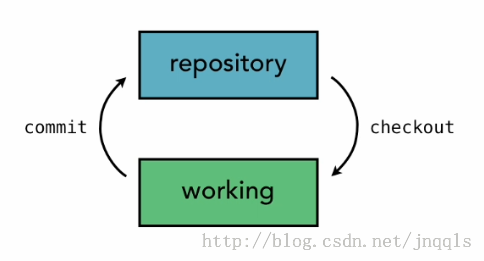
# 三区介绍

在上篇文章的结尾我们提到了Git 的三区结构,在版本控制体系中有这样两种体系结构,一种是两区结构一种是三区结构.接下来我们通过对Git三区的结构学习来帮助我们更好的去理解并运用Git.

两区结构是其他版本控制软件常用的结构:工作区--版本库,如下图

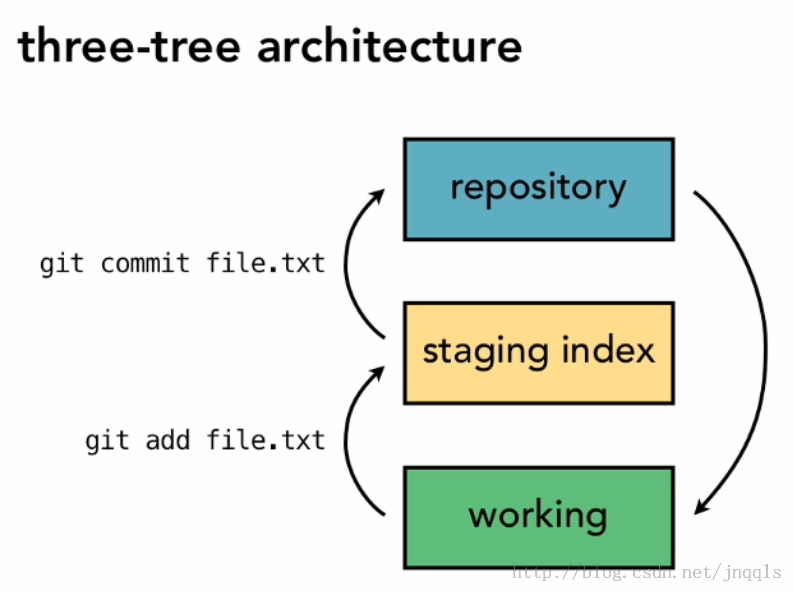


工作区直接跟版本库打交道

从版本库中检出内容到工作区.

从工作区提交内容到版本库.

三区结构是Git的结构:工作区--暂存区--版本库,跟二区的不同是在中间加了一个暂缓区.



此时工作区直接跟暂缓区打交道暂缓区跟版本库打交道.

从工作区中添加内容到暂缓区.

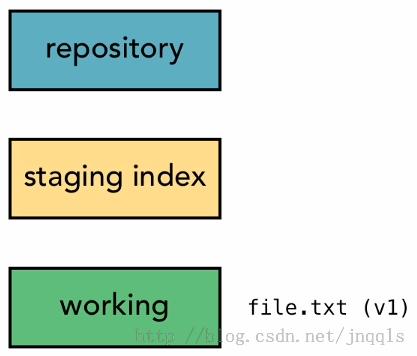
将暂缓区的内容提交到版本库.

从版本库中检出内容到工作区.

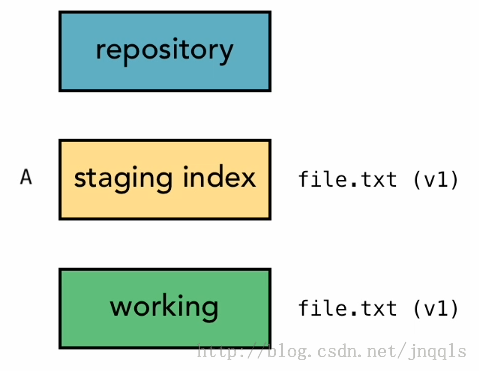
接下来将通过图文的方式为大家介绍Git的一个基本流程.主要是三个变化A,B在Git中是如何进行的.

第一个版本变化A,新建一个文件,然后添加到版本库中.

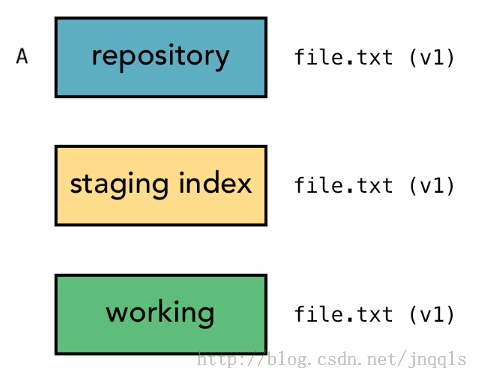
增加一个文件:file.txt



将文件添加到缓存区:git add file.txt

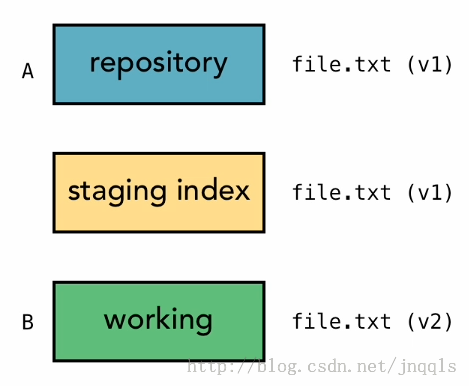


将文件从缓存区提交到版本库中:git commit –m “”

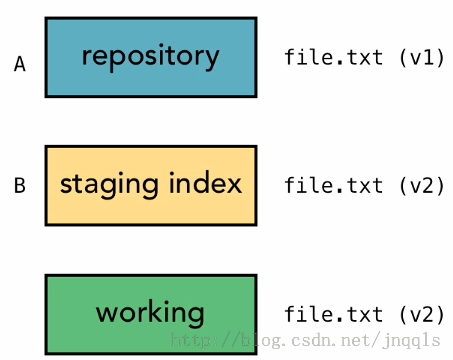


以上的过程已经建立了一个版本A,接下来我们再看一下在A的基础之上进行版本变化B.

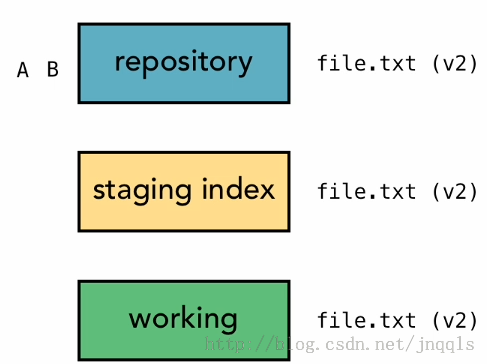
更改文件file.txt



将文件添加到缓存区:git add file.txt



将文件从缓存区提交到版本库中: git commit –m “”

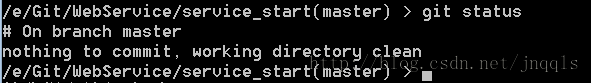


通过以上两个比较简单的流程图文示例相信大家已经非常清楚文档的变化是如何在Git体系中进行的.在接下来的一篇文章将会为大家通过实际的一个例子,同大家一同进行探讨.

给读者留下一个问题:你觉得在版本控制系统中增加暂存区有什么好处呢?

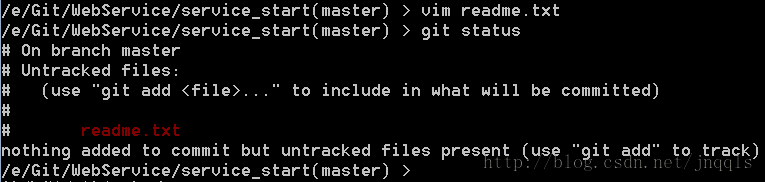
在 之前的文章中我们已经初始化了一个仓库,在接下来的内容我将为大家分享在Git中最为常用的一些操作和命令.如果你已经理解了Git三区的概念并且能够进行Git的基本操作,那么就可以说你已经入门了并且能够在企业的实际操作中进行运用.

         如果你已经完成了初始化仓库的工作,那么你已经可以通过Git来对仓库里的项目进行管理,一般而言,我们都需要确定一下那些文件处于什么状态(三区状态),这时候我们可以用gitstatus的命令.因为我们刚刚初始化仓库,所以读者会看到以下的内容:

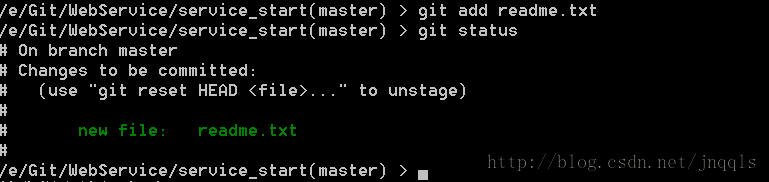


         通过Git的提示可以看出,没有什么需要提交的,工作目录很干净.也就是说当前的目录没有任何跟踪着的文件.

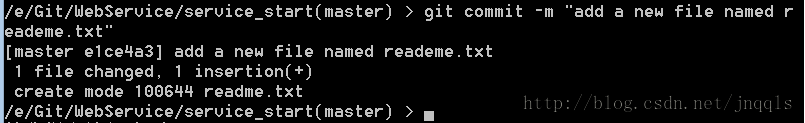
         接下来我们要模拟一下实际工作中的情况.我们要通过增加一个文件,此处用一个Readme.txt文件来模拟实际项目中增加的文件.请读者看下面的操作.



         我们通过Vim编辑器增加了一个readme.txt文件,当再运行git status的时候会看到该文件出现在未跟踪的列表中.这时候readme.txt文件不会纳入跟踪范围颜色显示为红色.而接下来我们要求将此文件纳入跟踪管理,此时需要使用命令git add readme.txt 来跟踪此文件.



         此时在运行git status 会发现readme.txt文件已经被跟踪 并且处于暂存的状态,而且颜色由原来的红色变为现在的绿色.剩下的就是需要将此变化进行提交更新,作为一个版本纳入仓库管理中.接下来我们需要使用命令git commit 来进行提交,最佳实践是在提交的同时写上相关的提交信息可以加上 -m参数来增加相关的信息.

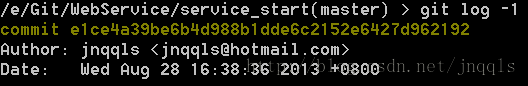


          恭喜你,现在你已经成功的完成了第一个提交.读者可以从提交的信息看到你是在那个分支提交的(master),本次提交有多少个文件修改过,多少行添加和修改等等相关的信息.

         每一次运行提交操作，都是对你项目作一次快照，以后可以回

到这个状态，或者进行比较。如果在提交了很多次之后想回顾一下提交的历史,此时可以用git log 命令来回顾.

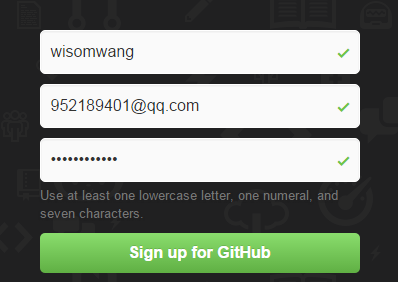
         在项目中运行 git log -1 会看到以下内容. -1 的参数表示只显示最近的一次更新.



           至此,读者基本上可以运用Git在你的项目中,当然,这一系列的博客是帮助读者以一个比较轻松的角度去了解并逐渐掌握Git这门技术,真正的运用还需要自己在实际的项目中不断运用总结

# 帐号相关

* Git 是一个分布式的版本控制系统，Git hub是一个源码库，可以托管各种git库
* bitbucket.org帐号注册(没有成功)
* github帐号



<https://help.github.com/articles/set-up-git/>

网址: <https://github.com/>

如果github已经创建了repository，及对应的工程，直接clone

git-bash

git config --global user.name "wisomwang"(密码404\*\*\*wsm\*\*\*)

git config --global user.email [952189401@qq.com](mailto:952189401@qq.com)

git clone <https://github.com/wisomwang/helloworld> 从github下载helloworld

如何创建分支到github上

前提条件

1. 创建ssh key，然后把id\_rsa.pub中的key复制到github中的ssh key中
2. 验证是否成功 ssh –T [git@github.com](mailto:git@github.com)
3. git config --global user.name "wisomwang"
4. git config --global user.email [952189401@qq.com](mailto:952189401@qq.com)

* 先在github上创建一个repository,如study
  + Git clone下来
  + 添加文件
  + Git add
  + Git commit
  + Git push origin master

这样就把本地的master分支与远程的master分支关联起来了

* 通过cmd创建远程repository及master分支
  + 创建一个目录，如study
  + Git init
  + Git add
  + Git commit
  + Git remote add origin git@github.com:wisomwang/study
  + Git push origin master

这样就把本地的master分支与远程的master分支关联起来了

* 添加远程非master分支
  + Git checkout –b branchname
  + Git push origin branchname(推送到远程branchname分支)
* git add

git add 某个具体文件

git add . 当前目录下所有文件

* git commit

git commit –m “comment” (递交所有文件)

git commit –a –m “”

-a 表示只会递交修改过的文件，不会递交新增的文件

* git amend(更正最后一次提交的信息)

git commit –amend –m “comment”

* git rm(删除文件，包括本地和远程)

git rm 文件名

然后commit, push

删除远程文件，但本地文件保留

Git rm –r –n--cached 文件名(执行命令时不会删除任何文件，只是显示要删除的文件预览列表)

Git rm –r –cached 文件名(最终删除文件)

Git commit –m “”

Git push origin master

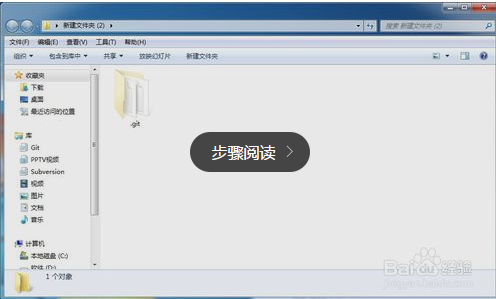
* git clone





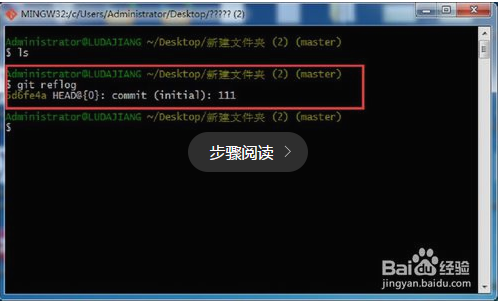
1. Git reset

打开git库文件，可以看到文件夹内的所有文件都没了，只有一个git仓库还存在



进入Bash页面后，我们执行ls操作，可以发现，文件夹下同样不存在文件

这时我们再执行git reflog。reflog它会记录所有HEAD的历史，也就是说当你做 reset，checkout等操作的时候，这些操作会被记录在reflog中



如果我们要找回文件，只需要做如下操作（\*表示上一步中出现的log号）：

git reset --hard \*(\*表示版本号，从reflog中可以看出)



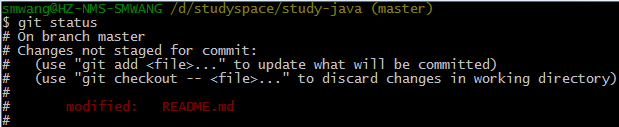
我们再次执行ls操作，可以发现文件已经还原回来了，如下图所示



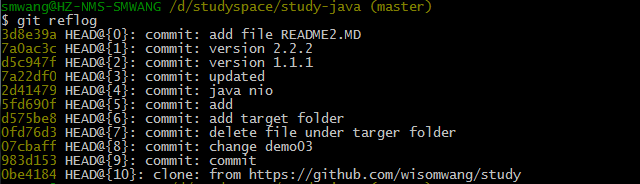
git reset --hard \*(\*表示版本号，从reflog中可以看出)

该操作只能针对整个branch，不能针对单个文件，该操作后，运行git status不会有任何需要提交的文件，包括你本地修改的也会覆盖掉

Git reset –soft \*



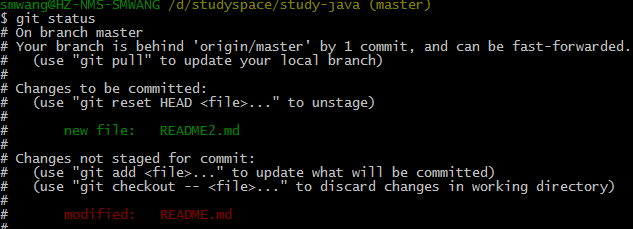
本地有一个文件修改了但没有commit



查看commit历史，最新的一次commit了一个文件README2.md

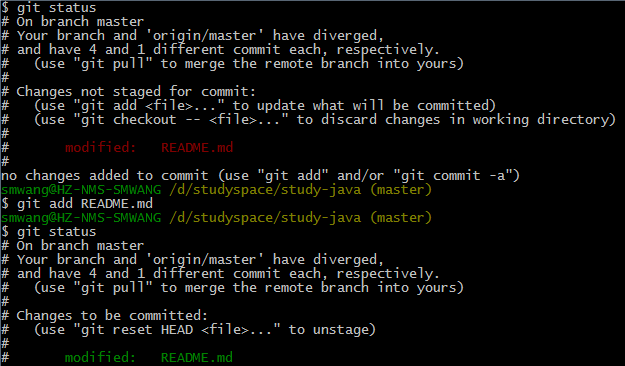
我现在想reset到版本7a0ac3c,但又想保留本地的文件内容，包括README.md修改过的内容和文件README2.md，如何操作，执行下面命令

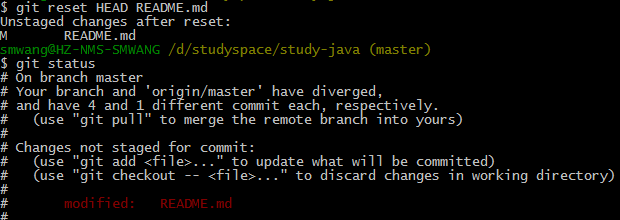
Git reset –soft 7a0ac3c执行完后，用git status查看,达到我们想要的结果



如果发现错误的将不想提交的文件add进入index之后，想回退取消，即remove from index,可用下面命令

git reset HEAD file





# Git checkout

git checkout -b new branch name

建立并切换到新的分支

# Git diff

Git diff README.md

比较工作目录与暂存区域

Git diff HEAD

比较工作目录和本地HEAD(本地的最新版本库)

Git diff –cached README.md

Git diff –staged README.md

比较暂存区域与本地版本库

Git diff branch master

比较本地的branch版本库与本地的master版本库

Git diff branch master README.md or

Git diff branch master -- README.md or

Git diff branch master README.md -- ./README.md or

Git diff branch master README.md ./README.md

比较本地的branch版本库与本地的master版本库的README.md文件

Git diff branch origin/master

比较本地的branch版本库与远程的master版本库，但实际上如果我在远程的master做了修改后，是比较不出差异的，一定要先git fetch master,然后

Git diff branch master

Git diff HEAD HEAD~2

Git diff历史提交的编号(省略了当前工作目录分支)

比较工作目录和某个历史提交

# Git stash

我当前在master,针对某些类做了一些代码的改动，但临时接到一个活，要去修改一个bug，但我的改动还不能commit上去，这个时候可以用git stash，他相当于是临时保存我的修改，然后我就可以去修改bug，等修改完bug以后，我可以从临时提交中取出我刚才的提交

git stash apply

在本地对README.md作了修改

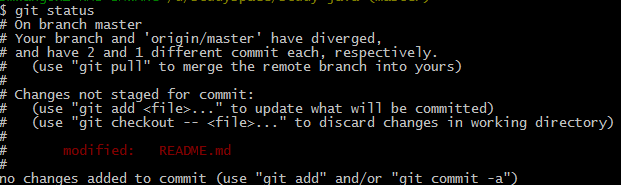
# study

for my self-study

version 1.1.1

version 2.2.2

version 3.3.3



执行git stash后，README.md恢复到修改前的内容,git status无内容变化



# study

for my self-study

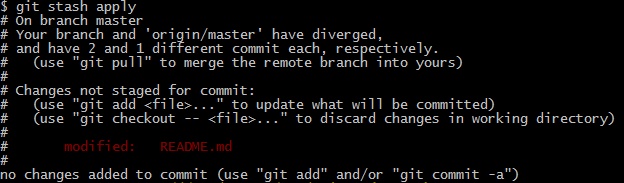
version 1.1.1

version 2.2.2

version 3.3.3

version 4.4.4

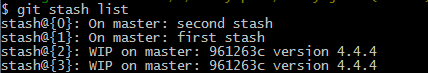
然后执行git stash apply,又恢复到原来修改的内容了



可以进行多次git stash

git stash save “first stash”

git stash save “second stash”



通过如下方式取回对应的stash

git stash apply stash@{1}

# 删除

查看本地分支

Git branch

查看包括远程分支

Git branch -a

删除远程分支

Git push origin :branchname

删除本地分支

Git branch –d branchname

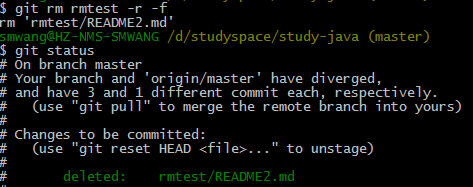
上面的命令有时候不一定成功

Git branch –D branchname

强制删除

删除文件夹

Git rm 文件夹 -r -f



还需要git commit和git push最终从HEAD中remove

Git commit –a –m 把修改过的文件commit到分支，不需要git add,但新增加的文件不会起作用

# Git rebase

两个分支 master branch

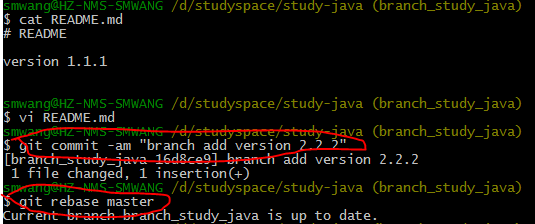
有两个文件 README README

README2 README2

一开始branch和master上的文件内容是一致的

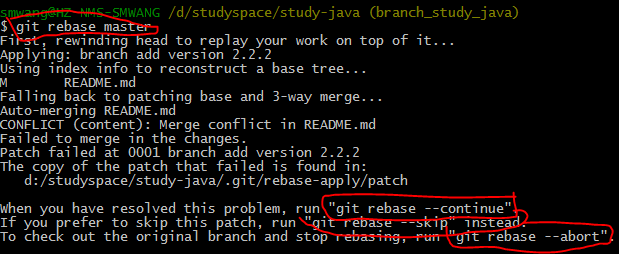
Step1: 对branch的README的做修改

Step2: 这时候在branch分支上做git rebase master是没有意义的，已经是最新了



Step3:在master上的README2做了修改，然后切换到branch分支上做git rebase master会把master上的README2所做的修改合并过来

Step4:在master上的README和README2都做修改，然后切换到branch分支上做git rebase master操作，这时会把master上的README、README2所做的修改合并过来，但README在合的过程中有可能会出现冲突，如果有冲突所要先解决冲突，对解决冲突的文件要先git add, 然后git rebase --continue,这时相当于重新rebase了



##################################################

##################################################

实际案例

解决方案一

本地master分支上有commit,想push到远程master分支上，结果被拒绝，原因是在远程master分支已经有人commit了，如何解决，下面是一种方法

先把自己改动的那个文件备份起来

通过git reset –hard HEAD^ 回退到上一次提交，相当于是取消最近一次提交

然后git pull

然后再把备份的文件拷贝回来

然后commit,再git push

解决方案二

直接git pull过来，如果修改的不是同一个地方，不会有冲突，如果有冲突，先解决冲突，然后commit,最后git pull

解决方案三

先执行git fetch origin master or git fetch origin(把远程master上的内容更新到本地的master分支上)

Git rebase master(本地的master)

如果有冲突，解决冲突，然后

Git add README.md

Git rebase --continue

##################################################

# 常用命令

Master:本地master分支

Origin/master:远程master分支

Remotes/origin/master：远程master分支

Git push origin master是简写形式，全写应该是

Git push origin master:master

Origin表明远程

第一个master表示本地分支名

第二个master表示远程分支名

Git push origin master:refs/for/mybranch(把本地的master合到远程的mybranch分支上)

Git push origin HEAD:refs/for/mybranch(HEAD指向当前工作的branch,master不一定表示master分支，不一定代表当前分支)

Git push origin :mybranch （再origin repository里面查找mybranch，删除它。用一个空的去更新它，就相当于删除了）

创建分支： $ git branch mybranch

切换分支： $ git checkout mybranch

创建并切换分支： $ git checkout -b mybranch

更新master主线上的东西到该分支上：$git rebase master

切换到master分支：$git checkout master

更新mybranch分支上的东西到master上：$git rebase mybranch

提交：git commit –a

对最近一次commit的进行修改：git commit -a –amend

commit之后，如果想撤销最近一次提交(即退回到上一次版本)并本地保留代码：git reset HEAD^

合并分支：$ git merge mybranch (merge from mybranch)  
删除分支： $ git branch -d mybranch  
强制删除分支： $ git branch -D mybranch  
列出所有分支： $ git branch  
查看各个分支最后一次提交： $ git branch -v

查看哪些分支合并入当前分支： $ git branch –merged

查看哪些分支未合并入当前分支： $ git branch –no-merged

更新远程库到本地： $ git fetch origin  
推送分支： $ git push origin mybranch  
取远程分支合并到本地： $ git merge origin/mybranch  
取远程分支并分化一个新分支： $ git checkout -b mybranch origin/mybranch  
删除远程分支：$ git push origin :mybranch