)



* 删除远程分支

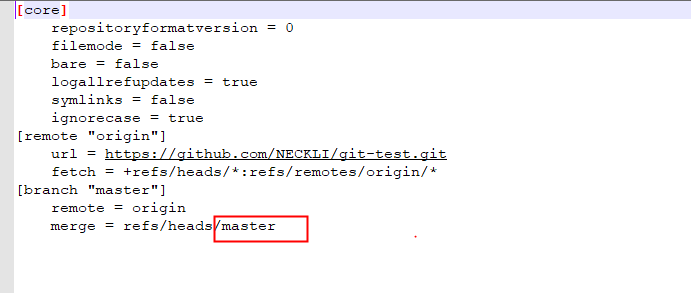
Git push origin :dev



# 常用技巧

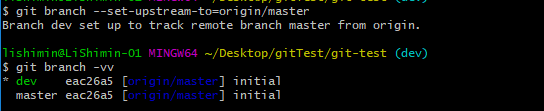
## 修改当前的追踪关系

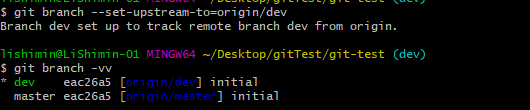
* 文件修改：本地打开.git 🡪 config 文件，别用记事本，修改[branch "master"]这一项是修改本地分支‘master’的远程追踪关系分支，直接修改merge = refs/heads/master为merge = refs/heads/dev；





* 命令修改：git branch --set-upstream-to=origin/master





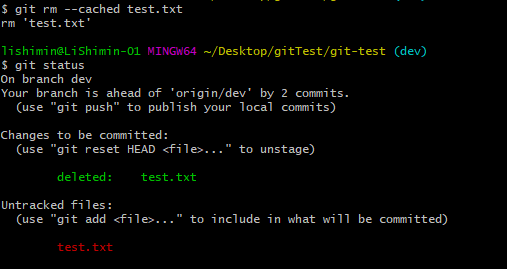
## 查看远程版本库信息：

Git remote –v:



## 停止追踪文件，但不删除本地文件：

Git rm --cached filename



## 更改远程仓库连接地址

* ssh执行：git remote set-url origin [git@github.com:NECKLI/git-test.git](mailto:git@github.com:NECKLI/git-test.git)
* http执行：git remote set-url origin https://github.com/NECKLI/git-test.git

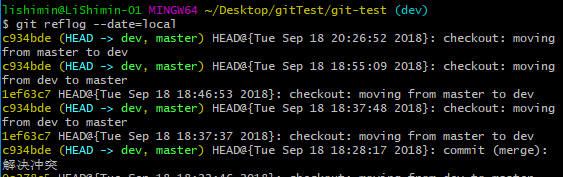


## 放弃本地修改，强制更新到本地

* git fetch --all //只下载远程的库的内容，不做任何的合并
* git reset --hard origin/master //把HEAD指向刚下载的最新版本

## 查看当前分支是基于哪个分支建立的

Git reflog –data=local



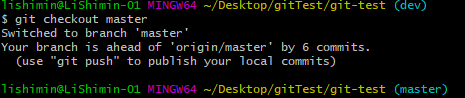
## 将某次提交合并到指定分支上

* 在修改分支上查看日志，找到对应的提交id



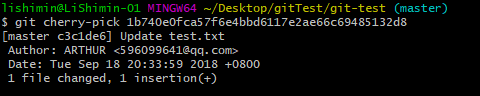
* 切换到指定分支

Git checkout master



* 把指定的提交合并到当前分支

Git cherry-pick commit-id

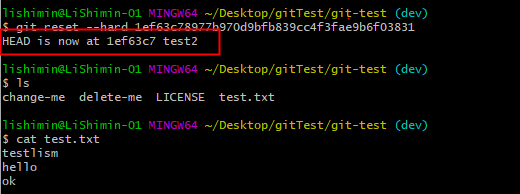


## 版本回退

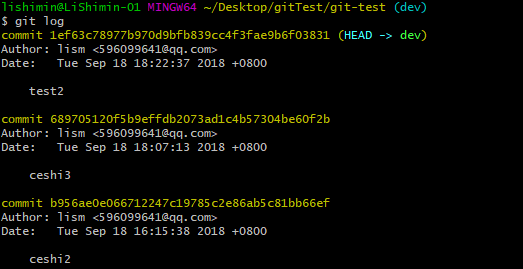
* 先查看提交日志，找到要回退的版本，git log



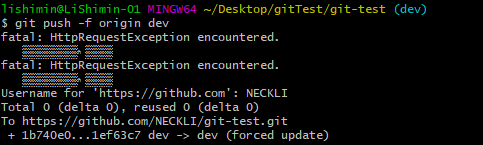
* 复制对应的commit id，假如：1ef63c78977b970d9bfb839cc4f3fae9b6f03831 ；test2提交； 本地版本回退：git reset --hard commit-id (如果确定不要该提交之后的提交，则可以用hard参数，如果要修改保留，则把hard参数去掉，重新提交)



* 本地之前的提交会全部丢失，使用的时候要谨慎；



* 本地强制覆盖远程，使用的时候要谨慎，谨慎，谨慎 ,会影响其他人员使用，如果某人错误覆盖，则远程的其他提交没有了记录，要找回的话，可以找其他人员没有更新之前的代码，再同步到远程，如果其他人手里也没有了，则可以使用git fsck找回，操作会麻烦，--没具体实现过（http://blog.sina.com.cn/s/blog\_66cd08930102x0ln.html）



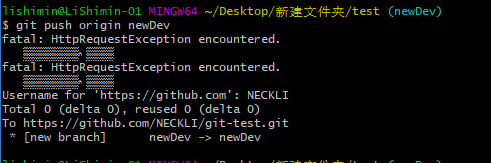
## 本地创建远程分支

例子：如现在有master , dev 分支，现在本地想以dev 分支为基分支创建新分支，并推送远程仓库

* Git checkout –b newDev



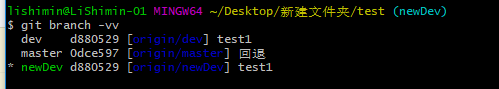
* 将本地newDev 分支作为远程newDev分支





* 跟远程对应仓建立追踪关系





这个时候创建远程仓完成；

* 在仓库里创建新分支比较简单，方便

## 本地多次提交，push之前优化commits

* Git log 查看日志，找到要合并的commits
* Git rebase - i HEAD~2 ; //优化最近的两次提交
* 对于 commit 合并可以使用 squash、fixup 指令，区别是 squash 会将该 commit 的注释添加到上一个 commit 注释中，fixup 是放弃当前 commit 的注释。
* 编辑后保存退出，git 会自动压缩提交历史，如果有冲突，记得解决冲突后，使用 git rebase --continue 重新回到当前的 git 压缩过程；
* Git push 推送到远程，再看git log ，提交日志就简洁多了。

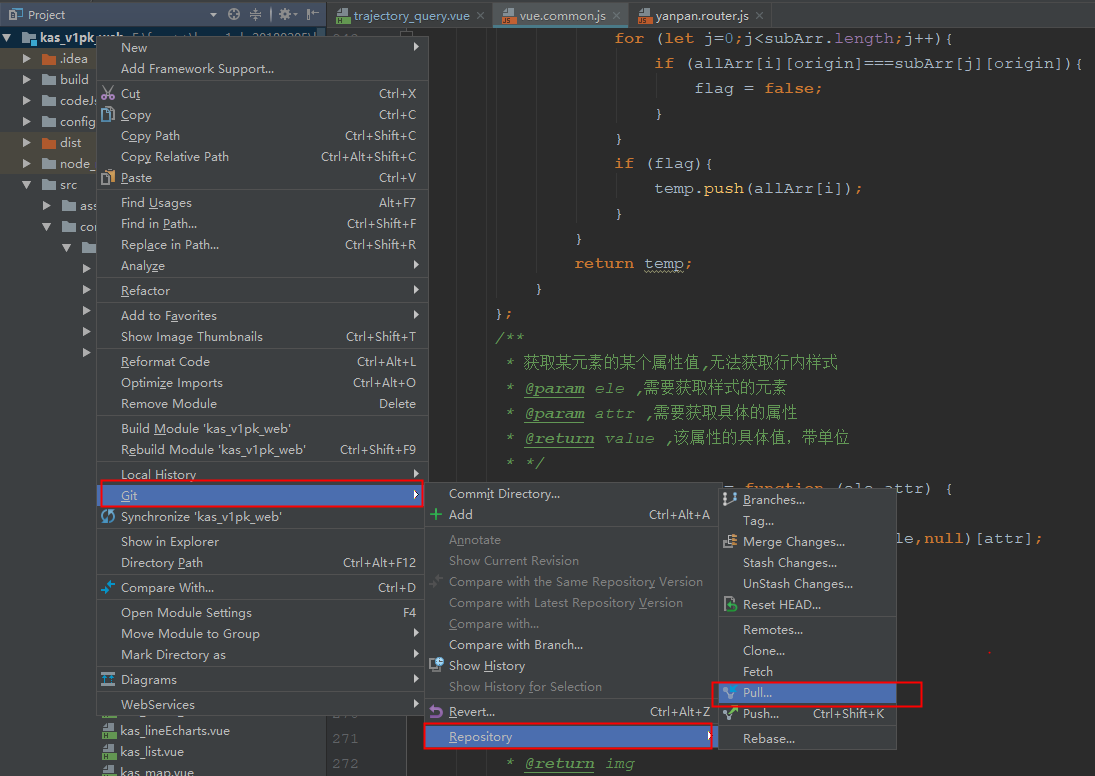
## 本地全新项目，建本地仓，然后推送到远程

* 远程仓库先建立一个仓库；
* 切换到项目目录下，然后初始化本地git仓，git init ；
* 将本地仓与远程仓进行关联，git remote add origin https://..... .git
* 把项目添加到本地仓，git add \* git commit –m ‘注释’;
* 推送到远程仓，git push –u origin master ，（远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令）。

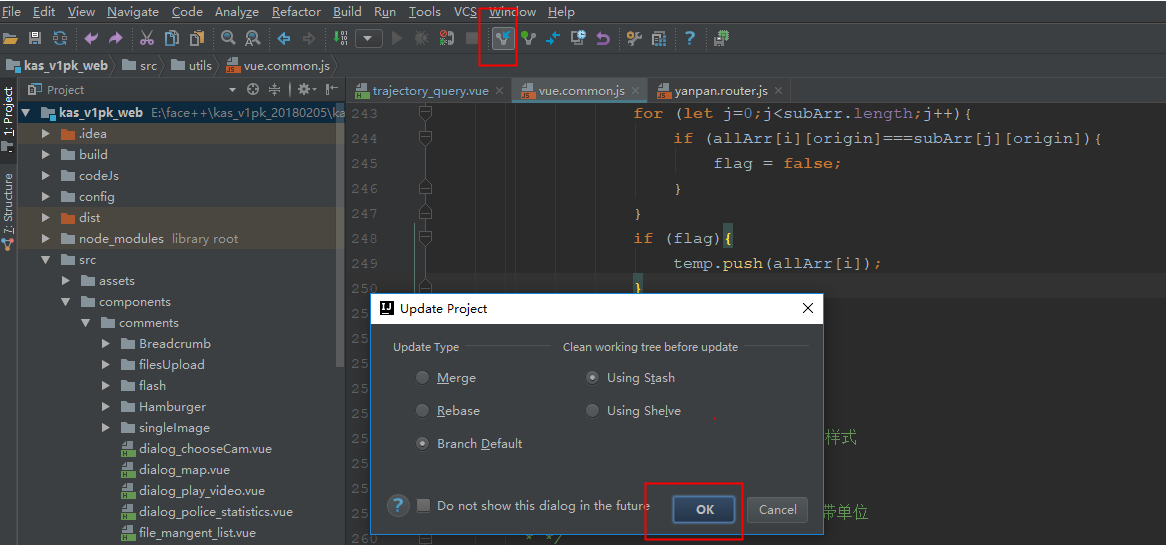
# Idea 工具中git 常用技巧

## 更新代码

* + 方法一：右键项目 -- git –> respository 🡪 pull



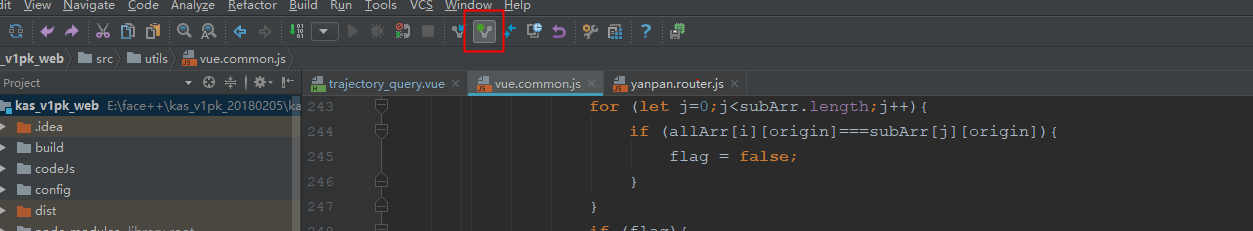
* + 方法二：工具栏中，快捷方式



* + 方法三：菜单栏 –> vcs 🡪 update project 或者 git 子菜单 pull
  + 方法四：idea 快捷键： ctrl + T;

## 本地提交

* 参照更新代码的1，2，3 ，使用commit



* Idea 的快捷键：ctrl + K;

## 推送远程

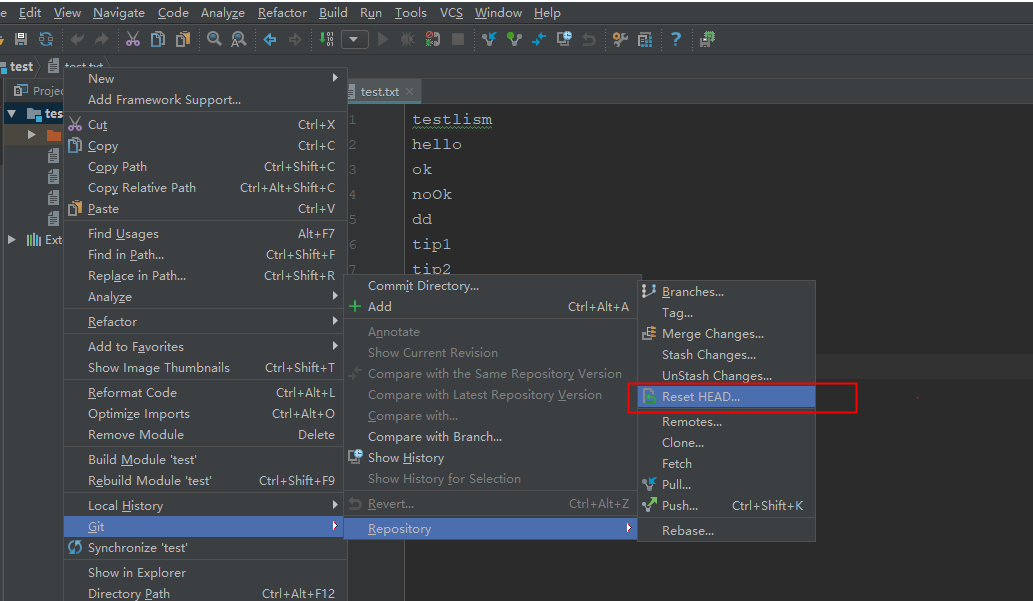
* 参照代码更新的1，3；操作变为push;
* Idea 快捷键：ctrl + shift + K;

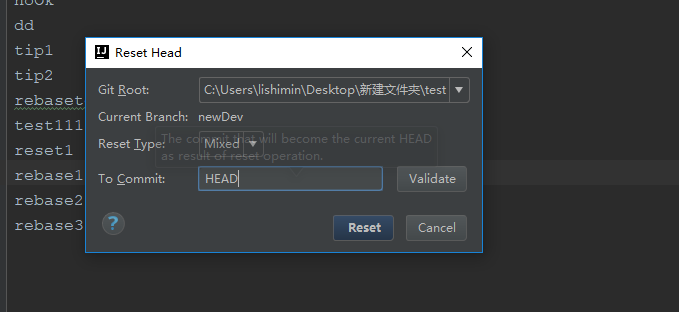
## 冲突解决

* Pull的时候，如果有冲突，idea 智能找多冲突的文件，提示选择本地文件，还是远程文件，还可以自己个性化选择解决；
* 当冲突内容太多的时候，看情况放弃本地或远程，手动把更新添加的方式进行解决；
* 当pull 或 push 报错，进而无法解决时，自己本地无修改，则可以适应强制覆盖本地措施；

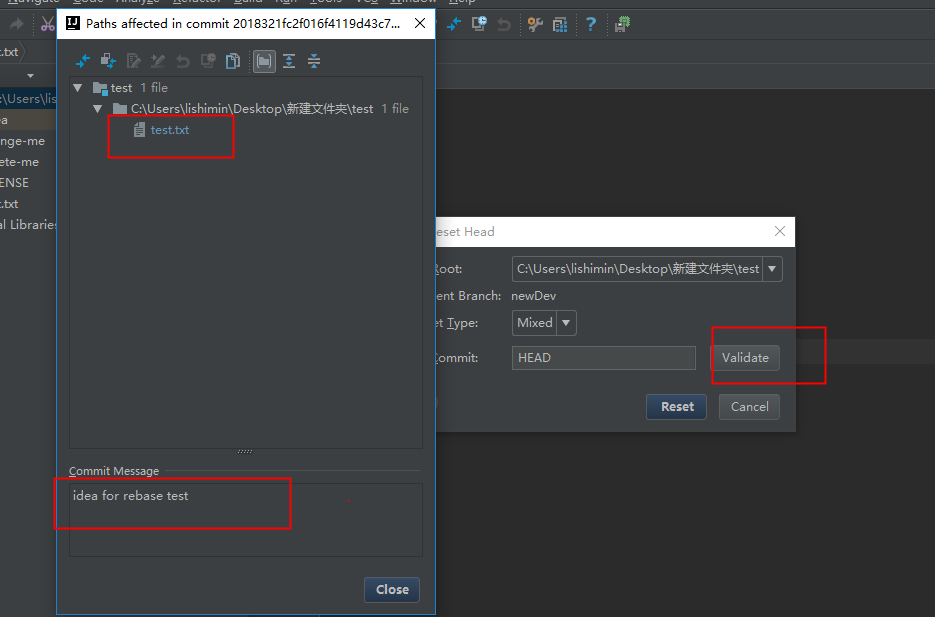
## 版本回退

* 右键项目 ，git -- repository – reset HEAD

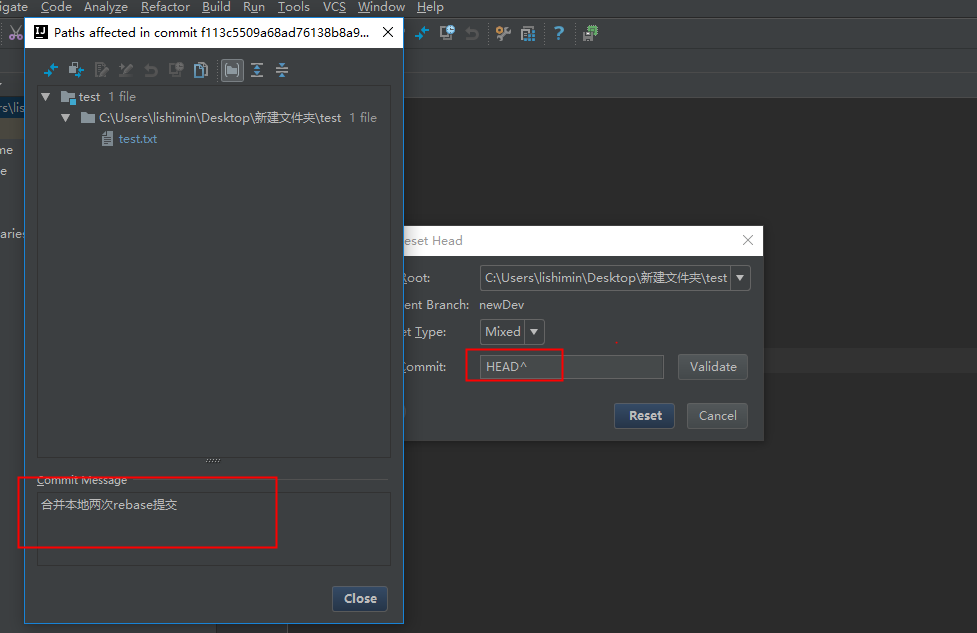


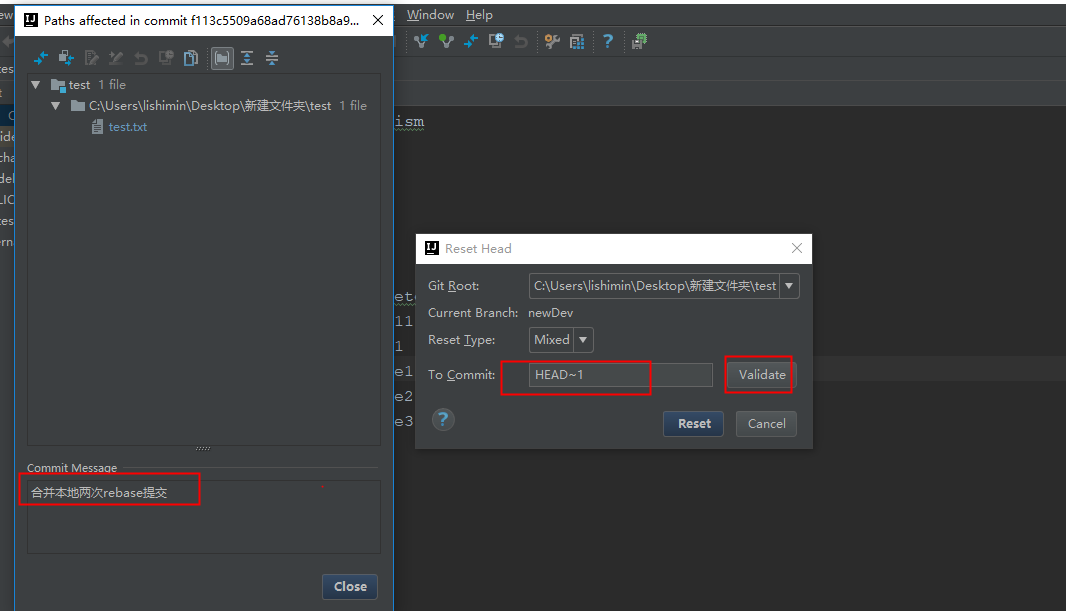


* 点击validate 可以查看要回退的版本提交信息，HEAD后面没东西，则指的是最近的一次提交；

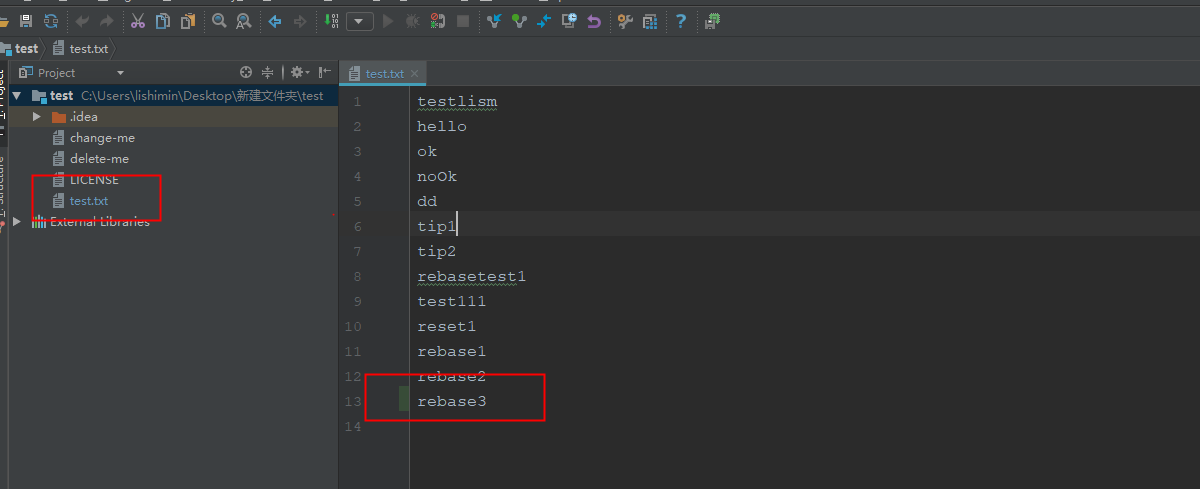


* 如果想回退最近一次提交的上一次提交，则可以HEAD后面加^ 或~1 ，与HEAD之间是没有空格的；

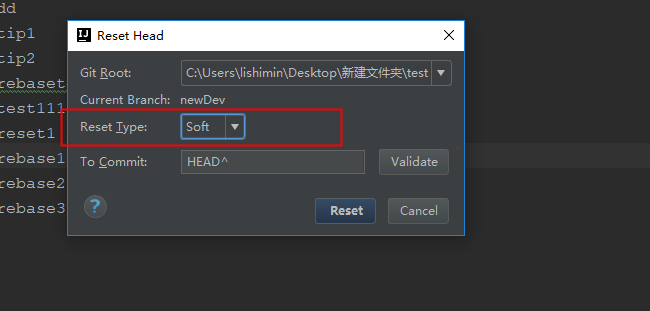




* 默认回退类型是mixed，即：把所有变更替换到工作区，如果要重新提交，则需要重新add,commit ; 命令行下两步，add , commit ; idea下一步操作，ctrl + K ,提交

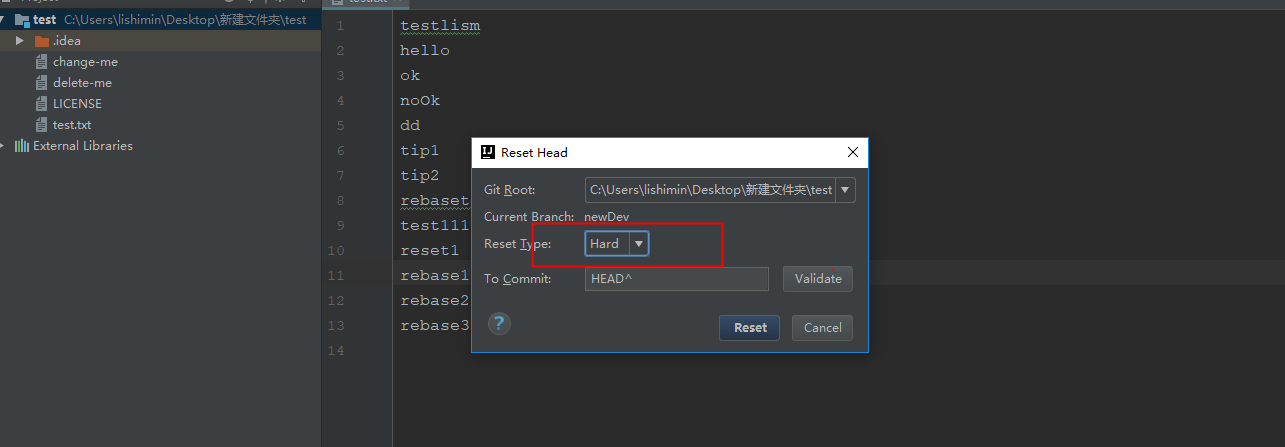


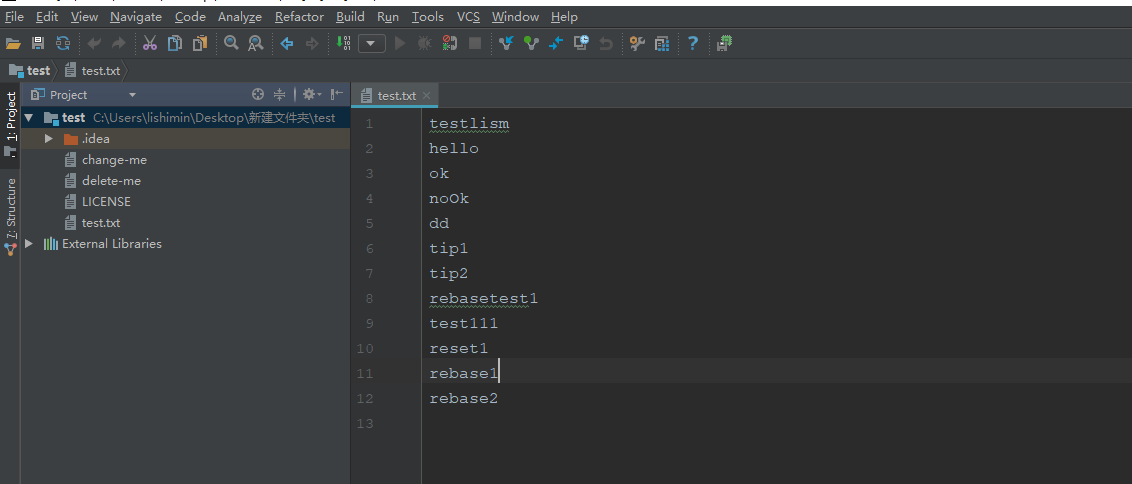
* 当选择参数为soft的时候，回退的结果是，把回退的变更替换到暂存区和工作区；



在命令行下的表现是不用重新add , 重新提交直接commit即可，idea 下的表现跟mixed参数下的形式一样，重新提交，直接一步commit 即可；

* 选择参数hard的时候，回退的结果是回退commit 之后的提交全部丢弃，



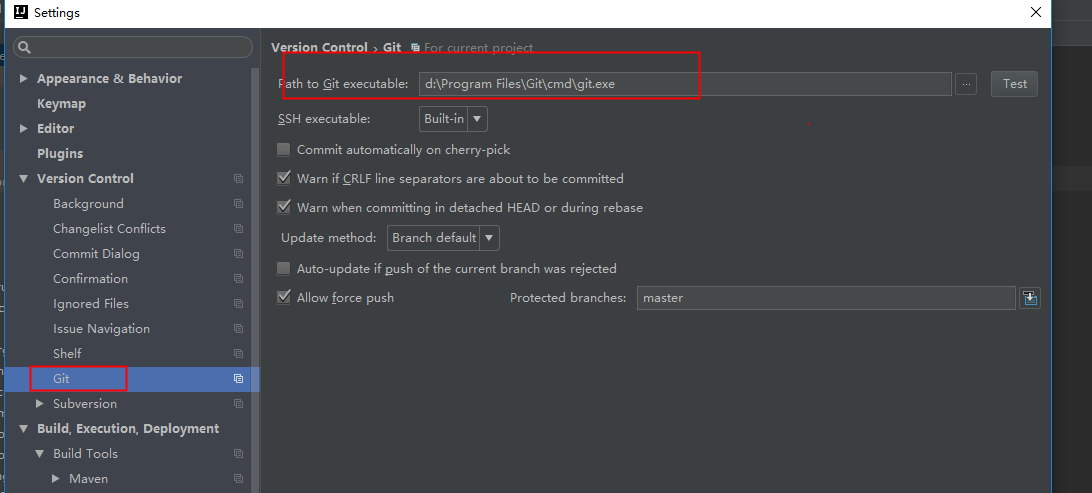


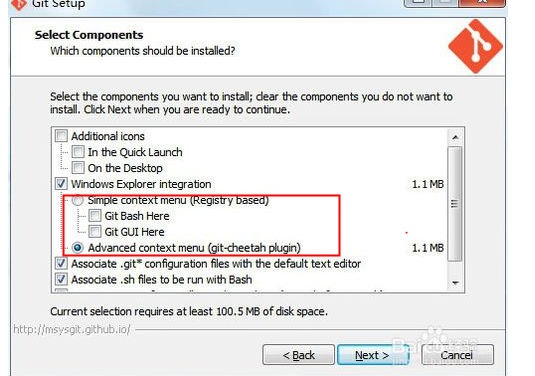
## 约定

* Push 代码之前先pull ;
* 本地个性化文件不push , 如：配置文件，依赖包之类；
* 代码中依赖的文件，别忘记提交；
* 回退之前一定看好回退的commit , 如果不确定，不要用hard回退，默认即可

## 遇到的问题

* 右键无法显示git 的一些操作：可以使用自己本地自定义安装的git命令，不用idea 内嵌的；注意：本地安装的时候自定义安装，把命令行的工具勾选安装；

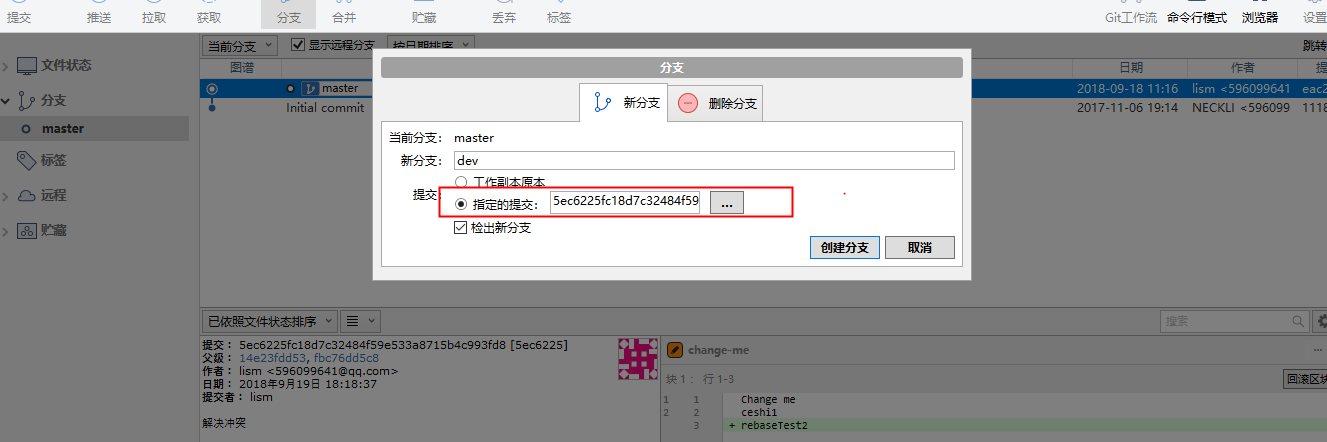




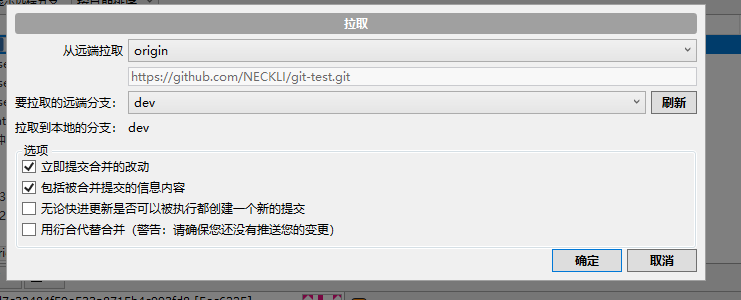
# SourceTree：使用技巧

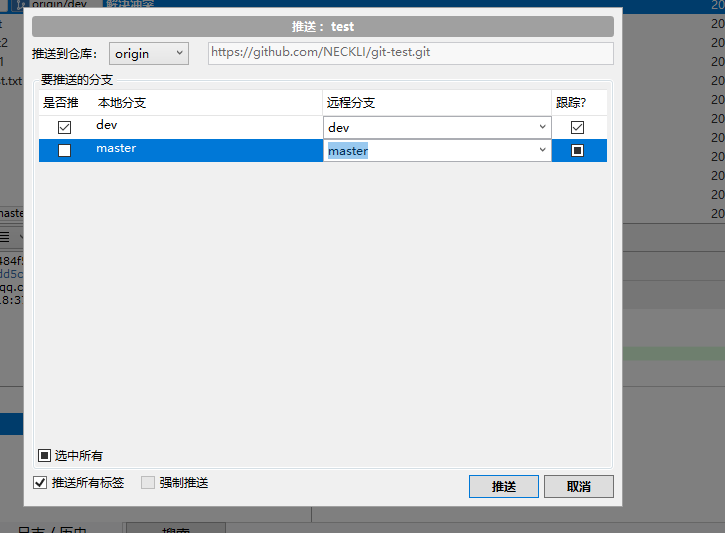
## 创建分支

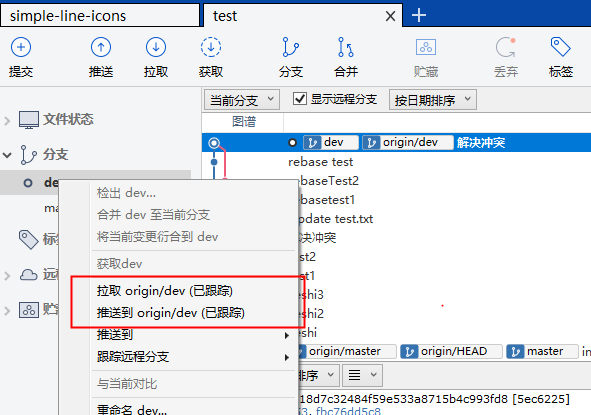
* 点击分支—>填写分支名称，指定从哪个分支上创建本地分支



* 关联远程分支，pull 的追踪，push 的追踪

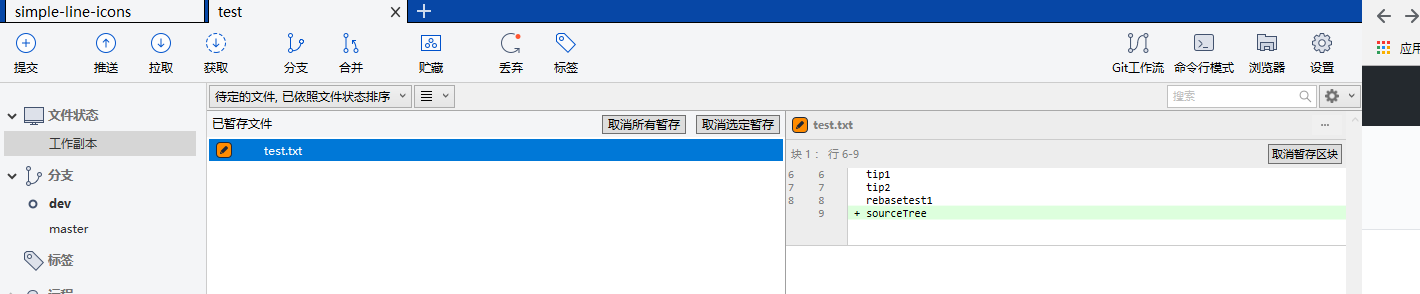


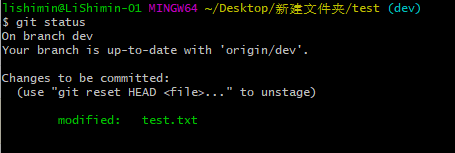




## 暂存文件

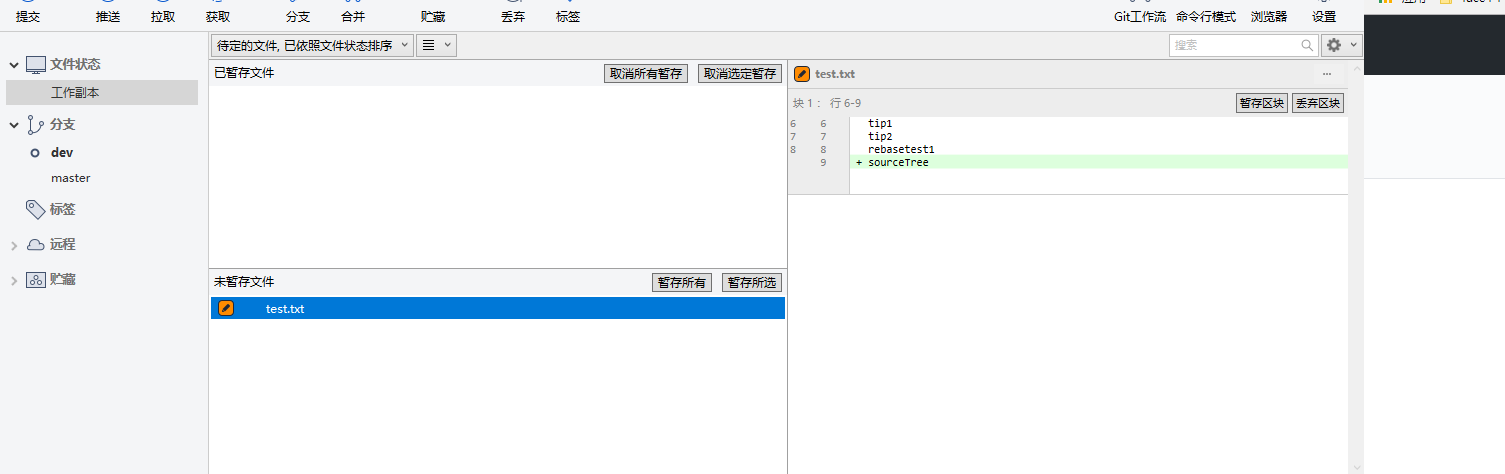
* 相当于git 命令的add

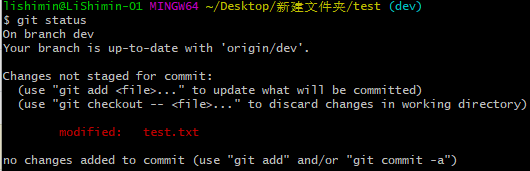




## 取消暂存

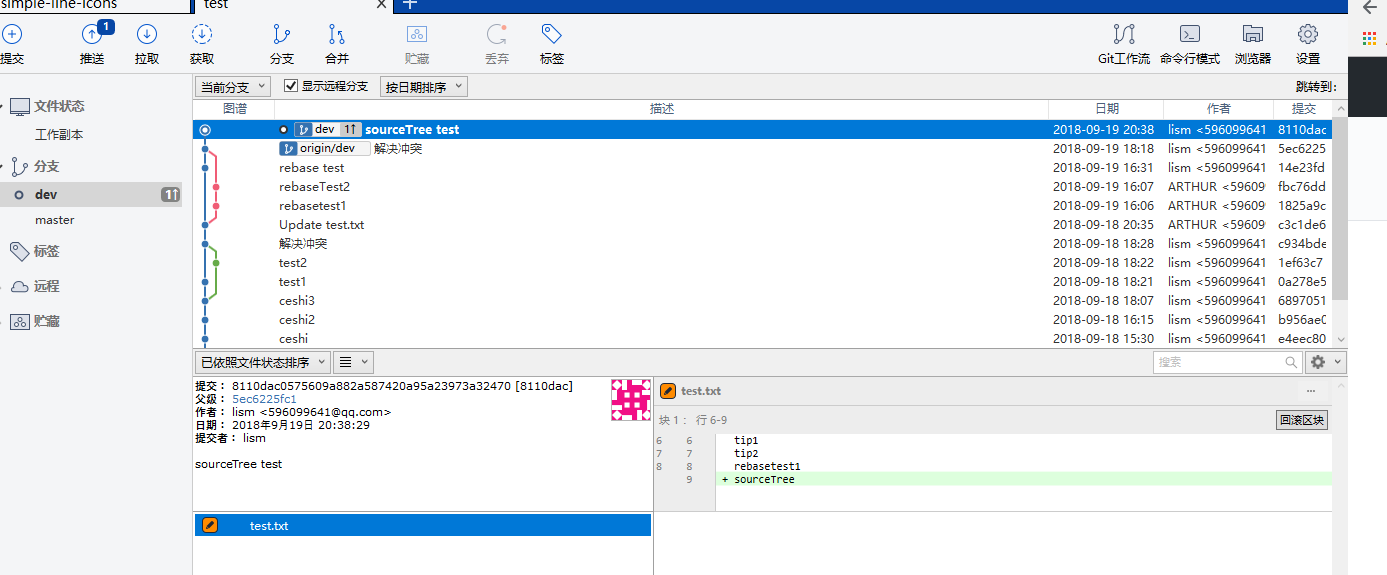
* 相当于git 命令的reset





## 提交

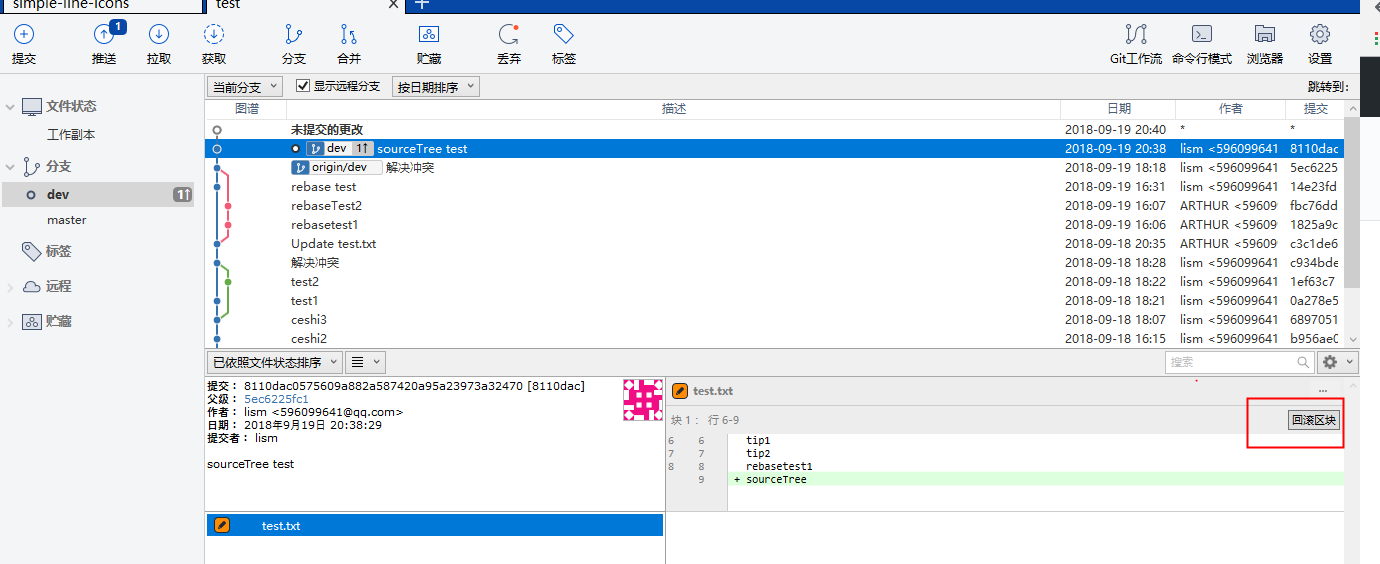
* 相当于git 里的commit命令

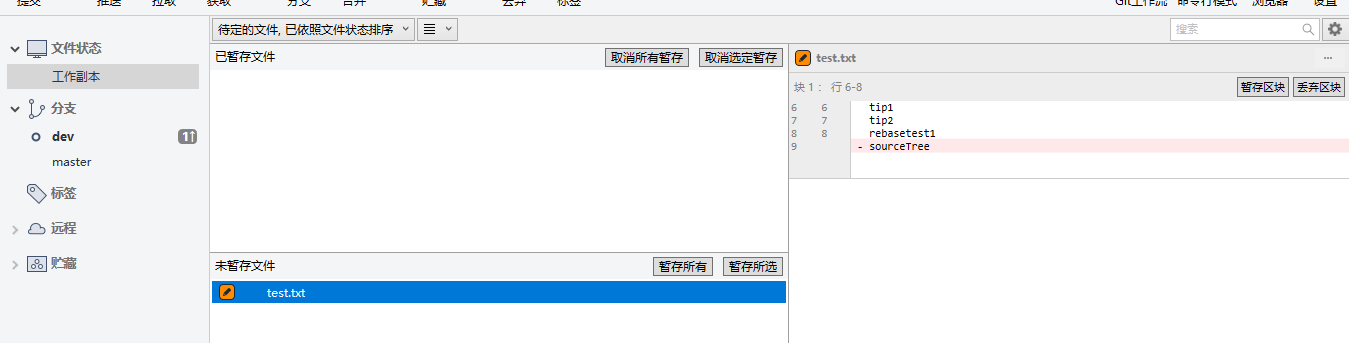




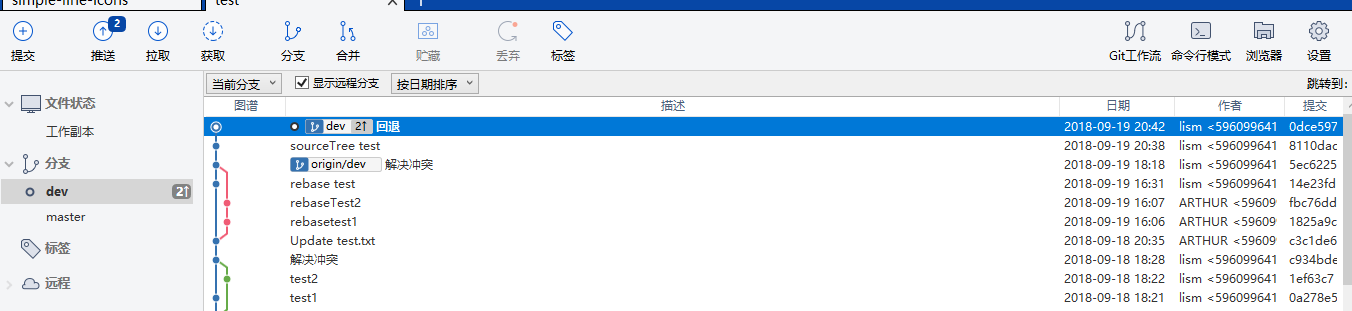
## 回滚

* 提交版本回退





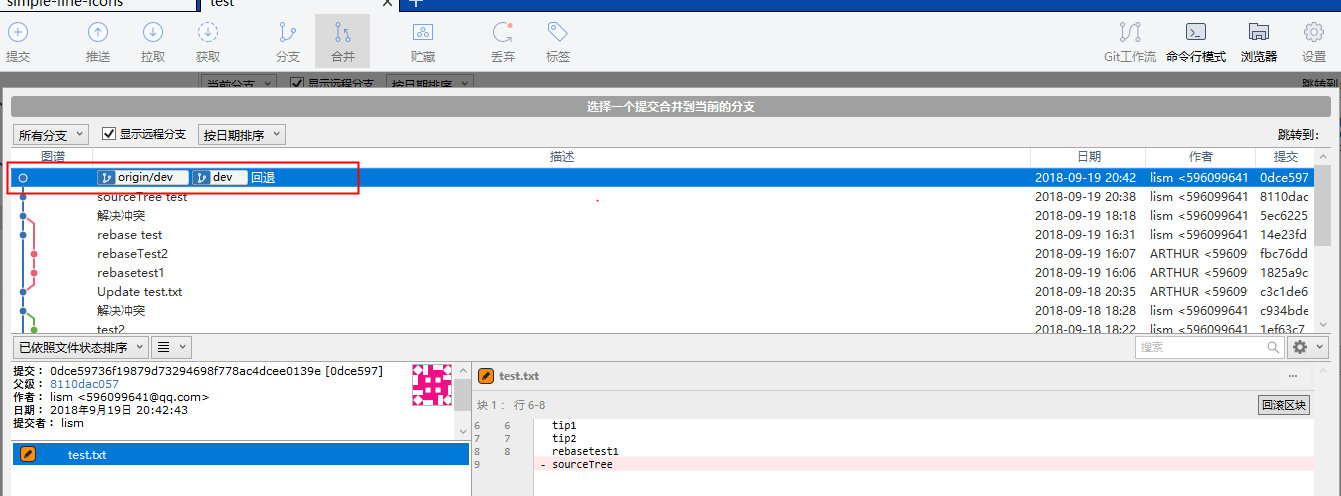
* 然后工作区，再提交

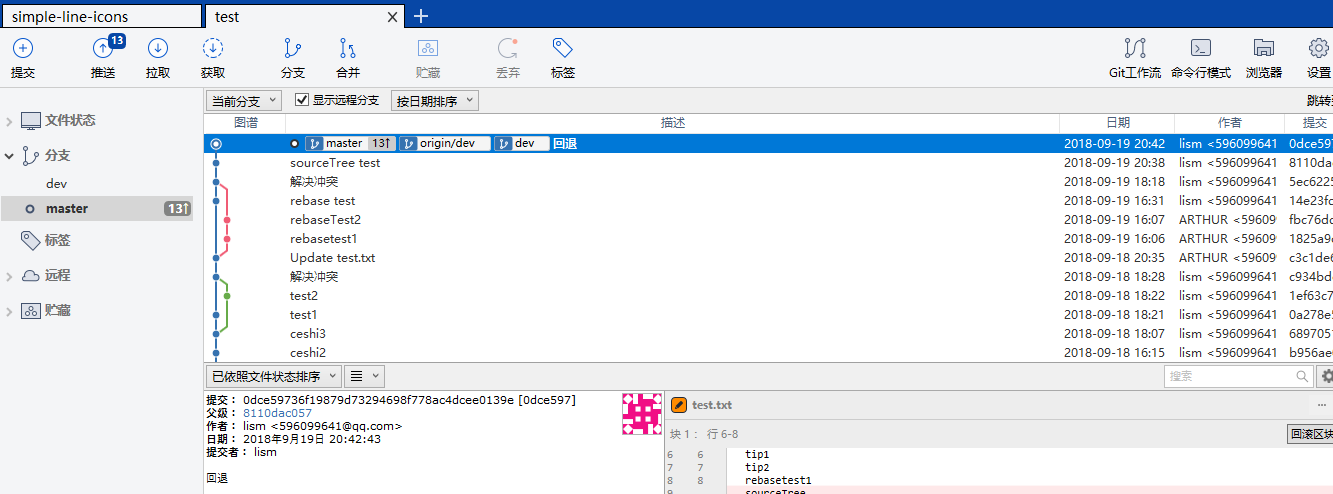


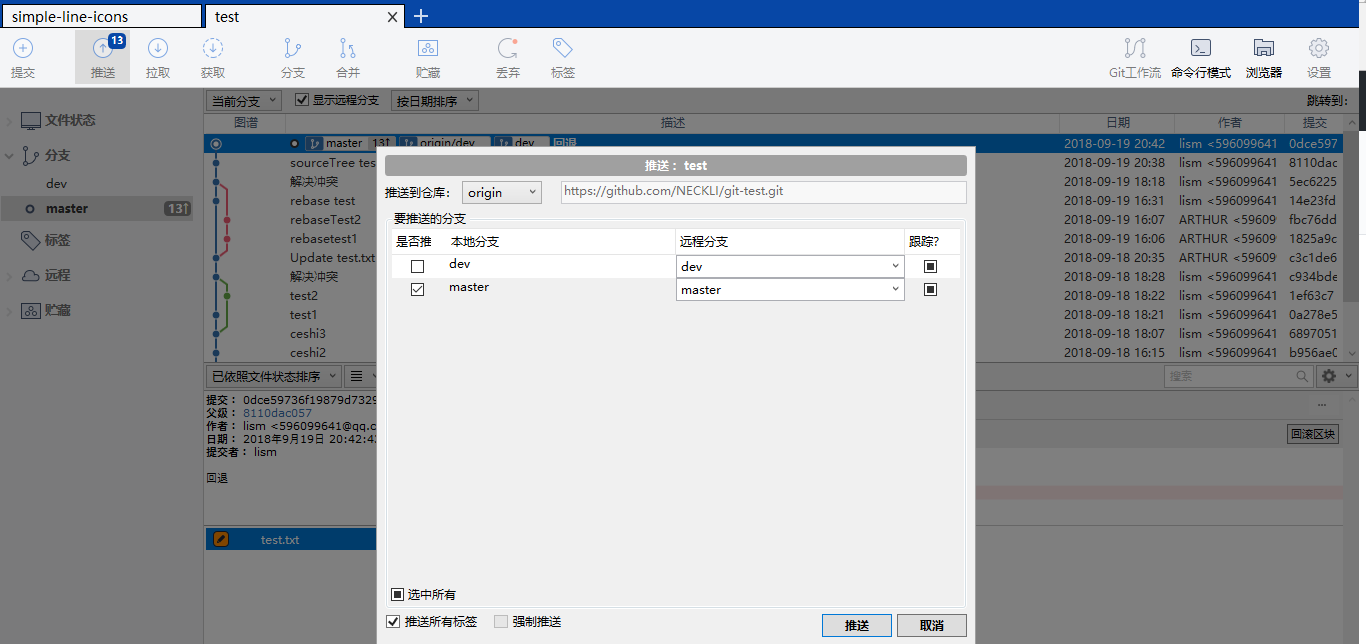


## 合并

* 先切换分支，然后点击合并







## 切换分支

* 在对应的分支上双击，则切换到当前分支；