# 6.15学习体会

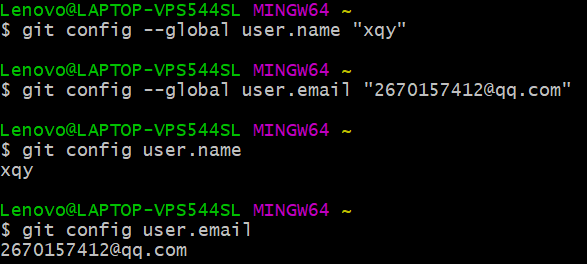
1. 学习内容
2. Git简介

2005年Linus花了两周时间自己用C写了一个分布式版本控制系统，并迅速成为最流行的分布式版本控制系统。在2008年，GitHub网站上线了，它为开源项目免费提供Git存储

1. Git安装

Git可以安装Linux macOS Solaris Windows Raspberry Pi Submit等操作系统上。

配置姓名和邮箱：



1. 创建版本库

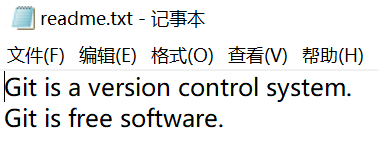
版本库：

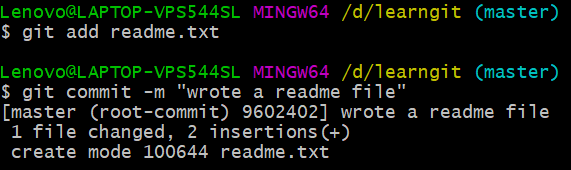
版本库又名仓库，英文名repository，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

1. 创建新目录

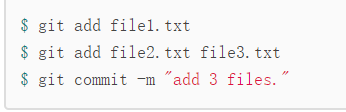


1. 把文件添加到版本库

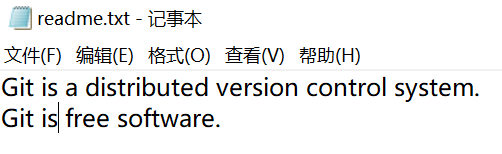




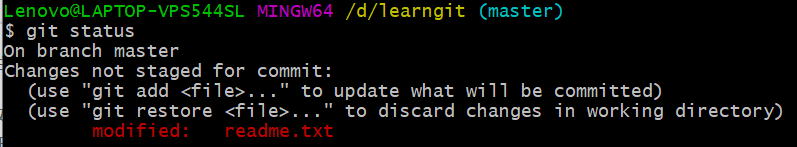
注：add可以添加许多文件 commit可以一次性提交多个文件



1. 修改文件

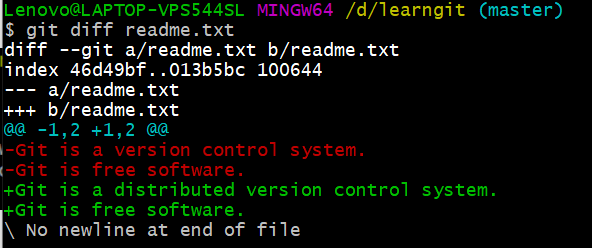


git status命令：掌握仓库实时状态

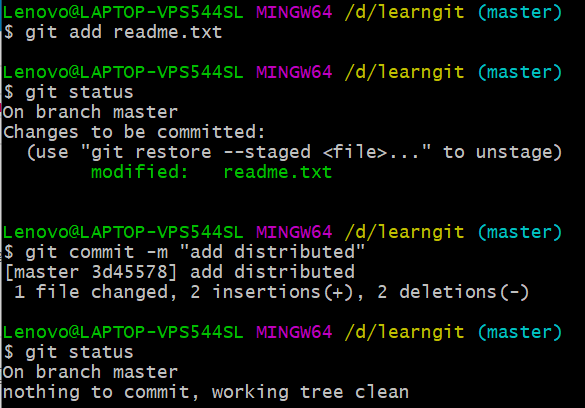


进一步了解被修改的内容

git diff命令：查看difference

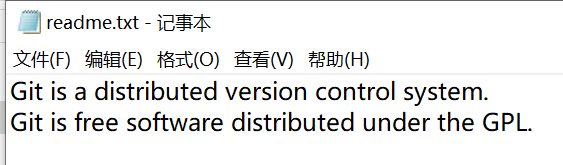


提交修改完的文件

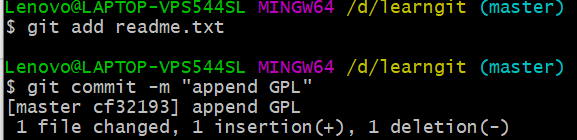


1. 版本回退

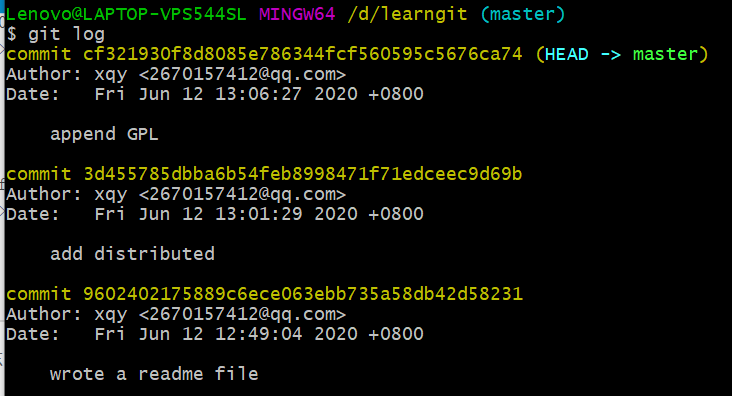
再次修改文件



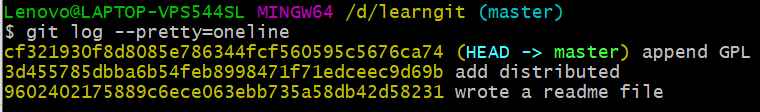
版本回退以commit为单位，如果误删文件，可以从最近一个commit恢复



git log命令：显示从最近到最远的提交日志

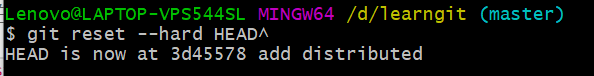


加上 --pretty=oneline简化输出

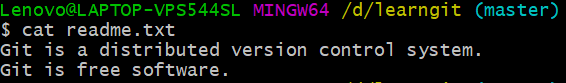


git中HEAD表示当前版本，HEAD^表示上一个版本，往上100个版本写作HEAD~100

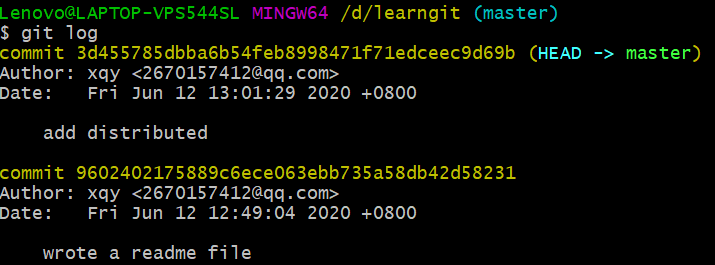
git reset命令：版本回退



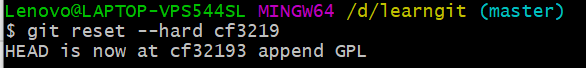
查看文件内容是否回退成功



查看当前版本库状态，最新版本已经看不到了



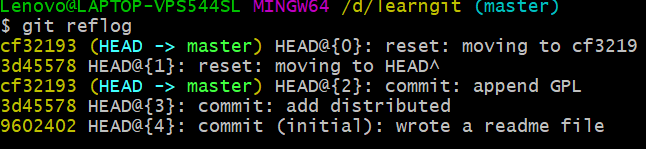
想要恢复append GPL就要找到它的commit id（只需要输入前几位，git会自动寻找）



git版本回退速度快，原因是，只改变了HEAD指针的指向

若无法得知commit id则使用git reflog指令

git reflog：记录每一次命令



1. 工作区和暂存区
2. Git和其他版本控制系统如SVN的一个不同之处就是有暂存区的概念。
3. 工作区：电脑里能看到的目录，例如learngit

版本库：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



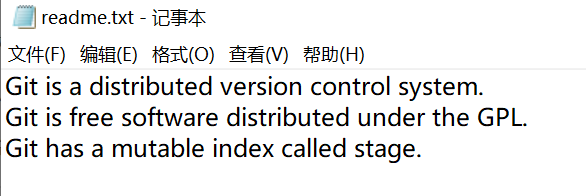
把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

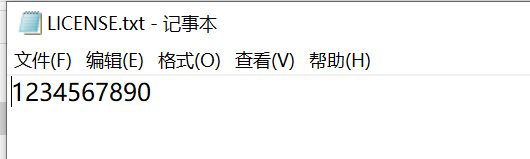
第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

（3）操作

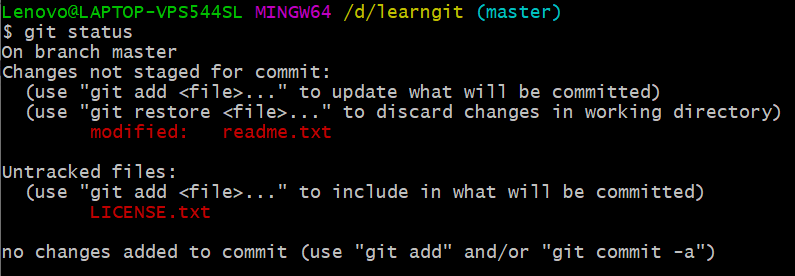
对readme再次进行修改



在工作区新增一个LICENSE文本文件

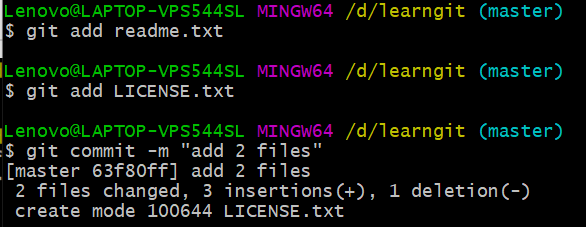


git status查看状态

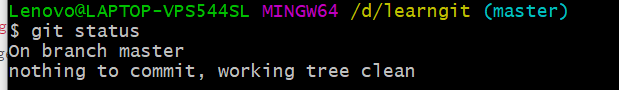


readme.txt被修改了，而LICENSE还从来没有被添加过，所以它的状态是Untracked

添加两个文件



再次查看

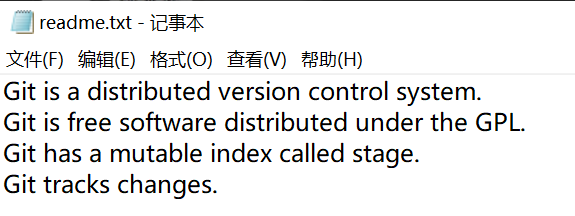


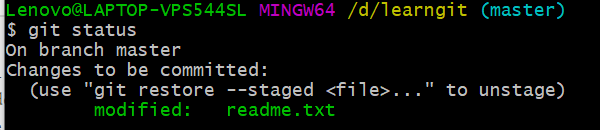


1. 管理修改

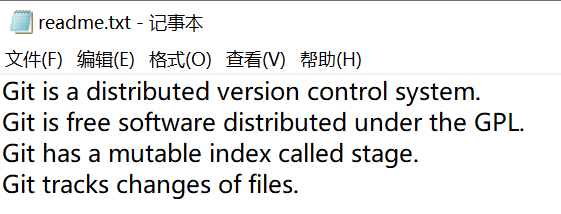
Git比其他版本控制系统设计得优秀，因为Git跟踪并管理的是修改，而非文件。

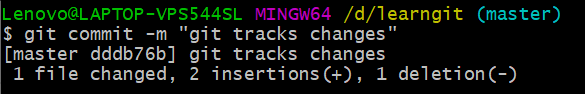
再次修改添加readme文件



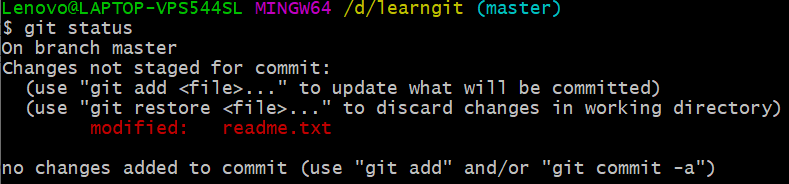


然后再修改然后提交



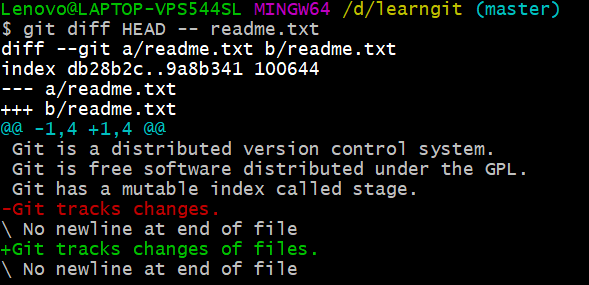


查看状态



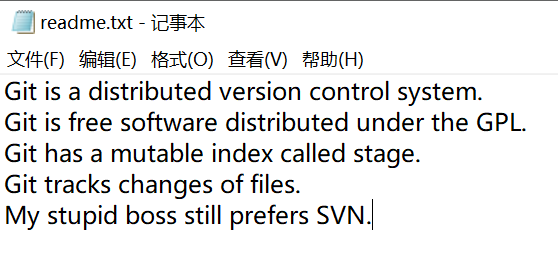
第二次修改没有提交成功，是因为第二次没有add

使用git diff HEAD -- readme.txt查看工作区和版本库里面最新版本的区别



所以，可以把每次的修改，都先add，然后一起commit，系统自动合并所有的修改

1. 撤销修改



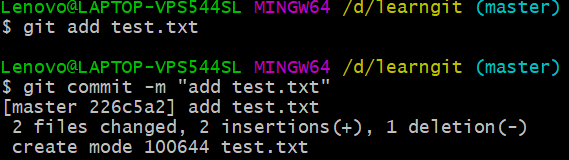
（1）场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

（2）场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

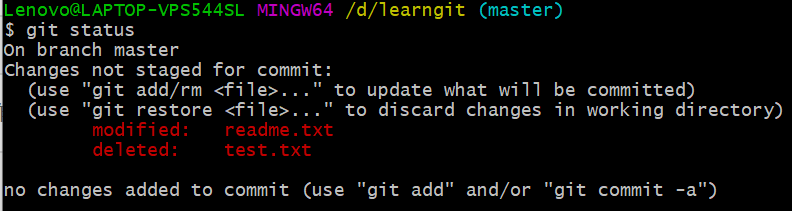
（3）场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/896043488029600/897013573512192)一节，不过前提是没有推送到远程库。

1. 删除文件

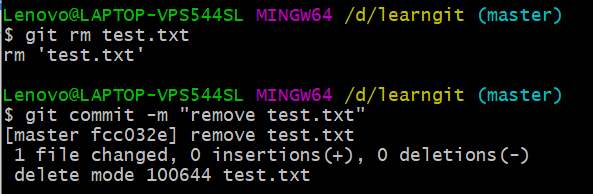
先添加一个新文件test.txt到Git并且提交



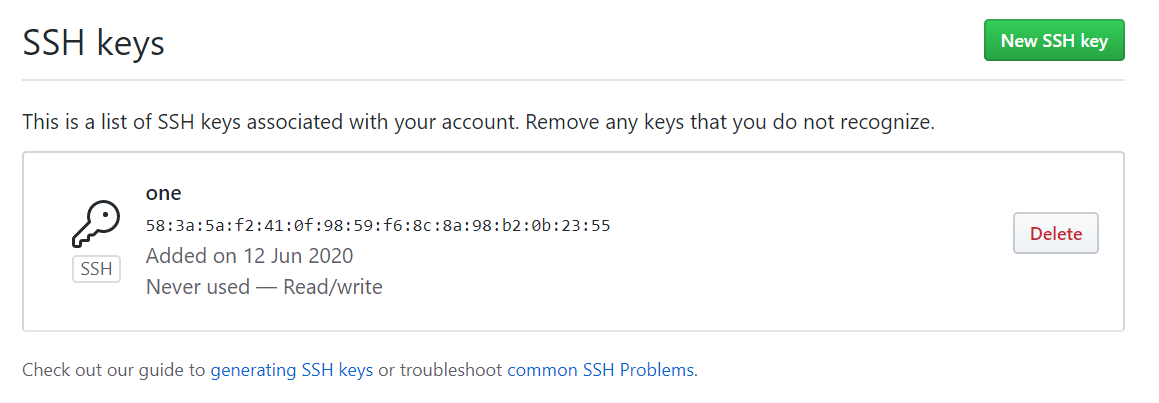
删除test文件



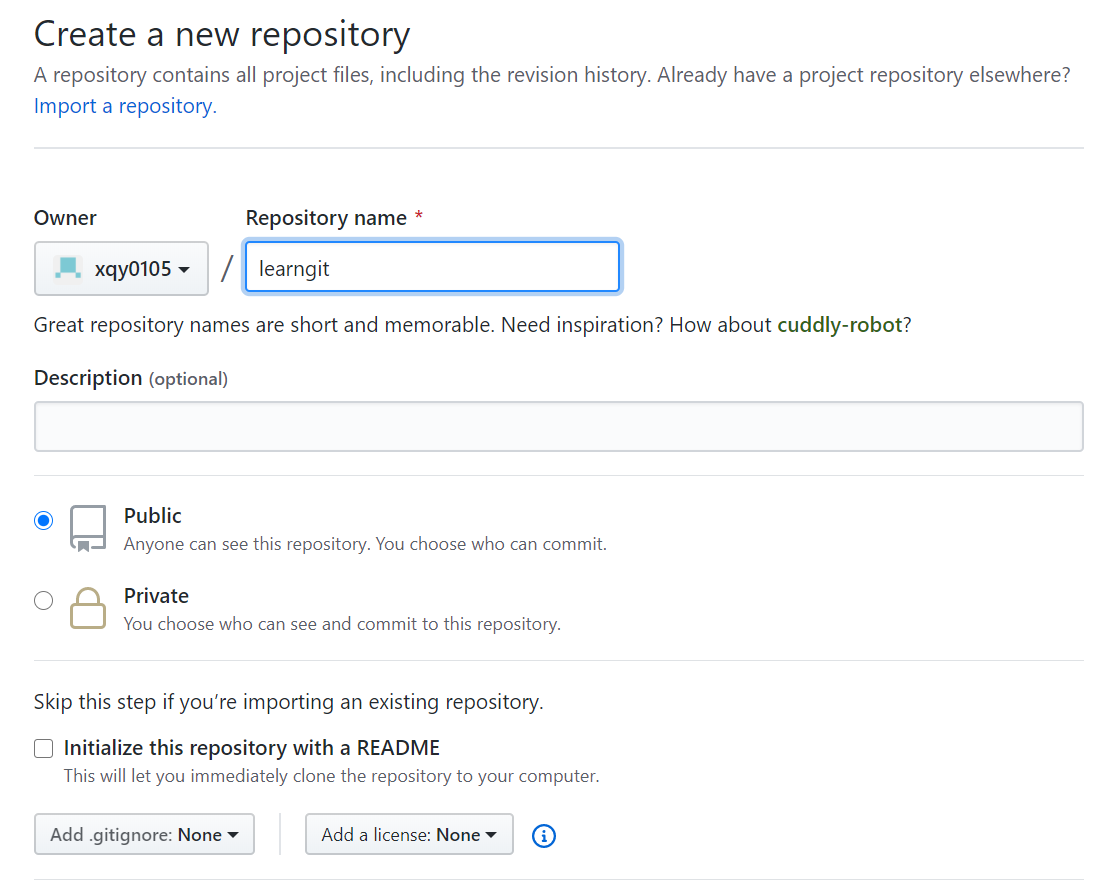
从版本库中删除该文件，那就用命令git rm删掉，并且git commit



1. 远程仓库



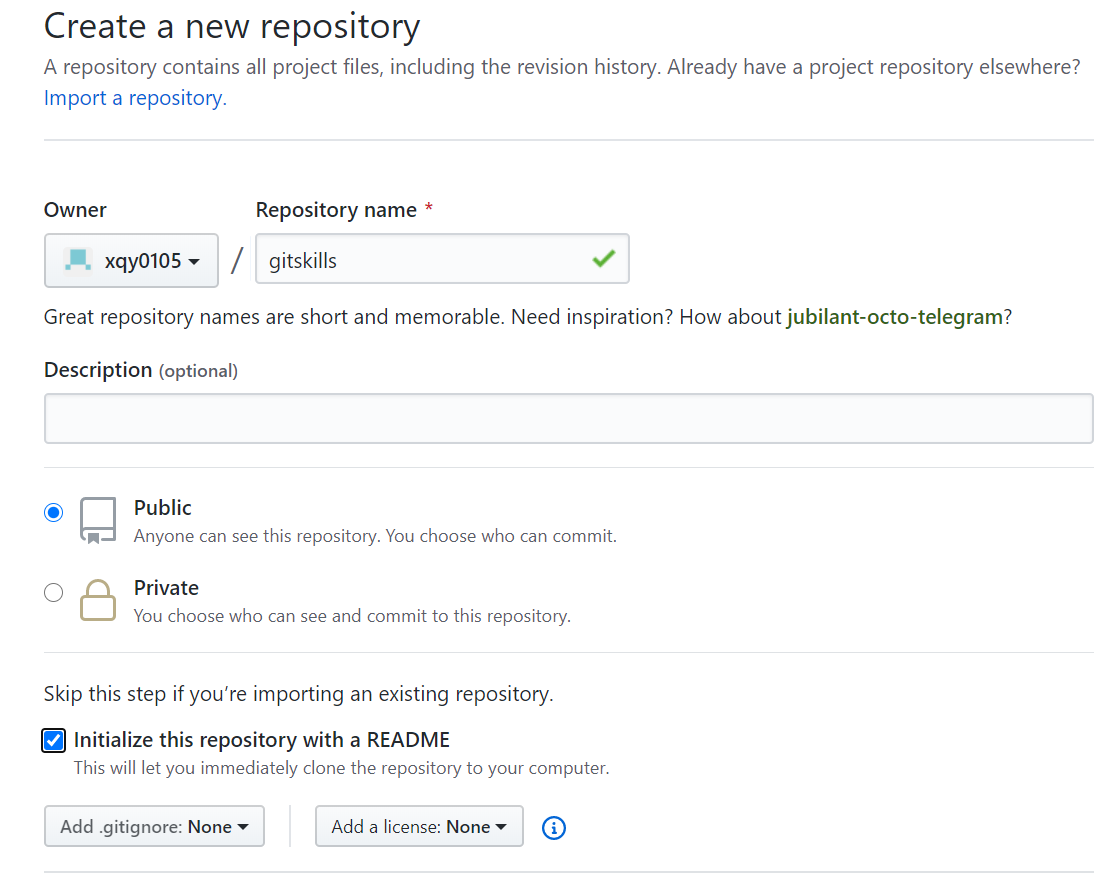
1. 添加远程库



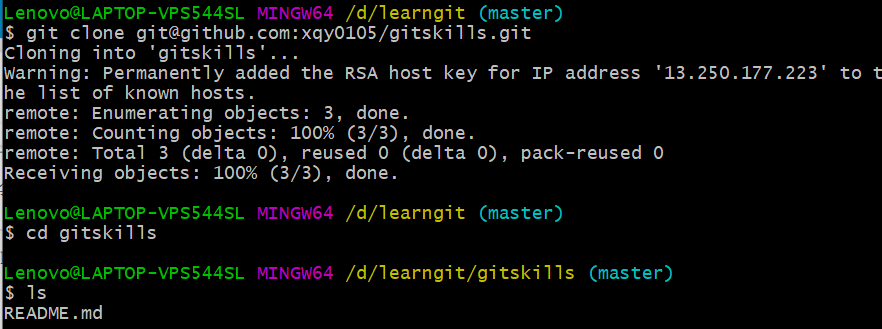
本地做提交之后

使用git push origin master操作上传到github

1. 从远程库克隆



使用git clone克隆一个本地库



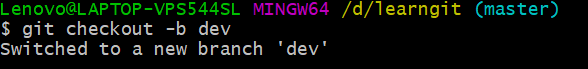
（Git支持多种协议，包括https，但ssh协议速度最快）

1. 分支管理

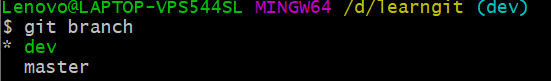
在各个分支上工作，互不干扰，可以通过创建分支，合并分支删除分支的方法进行同时工作，最终合并为一个分支

（1）创建并合并分支

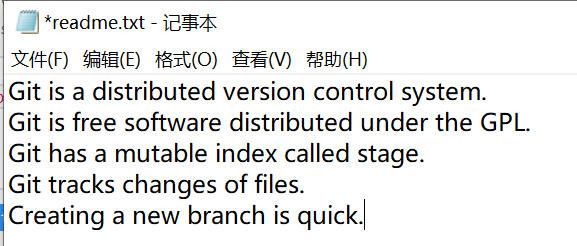
首先创建并切换dev分支



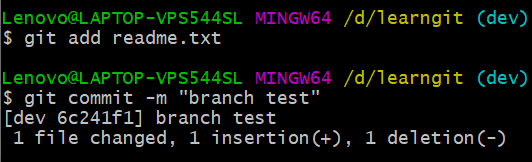
用git branch命令查看当前分支



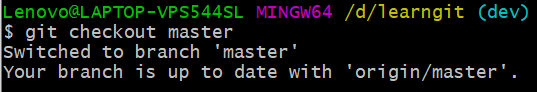
修改readme.txt



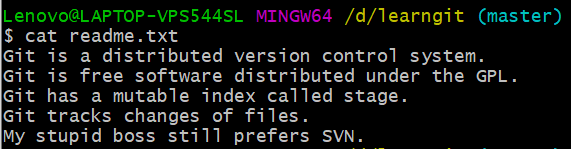
在dev分支上提交文件



切换回master分支

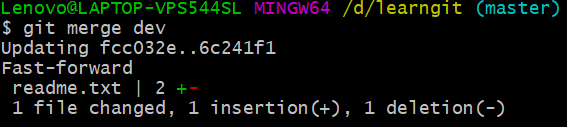


查看readme文件，发现内容并没有添加

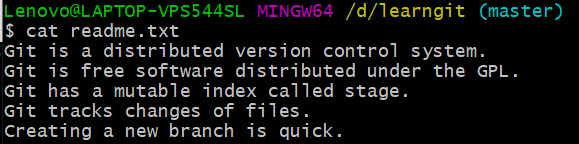


将dev分支上的内容合并到master分支上

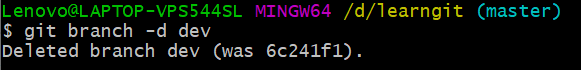
git merge命令用于合并指定分支到当前分支



再次查看readme



删除dev分支



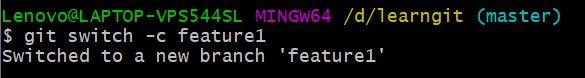
切换分支还可以使用

git switch -c dev

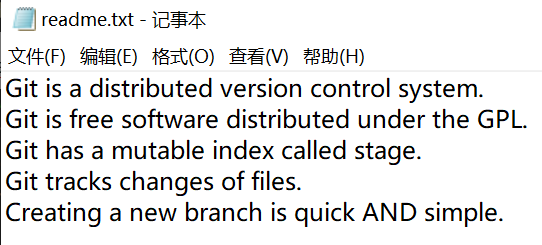
git switch master

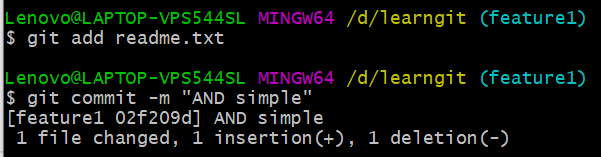
1. 解决冲突

创建新分支feature1

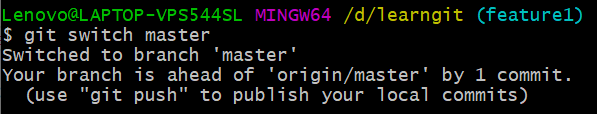


修改并提交readme文件

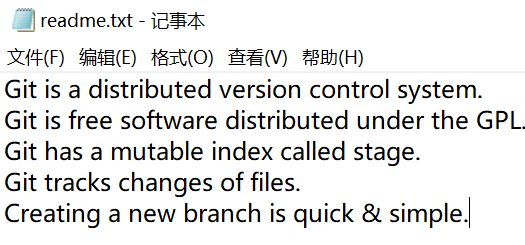




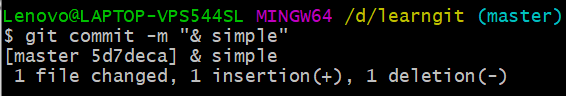
切换到master分支



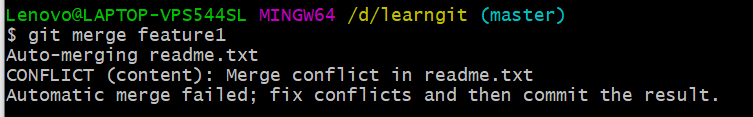
在master分支上修改readme



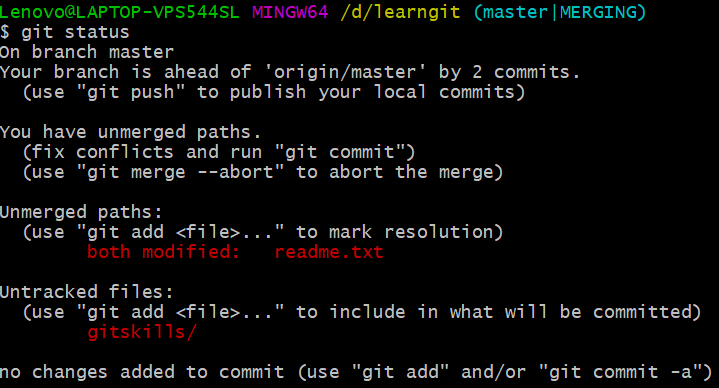
提交



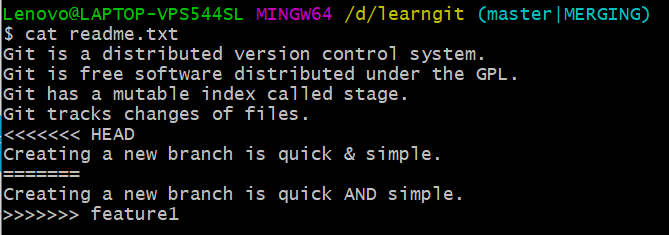
产生冲突无法合并



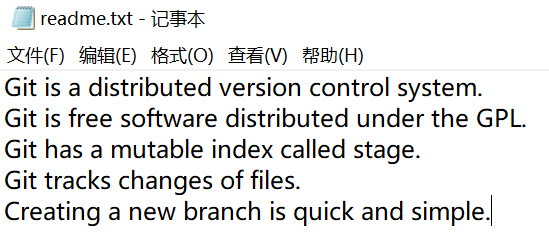
用git status查看冲突文件



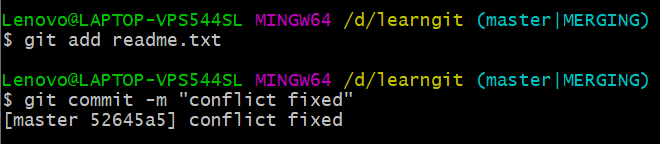
查看文件内容



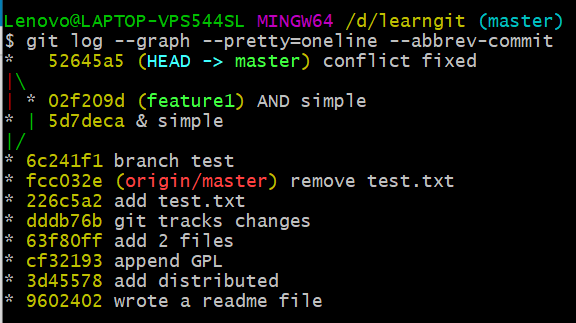
再次修改之后提交



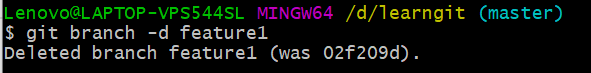
冲突解决



使用git log查看合并情况



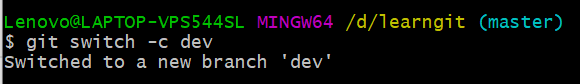
删除feature1分支



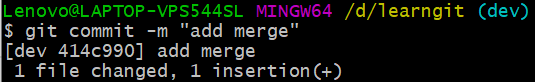
1. 分支管理

原则：master一般只用来发布新版本，不在上面工作，每个人在个人分支上工作，最后都提交在dev上，再最后合并到master分支上

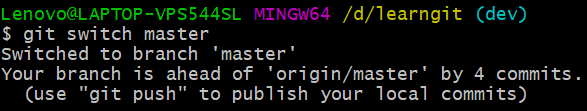
首先创建并切换到dev分支



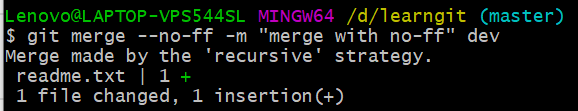
修改readme.txt文件，并提交一个新的commit



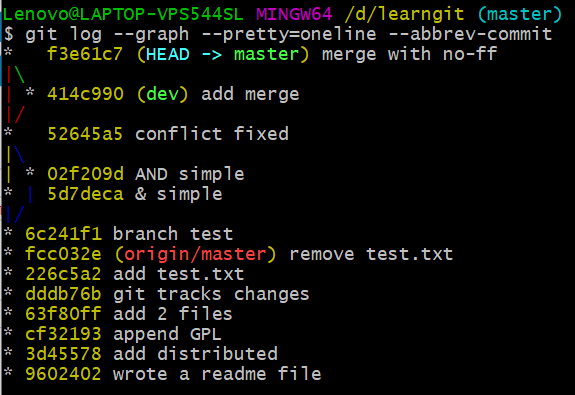
切换回master



准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward



用git log查看分支历史



结论：合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

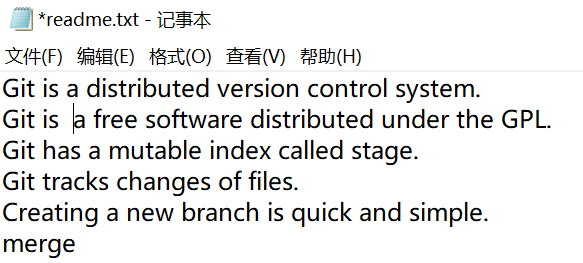
1. Bug分支

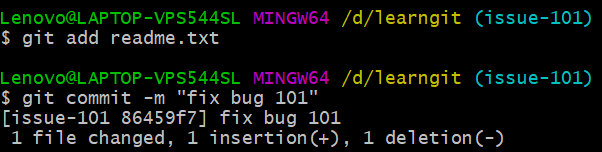
每个bug都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除

如果在dev分支上的工作还没提交，要建立一个修复101的分支issue-101

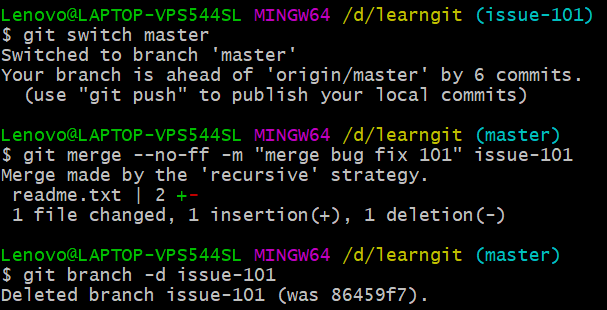
使用stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作

修改readme中的bug并提交





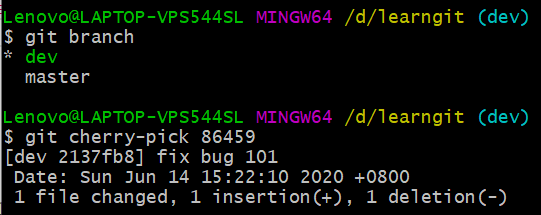
切换到master分支，完成合并，删除issue分支



回到dev分支继续工作，用git stash list查看stash内容

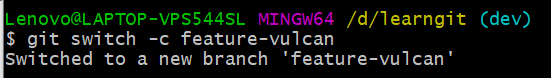
用git stash pop恢复内容

在master分支上修复的bug，想要合并到当前dev分支，可以用git cherry-pick <commit>命令，把bug提交的修改“复制”到当前分支，避免重复劳动。

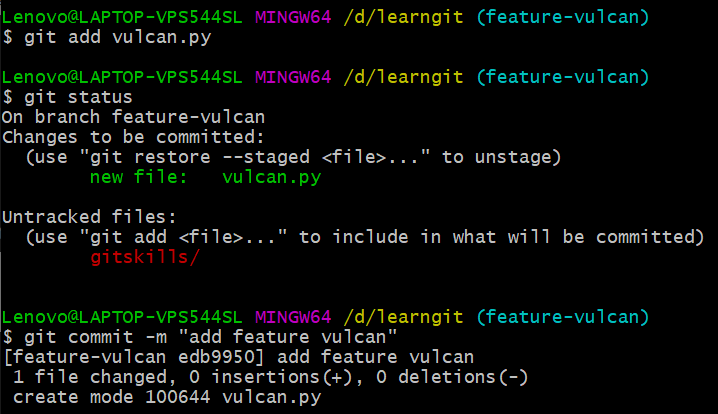


1. Feature分支

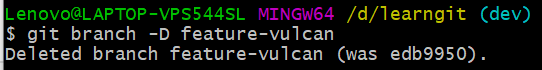
建立feature分支



新建vulcan文件并提交

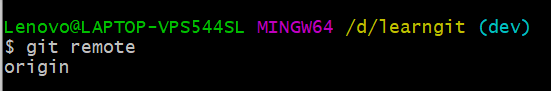


切回dev准备合并，但若项目临时不能做，因为feature分支未合并所以强行删除需要用D

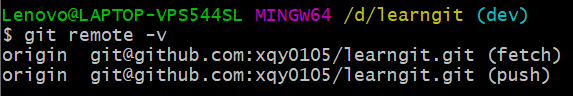


1. 多人协作

用git remote查看远程库的信息



用git remote -v显示更详细的信息



（显示了可以抓取和推送的origin的地址）

推送分支：把该分支上的所有本地提交推送到远程库

master分支是主分支，因此要时刻与远程同步；

dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

多人协作的工作模式：

1.首先，可以试图用git push origin <branch-name>推送自己的修改；

2.如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

3.如果合并有冲突，则解决冲突，并在本地提交；

4.没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>。

1. Rebase

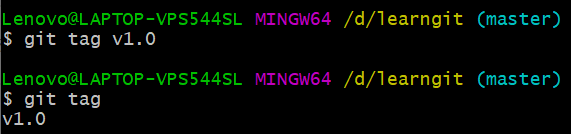
使用git rebase把本地未push的分叉提交历史整理成直线

1. 标签管理

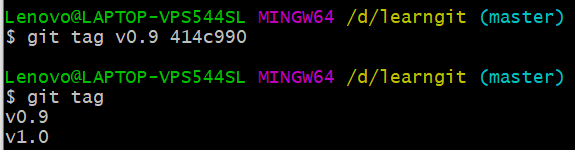
标签就是一个便于记忆和管理的代号，与commit对应

1. 创建标签

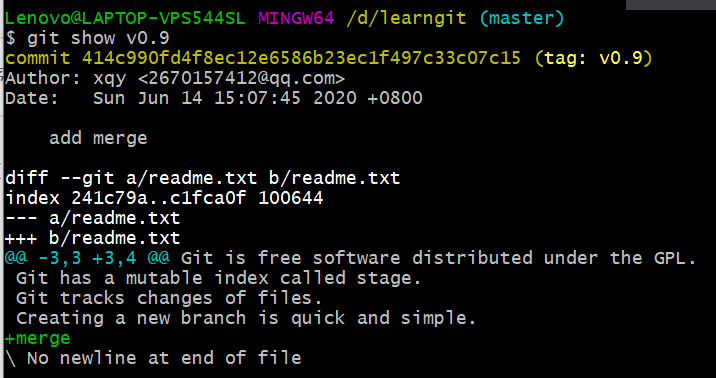
创建和查看标签（默认标签是打在最新的commit上的）



还可以通过历史提交的commit id打上标签



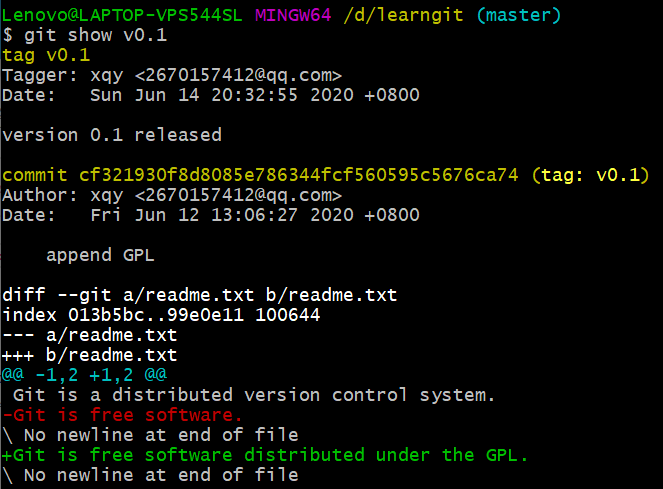
查看标签信息



创建带有说明的标签，用-a指定标签名，-m指定说明文字



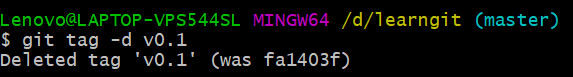
查看v0.1



（标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签）

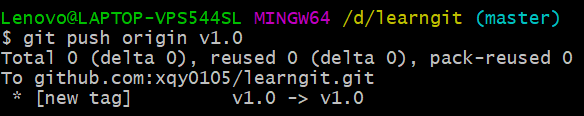
1. 操作标签

标签删除

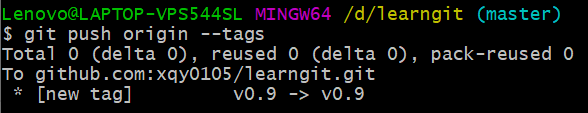


创建的标签都只存储在本地，不会自动推送到远程。所以，打错的标签可以在本地安全删除

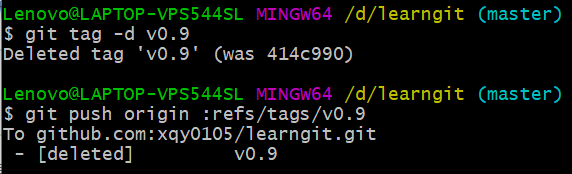
推送某个标签到远程



一次性推送全部尚未推送到远程的本地标签



如果标签已经推送到远程，要删除远程标签，需要先从本地删除，然后再push



1. 使用github

在GitHub上，可以任意Fork开源仓库；

自己拥有Fork后的仓库的读写权限；

可以推送pull request给官方仓库来贡献代码。

1. 忽略特殊文件

忽略文件的原则是：

（1）忽略操作系统自动生成的文件，比如缩略图等；

（2）忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的.class文件；

（3）忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

如果想添加一个文件到Git，但发现添加不了，原因是这个文件被.gitignore忽略了

可以用 -f强行添加git add -f App.class

用git check-ignore命令检查.gitignore哪个规则写错了

1. 设置别名

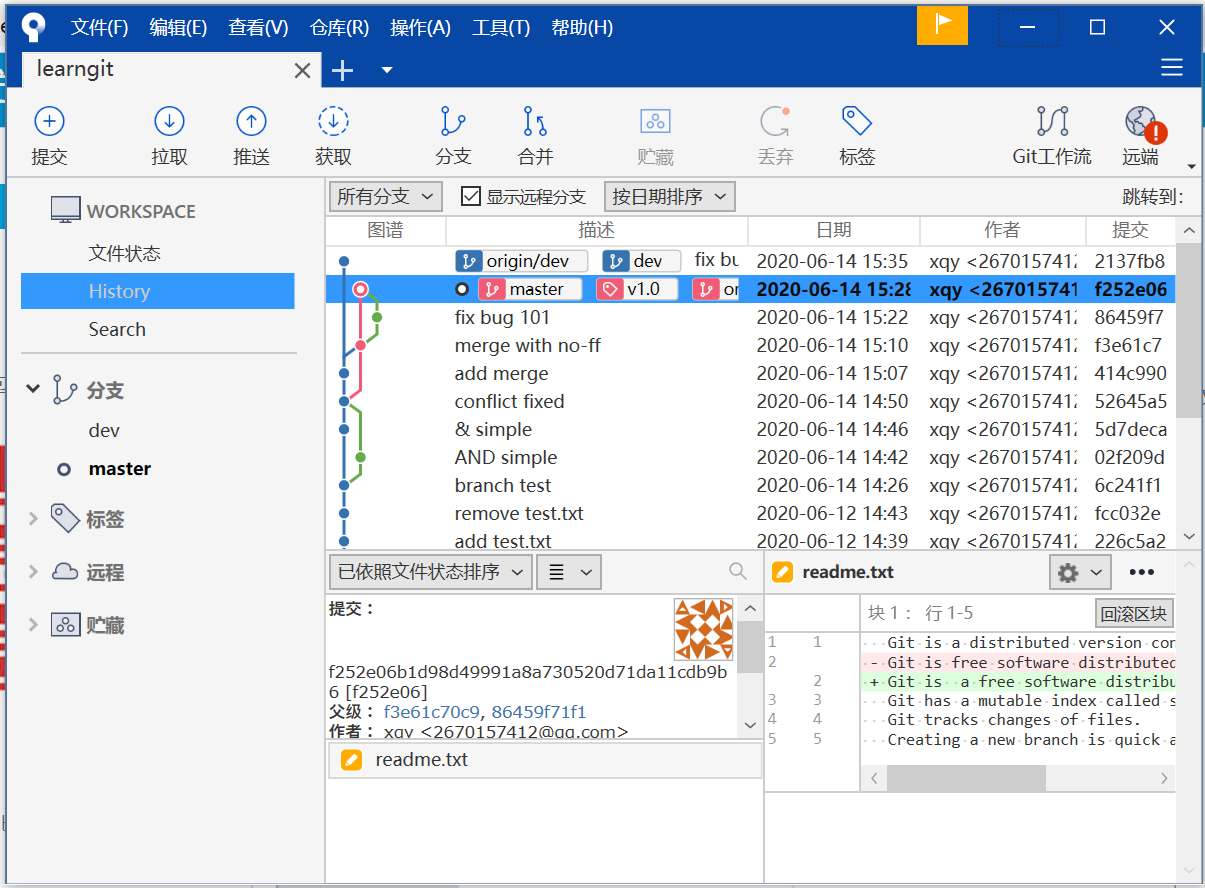
git config --global alias.st status（简写）

可以git config --global alias.last 'log -1'（显示最近一次的提交）

配置文件的位置 .git/config

可通过直接修改文件中alias里面的内容来修改配置

1. 使用source tree（免费Git图形界面工具）



以图形形式操作git

1. 存疑

1.版本回退时--hard参数有何意义？

2.分支和HEAD的概念