创建版本库repository

git init

添加未跟踪文件

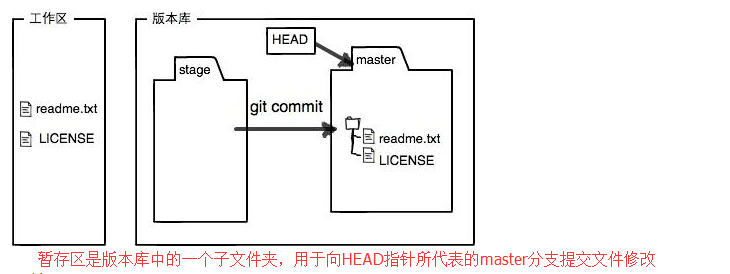
将文件改动暂存到暂存区

git add

将暂存区改动提交到版本库repository

git commit -m “commit information”

暂存区为stage



取消工作区中指定文件改动

git checkout -- fileName

取消暂存区中文件的暂存

git reset HEAD fileName (清零对HEAD指针代表的master分支中指定文件的暂存)

回退repository中的版本

git reset + 版本号

【

git reset --hard 版本号： 将repository、stage、working directory都回退至指定版本

git reset 版本号 : 将repository、state回退到指定版本，workint directory不变

git reset --soft 版本号：将repository回退到指定版本，state、working directory不变

即：

**--hard 三个都回退**

**默认 只回退repository、stage(暂存区)**

**--sort 只回退repository**

回退前2个版本时可以用HEAD^、HEAD^^

】

添加远程库

git remote add 远程库别名 远程库地址

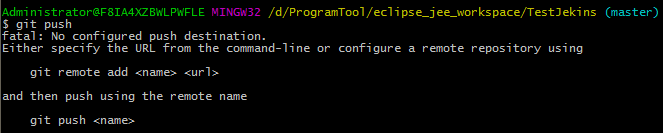
从远程库克隆

git clone 远程库地址

已经进行了如下设置：



但执行git push推送时发生错误：



原因是缺少推送目的地址，解决方法有：

1. 在git push后添加远程库地址，如：

git push git@github.com:ustraskyline/testJenkins.git

1. 设置默认的目的地址

git remote add **origin** git@github.com:ustraskyline/testJenkins.git

也就是说origin是默认的远程分支名，第一步执行git remote时设置的是testJenkins分支，不是默认远程分支.

从repository中删除文件：

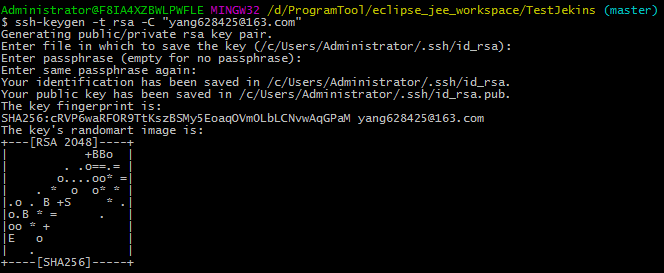
1. 先执行 rm fileName，然后git add，git commit将删除这一改动提交到repository
2. 直接git rm fileName，然后git commit将删除这一改动提交到repository

**更换传输协议：**

Git支持多种协议，默认的git://使用ssh，但也可以使用https等其他协议，通过ssh支持的原生git协议速度最快。

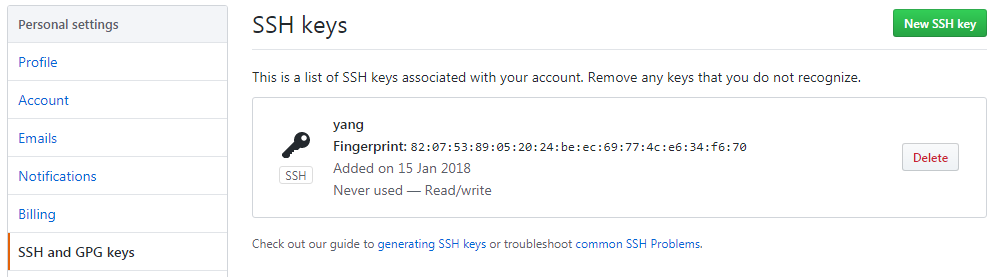
使用https协议在git push时需要输入用户名和密码，也可以在个人文件夹下设置用户名和密码的缓存时间，这里可以采取使用ssh协议来避免输入密码，更改传输协议为ssh步骤如下：

1. 配置用户名和邮箱
2. 执行ssh-keygen -t rsa -C "email-address"生成id\_rsa、id\_rsa.pub



文件名留空可以将id\_rsa/id\_rsa.pub放置在个人文件夹/.ssh/下，可以不输入密码passphrase，这样在git push可以不输入密码

1. 将公钥id\_rsa.pub中的文本添加到github远程仓库

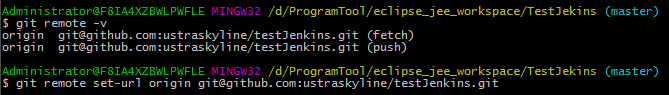


4、验证key是否可以正常工作，执行ssh -T git@github.com



以上即为可以正常工作

1. 修改远程库url使用ssh协议访问，执行git remote set-url origin + 远程库地址



==========End==========

根据协议设置连接远程仓库的地址：

git remote set-url origin repository-url

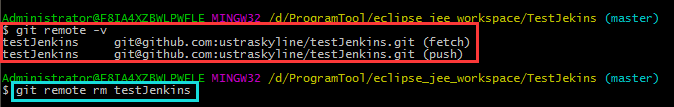
使用https协议时，repository-url以https:开头，如：

IMG_256

使用ssh协议时，repository-url以git开头，如：

IMG_256

删除添加的远程分支名：git remote rm + 分支名



再运行git remote -v就没有该远程分支了:

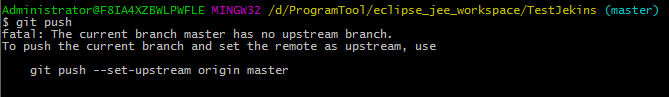


**远程仓库建好之后，本地初始化版本库并推送流程：**

1. git init -> git add -> git commit，将本地内容纳入版本库
2. 设定本地主分支和远程origin分支连接



1. 第一次git push时会有如下错误：



表示当前分支没有上有分支，执行 git push --set-upstream origin master生成远程上游分支

4、执行步骤3后，以后就可以直接使用git push推送

创建分支

git branch 分支名

切换分支

git checkout 分支名

创建并切换分支

git checkout -b 分支名

查看当前分支

git branch

查看当前分支(包括远程分支)

git branch -a

当前分支前会标一个\*号



合并分支：

先回到主分支上，然后执行 git merge 分支名，将指定分支合并到当前分支

删除分支：

git branch -d 分支名

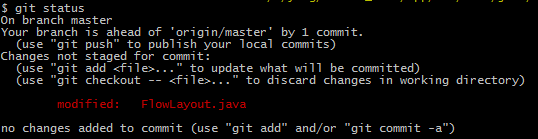


分支冲突：

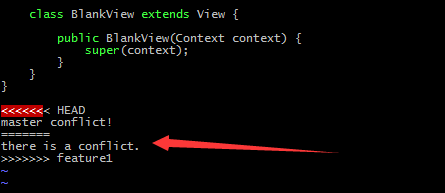
当两个分支都对同一个文件有改动时，合并分支时将发生冲突。此时Git无法执行“快速合并”，只能试图把各自的修改合并起来，如下：



执行git status可以查看具体的冲突情况：

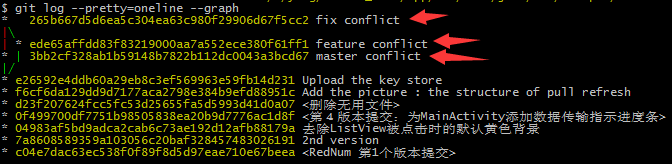


打开冲突的文件，发现git会使用 <<<<<<<、========、>>>>>>>分支名 将有冲突的地方标出来：



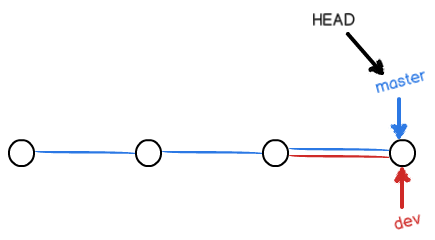
此时可以手动的修改这些冲突，然后执行git add, git commit 提交。也可以不管这些冲突，直接git add, git commit提交，只不过<<<<<<<，=======，>>>>>>>等符号就保存到repository中。

此时应该比master上一个提交超前3个提交，一个是子分支上的改动提交，一个master分支上的改动提交，一个是合并后修改冲突是的改动，如：

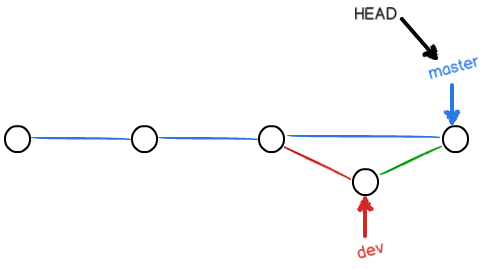


当Git无法自动合并分支时，就必须首先解决冲突，解决冲突后，再提交，合并完成。

快速合并时，直接将master指针指向子分支最新提交，并将HEAD指向master，如下：

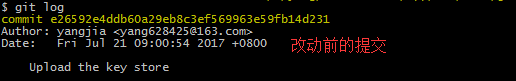


非快速合并关系图：



将当前工作区和暂存区改动储存起来，然后恢复其到上一个提交时的状态：

git stash





现场：



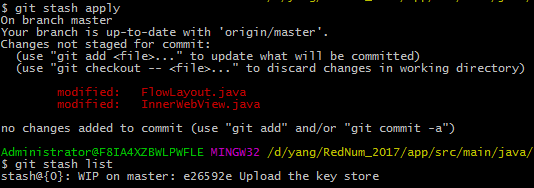
查看储存的工作现场：

git stash list



恢复现场：

git stash apply (不会删除储存的现场)

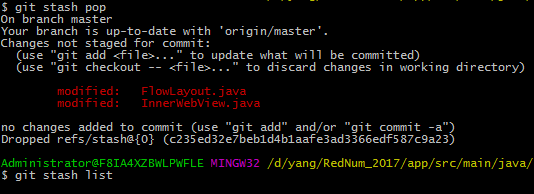


删除储存现场：

git stash drop

恢复现场并删除储存现场：

git stash pop



恢复指定的现场：

git stash apply stash\_id

强行删除一个未被合并的分支

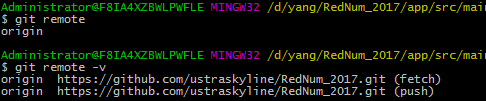
git merge -D 分支名

从远程仓库克隆时，Git自动把本地的master分支和远程的master分支对应起来，并且，远程仓库的默认名称是origin.

查看远程库

git remote

git remote -v



若没有推送权限，就看不见push地址

将该分支上的所有本地提交推送到远程库

git push 远程分支名 本地分支名

抓取分支

git pull

从远程库clone时，默认情况下，只能看到本地的master分支，无法看到其他分支.

在本地创建一个分支用于跟踪指定远程分支

git branch 本地分支名 远程分支名

git checkout -b 本地分支名 远程分支名

指定本地分支与远程分支的链接：

git brach --set-upstream 远程分支名 本地分支名

标签是跟某一个commit id绑定在一起的、有意义的容易记住的名字，标签只存在在本地，不会自动推送到远程。

为当前分支的最新提交打标签

git tag <tag\_name>

为指定提交打标签

git tag <tag\_name> <commit\_id>

查看标签(标签不是按时间顺序列出，而是按字母顺序排序)

git tag

查看标签信息

git show <tag\_name>

创建带有说明的标签  
 git tag -a <tag\_name> -m <说明信息>

删除标签

git tag -d <tag\_name>

将标签推送到远程服务器

git push origin <tag\_name>

一次性将所有尚未推送到远程的本地标签推送到远程服务器

git push origin --tags

删除远程服务器上的标签

git tag -d <tag\_name> //先从本地删除

git push origin :refs/tags/<tag\_name>