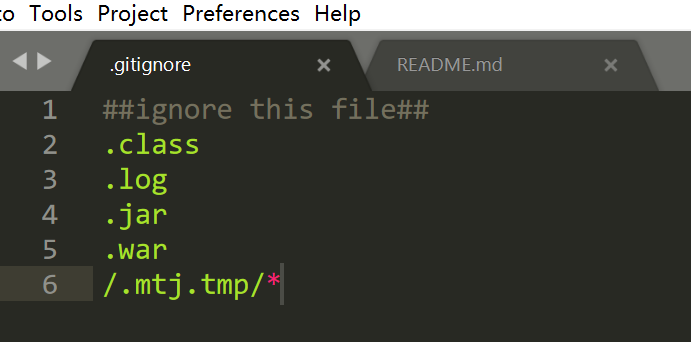
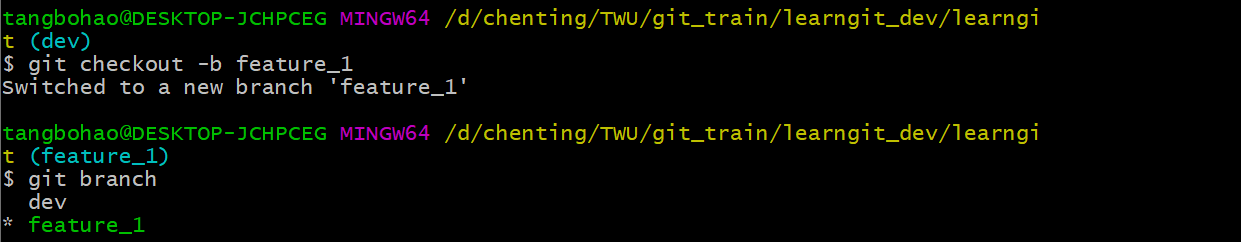
1. 本地初始化名为learngit的仓库，并配置gitignore文件，忽略掉所有 .class, .log, .jar, .war文件以及.mtj.tmp文件夹下的所有内容。将该本地仓库推送到个人github中。
2. git init
3. git remote add origin [git@github.com:tingzi123/learngit.git](mailto:git@github.com:tingzi123/learngit.git)
4. touch .gitignore



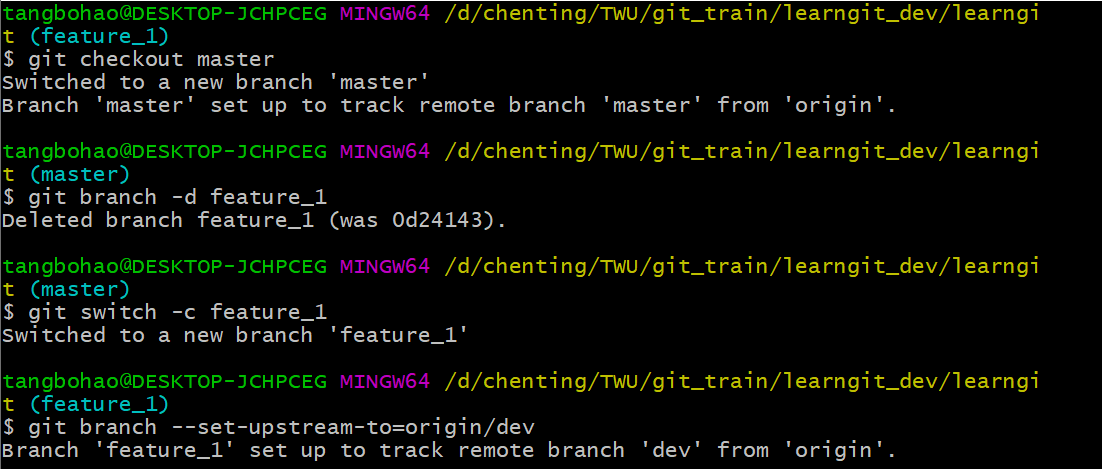
1. 仅执行 clone learngit 仓库的 dev 分支到本地。分别使用 switch和checkout命令在本地创建 feature-1 分支并对应到远程该分支。
2. git clone -b dev [git@github.com:Tingzi123/learngit.git](mailto:git@github.com:Tingzi123/learngit.git)



3）git branch --set-upstream-to=origin/dev

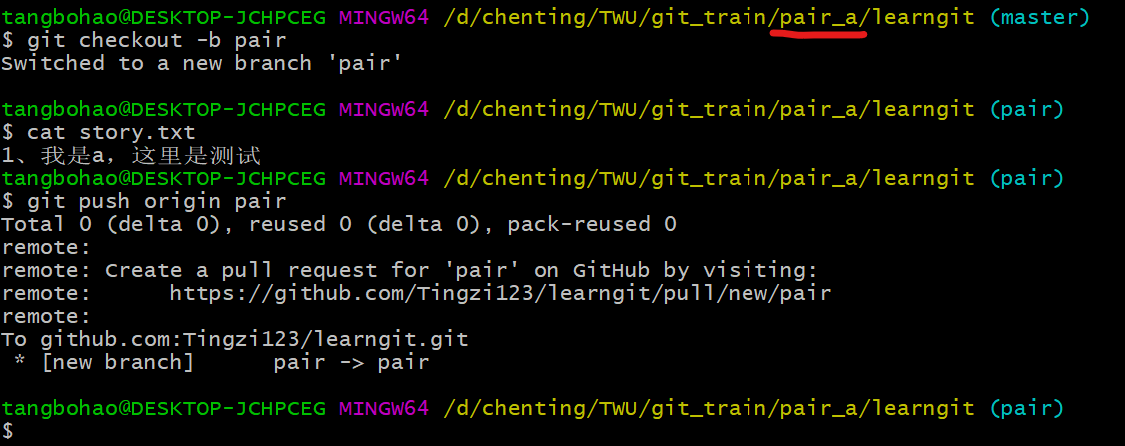


4）用switch创建分支

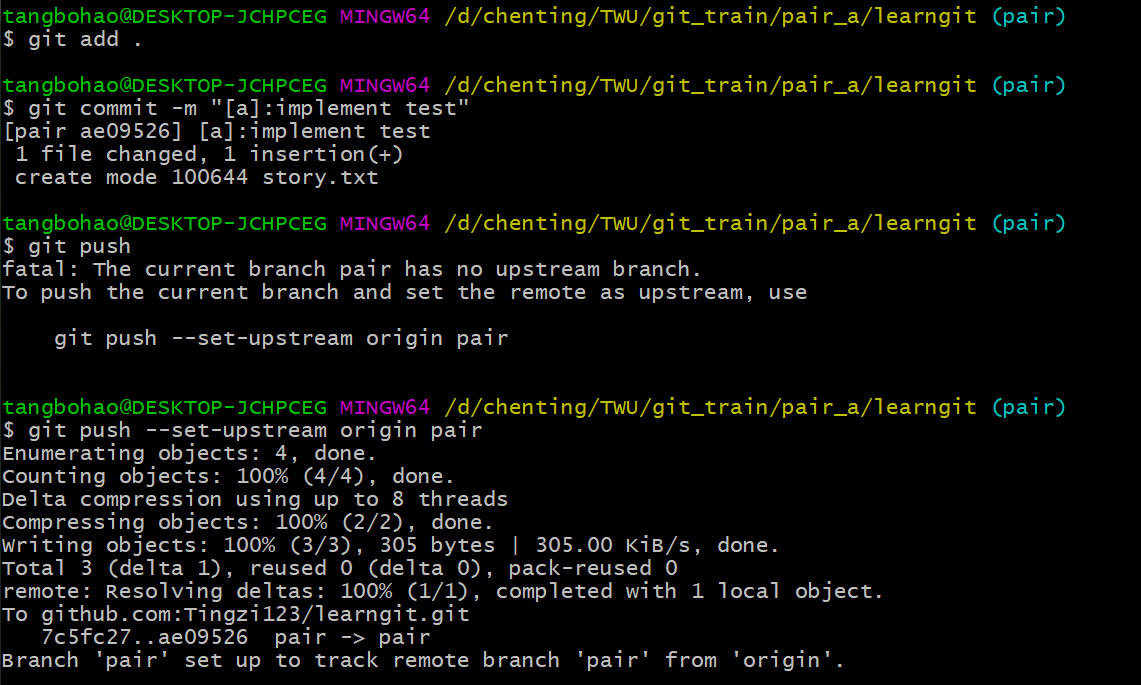


1. 当不在同一个地方的两个开发人员需要pair着完成同一个故事卡内容的开发，如何交替使用每个人的电脑进行协同开发？请写出这个过程中所涉及的 git 命令。例如：在 tdd 的过程中，一个人先在自己的机子上写了测试，然后另一个人需要在自己机子上依据测试来写实现。(提示：可以新建一个协同开发分支，分别提交然后拉取的方式，进行协调工作)

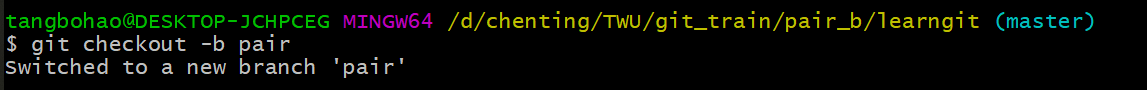
假设有A和B两个人进行开发，A的工作目录为pair\_a，B的工作目录为pair\_b，A负责写测试，B负责依据测试来写实现。

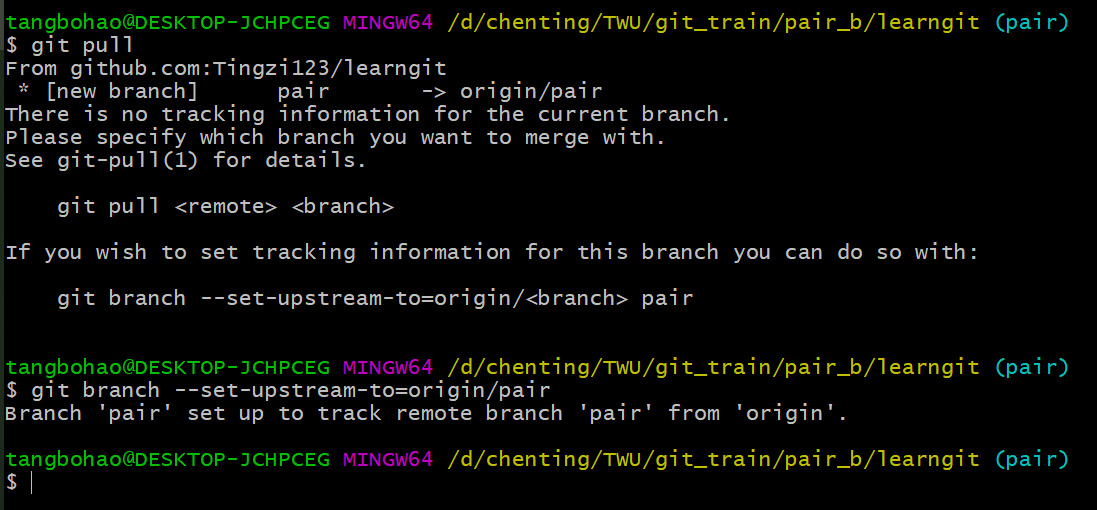
现在A创建pair分支，新建story文件并写好测试，将分支pair推送到远程生成远程的pair分支

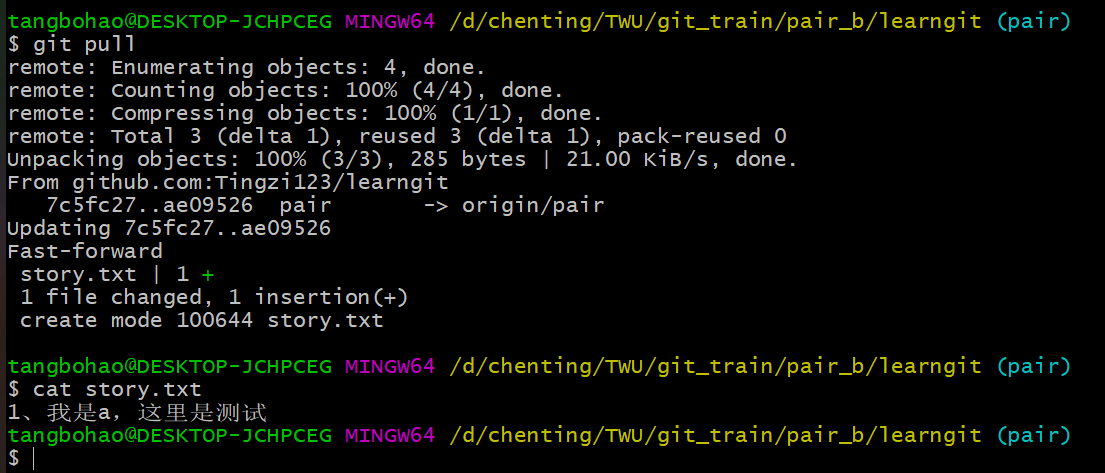
将本地的修改提交到远程pair分支，在提交之前需要与远程分支先建立链接



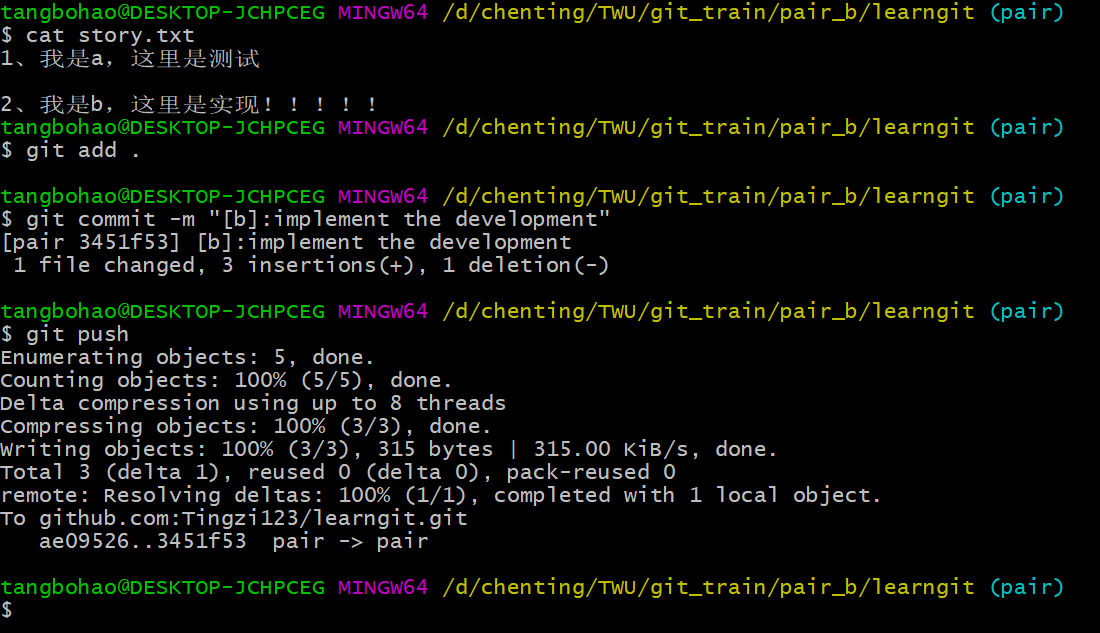
然后B在本地新建pair分支，pull远程的更新到本地，并于远程pair分支建立链接，



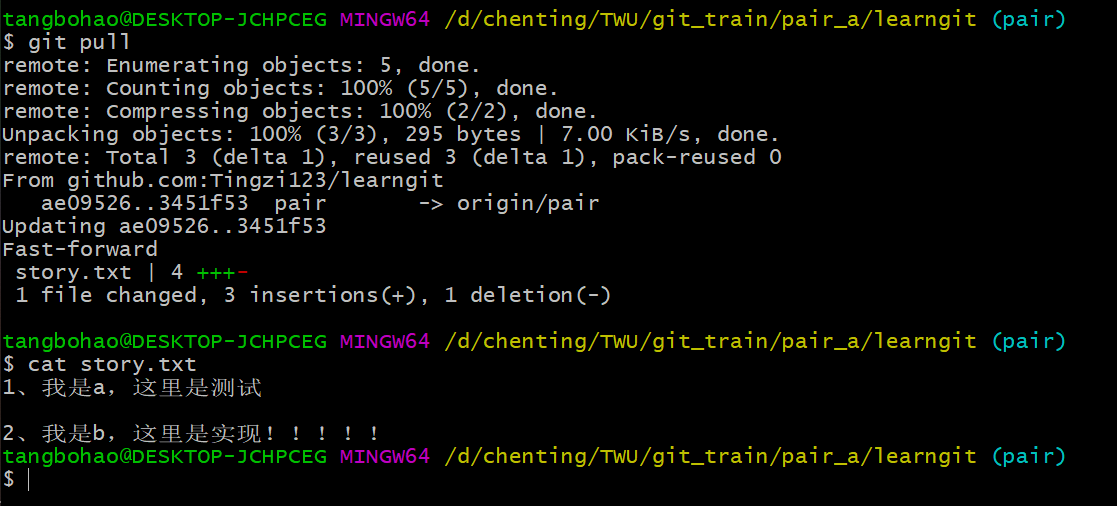




然后B在story文件中写上实现再提交到远程pair分支

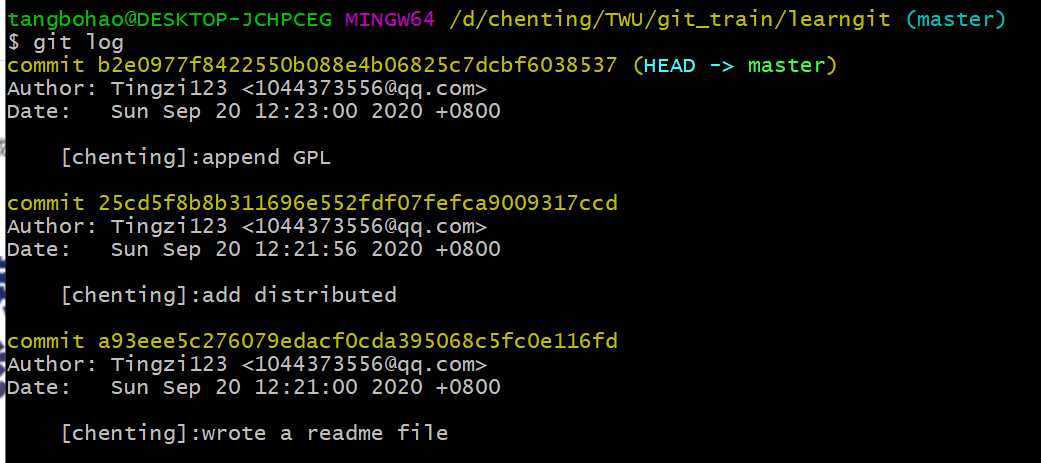


然后同理，a在继续开发之前先pull就可以看到b的实现

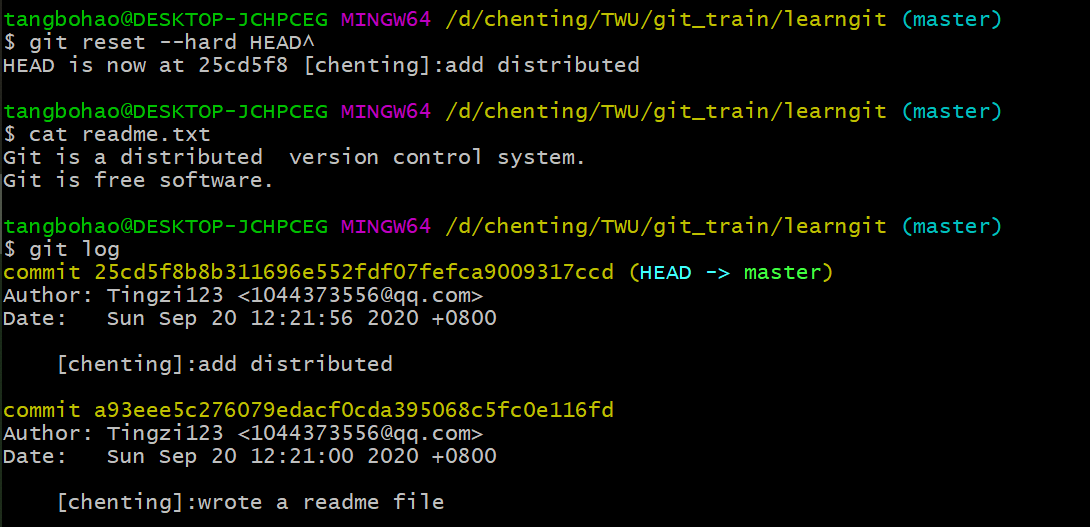


1. 将本地仓库版本回退到两次提交之前的版本后，突然发现回退错了，想将回退后的仓库版本还原到执行回退操作之前的版本。

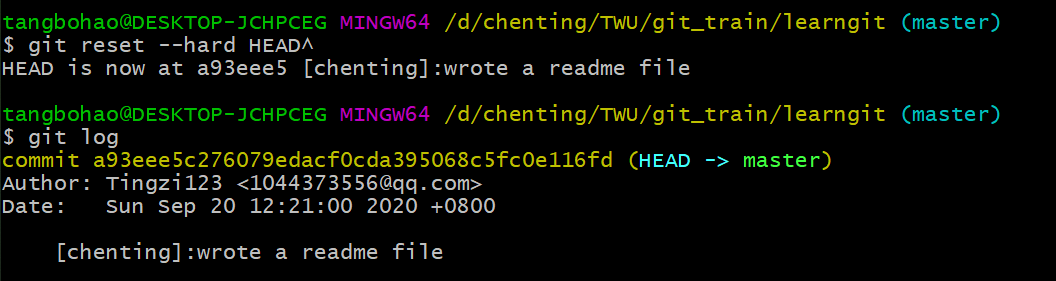
最近三次修改：



回退一次：

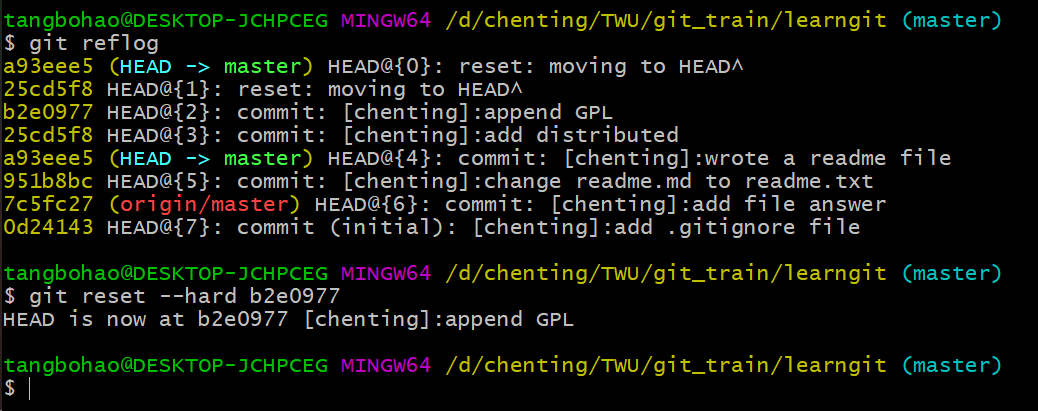


再回退一次：

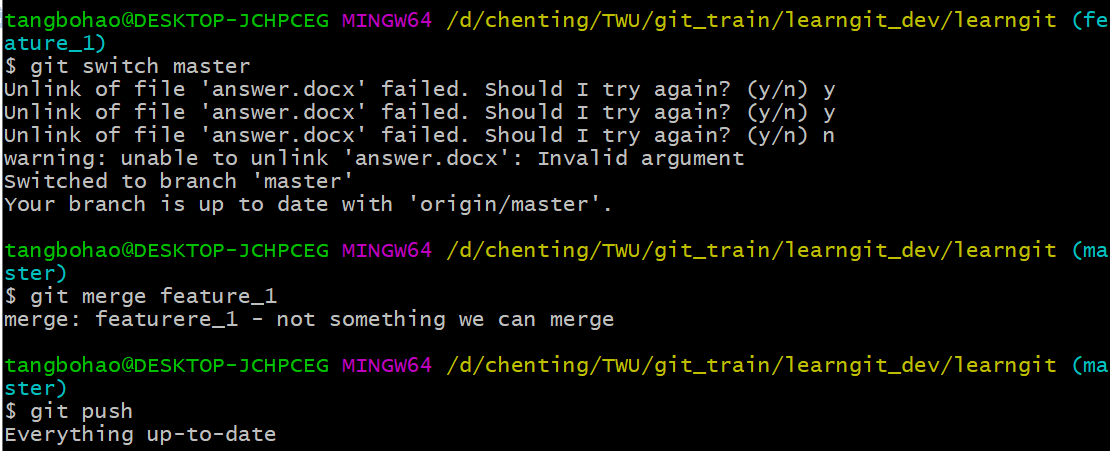


还原到执行回退操作之前的版本：

使用git reflog查看每一次命令记录，找到要回退的版本所对应的commit的id，然后使用git reset --hard id回退

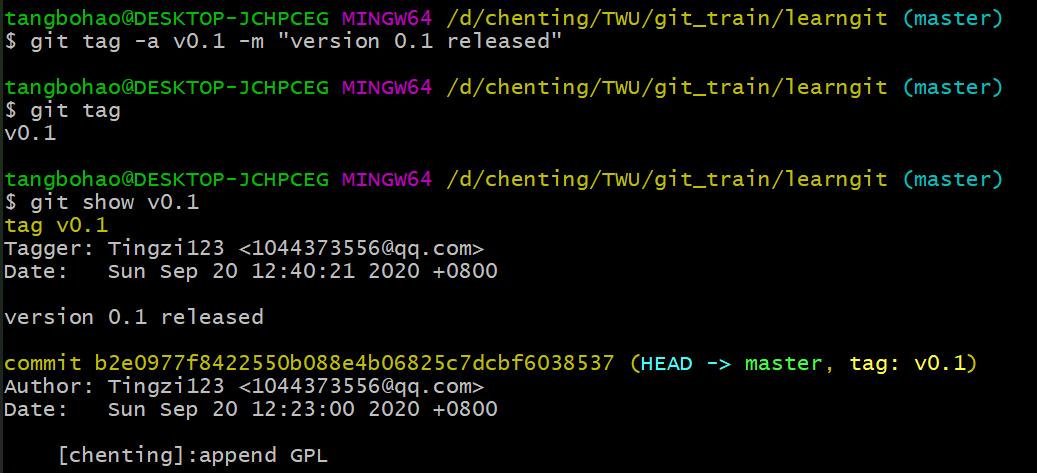


5、将本地 feature-1 分支内容推到 github 上 git@github.com:cheny/learngit.git 仓库中的 master 分支上。

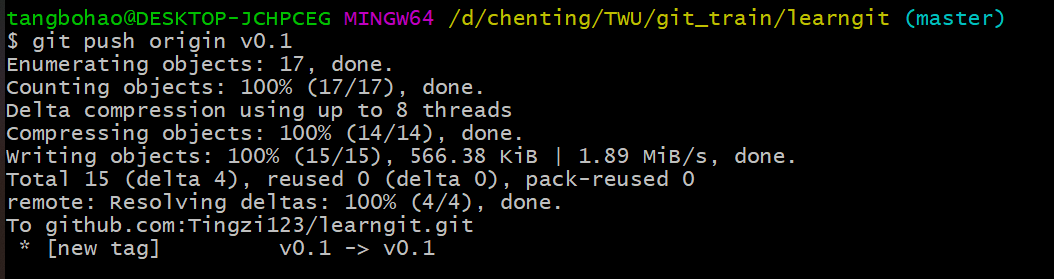


6、版本发布后，需要给代码库打 tag，tag 名为v0.1，描述信息为version 0.1 released。并将该 tag 推到远程分支。以及如何删除本地和远程仓库名为 v0.9的 tag。

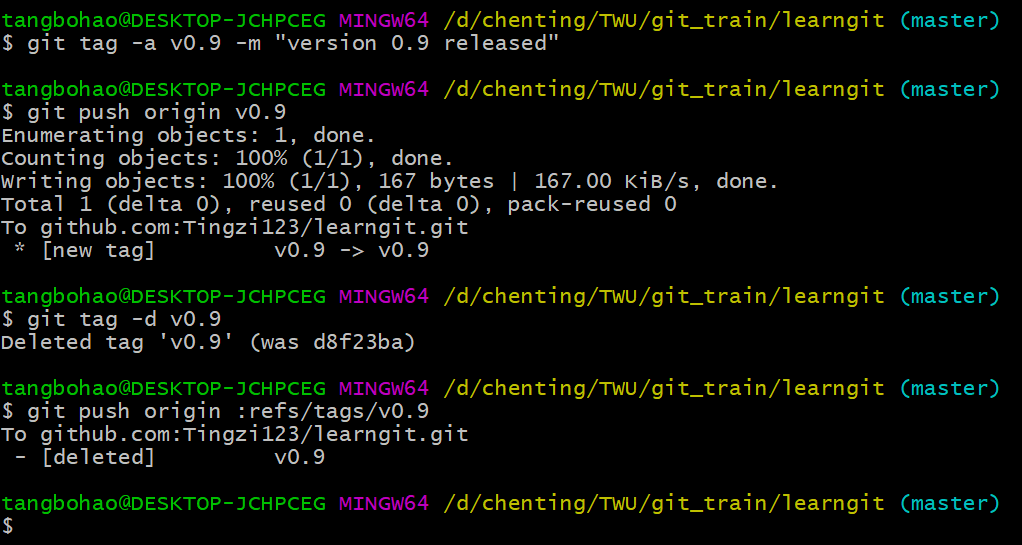
1）给代码库打 tag，tag 名为v0.1，描述信息为version 0.1 released



2）将该 tag 推到远程分支



3）删除本地和远程仓库名为 v0.9的 tag



7、开始了一天的工作，先从 feature-1 分支拉取了最新的代码，然后开始对故事卡的内容进行开发。开发到一半的时候，接到 QA 通知，需要在 dev分支上紧急修复一个 bug。当你在dev分支上修复完成并提交后，又有其他人在 dev分支上提交了新的内容。接着收到通知，需要仅仅将修复该 bug 的提交内容合并到 master 分支上，该修复内容之后的提交不能一起合并到 master 分支上。全部操作完成后，回到 feature-1分支继续进行卡的开发工作。（提示： stash 、 cherry-pick）

1. 在feature-1分支上git stash把当前工作现场“储藏”起来
2. 使用git log查看仅仅修复该 bug 的那次commit id
3. 使用 git cherry-pick commit id进行提交
4. 显示提交人为 cheny的从 2020.04.02到 2020.08.06的所有提交记录，只显示提交的 message 即可，不需要显示详细内容。

git log --author=”cheny” --after ‘2020-04-02’ --before ‘2020-08-06’ --no-merges

1. 将远程代码库中的提交回滚到 id 为 01e31c的提交。
2. git reset --hard 01e31c
3. git push origin master
4. 如何修改倒数第 5 次（HEAD~5）提交的 message 信息。如果要将此次修改同步到远程仓库，如何操作？应该注意什么，如果不注意会造成什么后果
5. git rebase -i HEAD~5
6. git rebase --continue
7. git push -f

应该注意rebase 是会隐藏你真实的修改记录的，最后呈现出来的 git 历史并不能表现你的真实操作

1. 简述 merge 和 rebase的区别。

采用merge和rebase后：

1）git log的区别：merge命令不会保留merge的分支的commit

2）处理冲突的方式：

使用merge命令合并分支，解决完冲突，执行git add .和git commit -m'fix conflict'。这个时候会产生一个commit。

使用rebase命令合并分支，解决完冲突，执行git add .和git rebase --continue，不会产生额外的commit。这样的好处是，‘干净’，分支上不会有无意义的解决分支的commit；坏处，如果合并的分支中存在多个commit，需要重复处理多次冲突。

12简述fetch 分别与 pull和 pull --rebase的区别。

1. fetch和pull都是从远程拉取代码到本地，但是git fetch只是拉取到本地，git pull不仅拉取到本地还merge到本地分支中。所以git pull是git fetch与git merge的集合体。
2. pull操作中merge时会生成一个新的节点，之前的提交分开显示；而rebase操作不会生成新的节点，是将两个分支融合成一个线性的提交。

git pull = git fetch + git merge FETCH\_HEAD

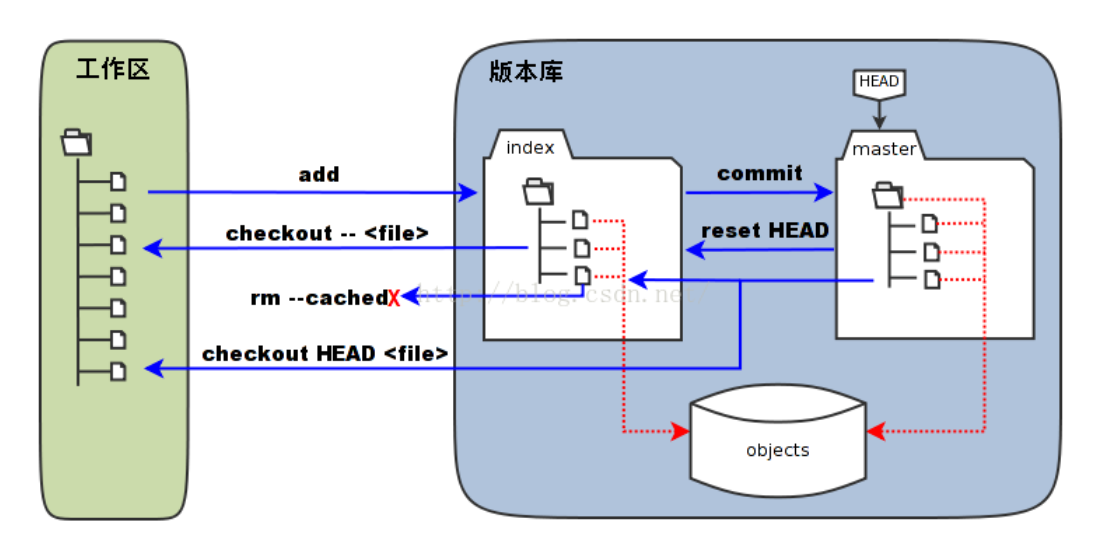
git pull --rebase = git fetch + git rebase FETCH\_HEAD

13、简述 git 中工作区、暂存区和版本库的区别。

1）版本库：在初始化git版本库之后会生成一个隐藏的文件 .git ，可以将该文件理解为git的版本库 repository

2）工作区：我们自己建立的项目文件夹即工作区 working directory

3）暂存区：在.git 文件夹里面还有很多文件，其中有一个index 文件 就是暂存区也可以叫做 stage ,git还为我们自动生成了一个分支master以及指向该分支的指针head ,如下图



1. 简述使用 feature 分支模式进行开发的流程中（拉取代码、创建分支、开发完成后合并分支解决冲突）如何使用一些列的 git 命令。（如 pull, commit, merge, add, fetch,push等）

答案见第3题。