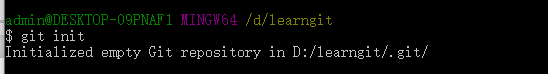
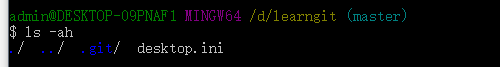
# Git命令学习与记录

1. git init ==》初始化版本库



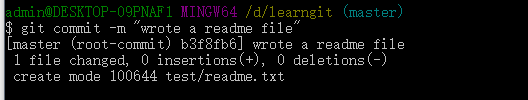
1. ls -ah ==》查看.git文件



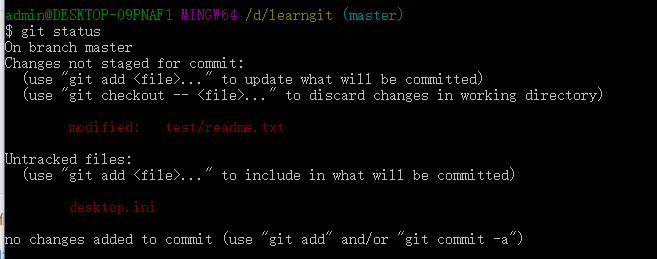
1. git add test/readme.txt ==》添加文件



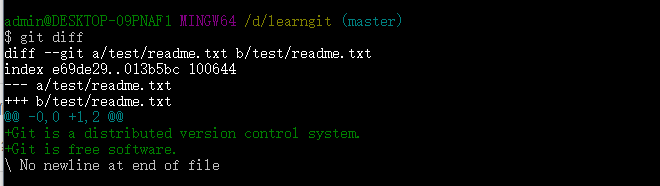
1. git commit -m "修改记录" ==》提交修改，并且标明备注

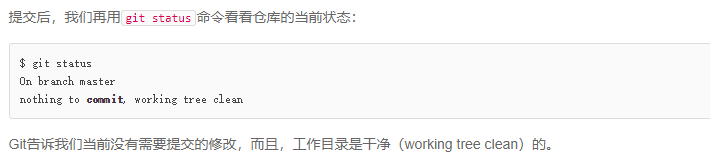


1. git status ==》查看当前的git状态



1. git diff ==》查看上次修改提交的内容

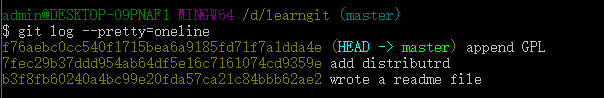




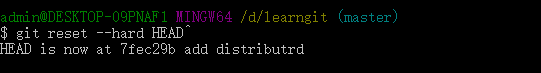
1. git log ==》git提交的日志记录



1. git log --pretty=oneline ==》省去复杂参数

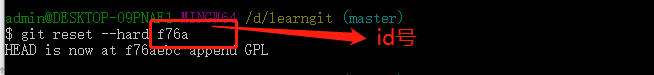


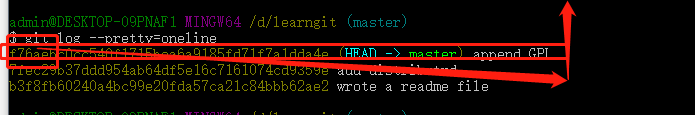
1. git reset --hard HEAD^ ==》回撤到上个版本

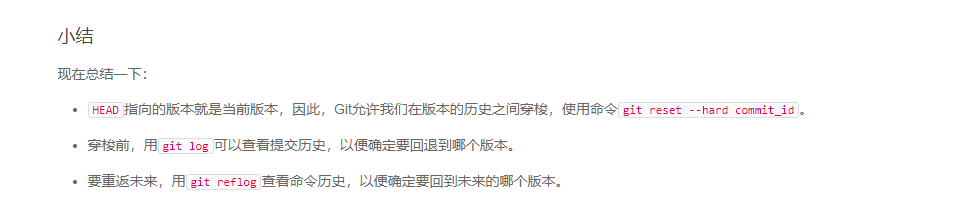




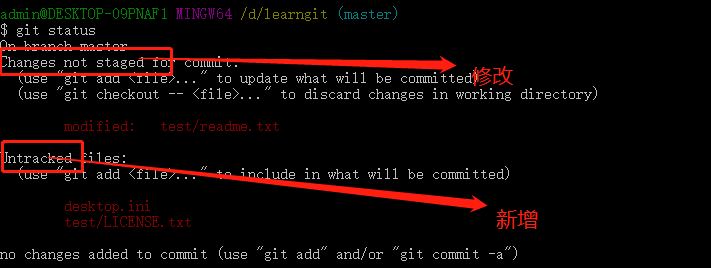
1. git reset --hard f76a ==》找到版本id就可以进入到下个版本



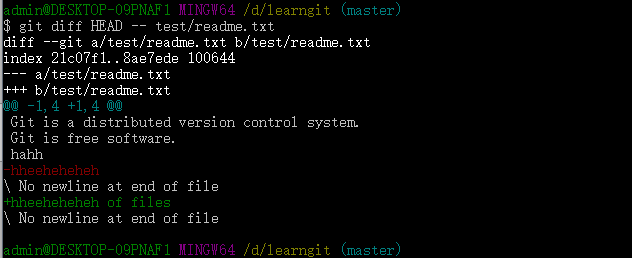




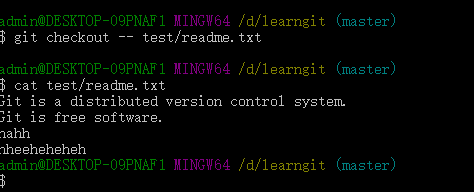
1. 新添加文件，并且修改旧文件的话，用 git status来查看



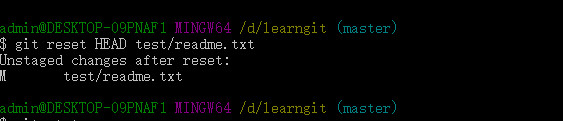
1. git diff HEAD -- readme.txt ==》查看工作区与版本库里面最新版本的区别



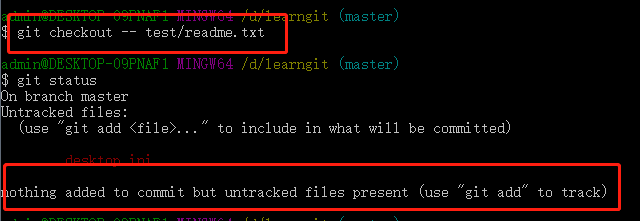
1. git checkout -- test/readme.txt ==》撤销修改，把工作区的文件全部修改，就是让这个文件回到最近一次git commit或git add时的状态。



1. git reset HEAD test/readme.txt ==》把修改区添加的文件放回到工作区



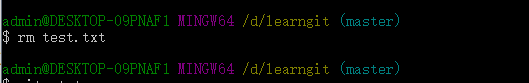
1. $ git checkout -- test/readme.txt ==》丢弃工作区的修改



1. 分情况而定

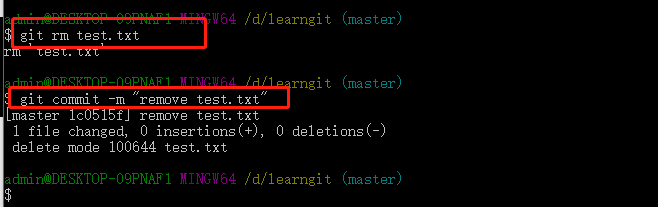


1. $ rm test.txt ==》删除本地文件

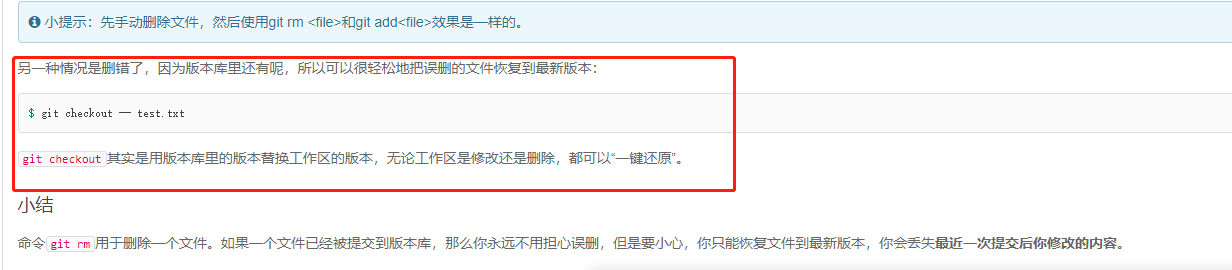


1. git rm test.txt ==》从版本库中移除文件

git commit -m "remove test.txt" ==》并且提交



1. 注意点



1. 远程仓库

第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

ssh-keygen -t rsa -C "youremail@example.com"

你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

之后在github上创建远程仓库，然后在本地项目文件夹下面运行命令

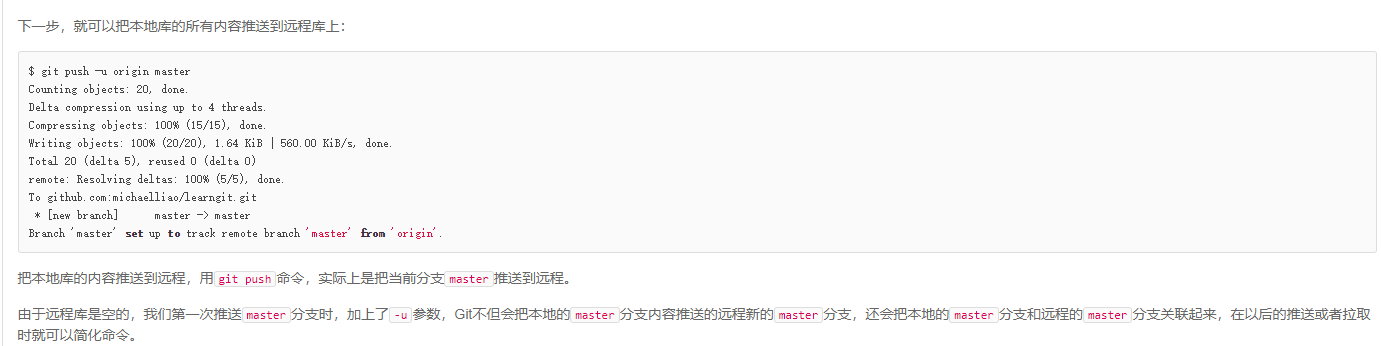
git remote add origin git@github.com:michaelliao/learngit.git

请千万注意，把上面的michaelliao替换成你自己的GitHub账户名，否则，你在本地关联的就是我的远程库，关联没有问题，但是你以后推送是推不上去的，因为你的SSH Key公钥不在我的账户列表中。

添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

下一步，就可以把本地库的所有内容推送到远程库上：

git push -u origin master

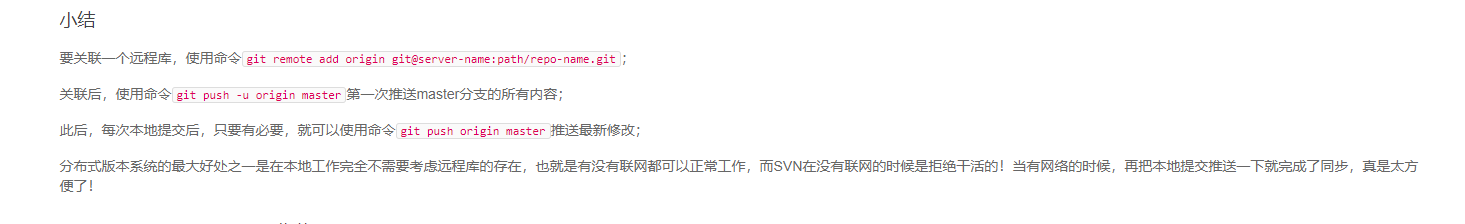


从现在起，只要本地作了提交，就可以通过命令：

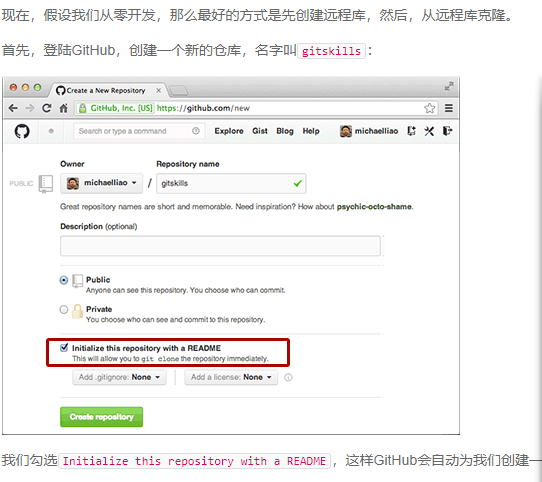
$ git push origin master

把本地master分支的最新修改推送至GitHub，现在，你就拥有了真正的分布式版本库！



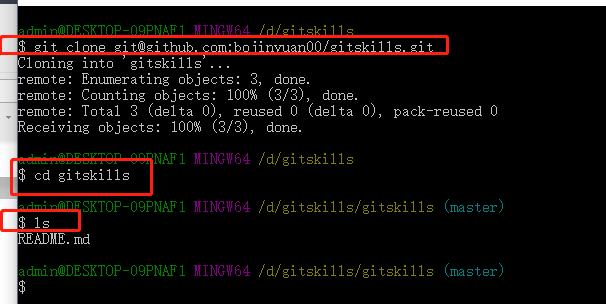


1. 从远程仓库克隆





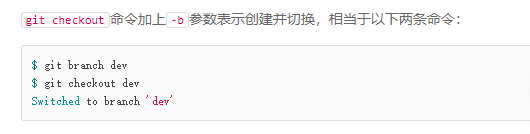




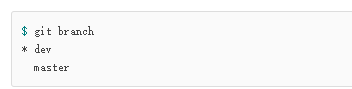
1. 分支-创建分支

git checkout -b dev

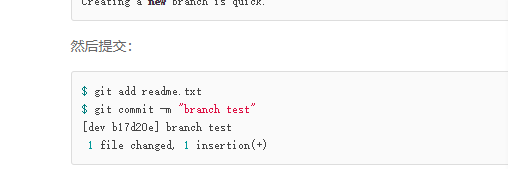




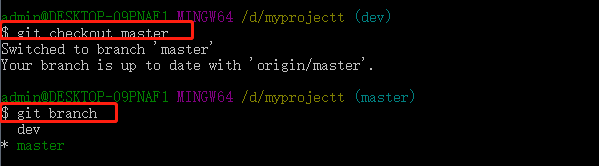
1. 查看当前分支



1. 添加并提交当前分支的工作



1. 
2. 切换回master分支并且查看当前分支



1. 把dev分支上的工作成果合并到master分支上



1. 删除之前的个人分支

git branch -d dev



1. 小结归纳



1. Git解决分支提交的冲突

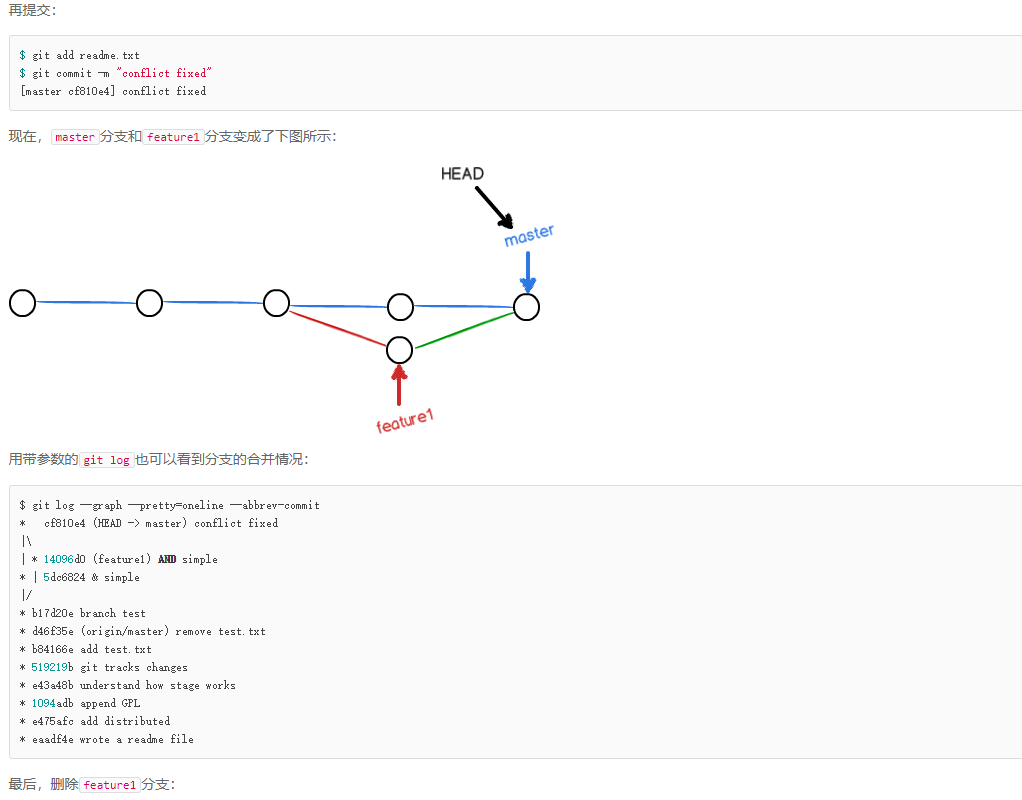


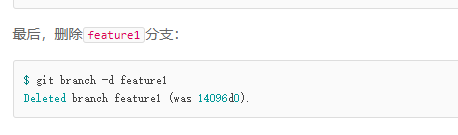
假设此时你对master分支的该文件也做了修改

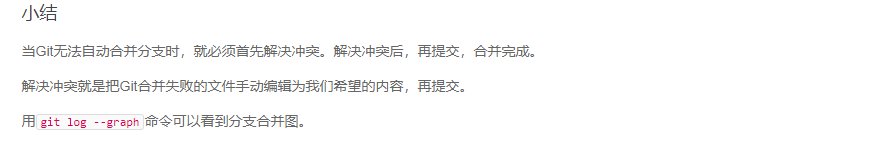
这种情况下是没法对feature1分支进行快速合并到master分支上的



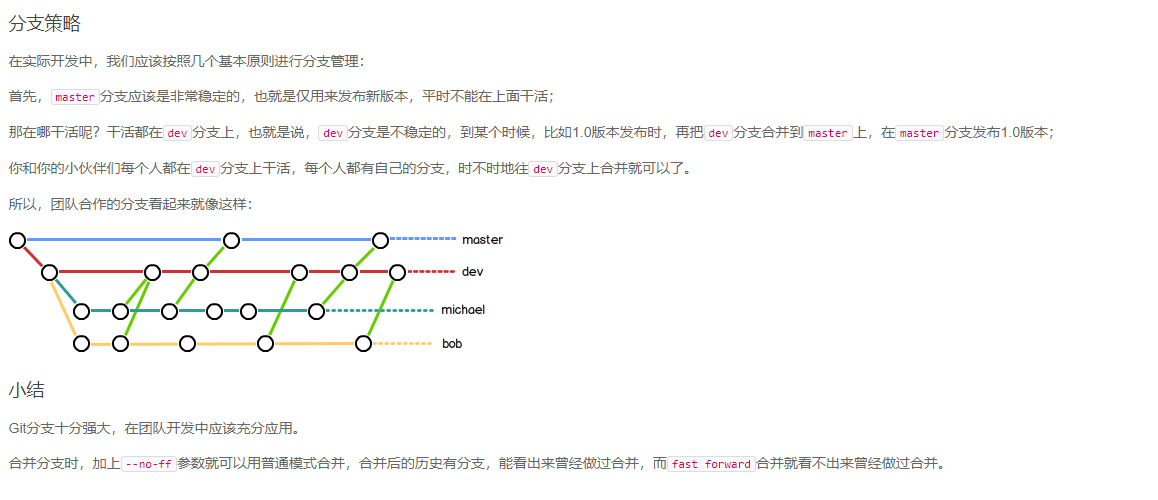
手动修改文件后，再次提交，然后删除feature1分支



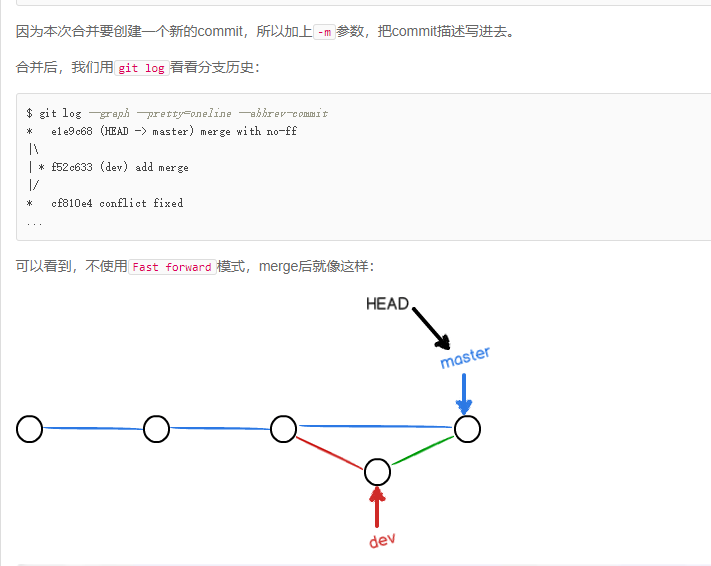




1. 分支管理策略





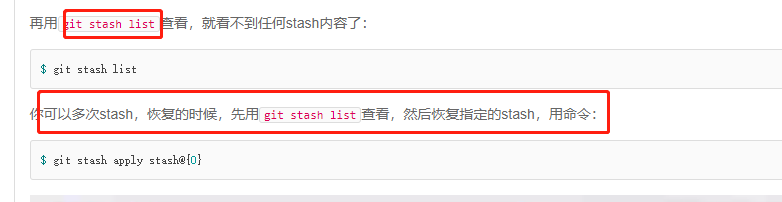


1. Bug分支

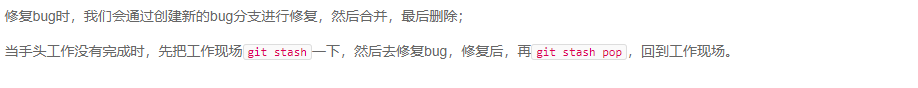








小结：



1. Feature分支

由于新的功能需求，添加一个新的分支用来开发，开发完成后在新分支上已经添加和提交了，准备合并到主分支时，计划取消该功能，并且删除文件数据，这时使用git branch -d feature-vulcan

但是没法删除成功，因为你还没有合并到主分支，所以可以使用另一个命令来强行删除。git branch -D feature-vulcan



