

git init

git add

git commit

git remote add xxx.git (**本地新项目**，远程没有，这时要remote add)

git push -u origin master

### git branch

git branch xxx 在当前分支新建一个分支（这时新分支的初始内容同当前分支，类似于复制，但是可以用新分支去做别的事情，而不影响当前分支）

git branch [**-a** 本地远程所有分支] [**-r** 远程所有分支] [**-d** 删除本地分支] [**-d -r** 删除远程分支，删除后还需推送远程服务器] [**-m** 重命名本地分支]

git checkout [**-b** 创建并切换] xxx

git push **--set-upstream** origin xxx 将当前**本地新分支**和远程指定分支关联

**注：加上--set-upstream之后，以后再提交这分支，只需要git push即可，因为该分支已经和远程指定分支关联了，git 会自动push到关联的远程分支上**

### git add

git add . 提交新文件、编辑文件到暂存区

git add -u 提交编辑文件、删除文件到暂存区 -u update

git add -A 提交新文件、编辑文件、删除文件到暂存区 -A all

### git fetch

git fetch <远程主机名> ：将远程主机的更新全部取回本地

git fetch <远程主机名> <分支名> ：取回远程主机的指定分支到本地

git fetch origin master ：取回远程master分支到本地

取回分支后，会返回一个 FETCH\_HEAD，指某个分支在服务器上的最新状态，可以通过FETCH\_HEAD在本地查看刚取回的更新信息

git log -p FETCH\_HEAD 查看FETCH\_HEAD的信息

### git pull

git pull 的过程可以理解为

1. git fetch origin master
2. git merger FETCH\_HEAD

git pull = git fetch + git merge ，有冲突需要手动更新

### git diff

git diff [--stat] branch1 branch2 比较两个分支之间的差异（两个分支可以分别是本地和远程，共四种情况）

### git revert（逻辑删除）

git revert的作用通过反做创建一个新的版本，这个版本的内容与我们要回退到的目标版本一样，但是HEAD指针是指向这个新生成的版本，而不是目标版本

### git reset（物理删除；撤销本地commit，add操作）

<https://www.jianshu.com/p/c2ec5f06cf1a>

回退后，暂存区和工作空间的内容细节**需要注意**，并不止有原版本内容，可能会有目标版本的暂存区和工作空间内容

git reset --soft HEAD^ ：只撤销当前commit操作，保存暂存区和工作空间

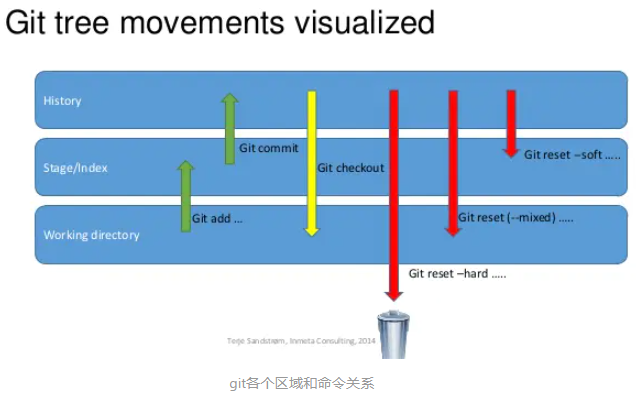
git reset --mixed HEAD^ ：撤销当前commit 和 add操作，保存工作空间代码。**默认**

git reset --hard HEAD^ ：删除暂存区和工作空间代码，HEAD指向目标版本号

**HEAD^(HEAD~1)表示上一个版本**，也可以通过git log获取指定版本号进行回退

**注：**

**使用git reset --hard后，本地库中回退那部分版本内容会被全部删除，而这时远程库中还是原来的版本，这是使用git push 推送时，会提示报错，但是如果使用git push -f 强制推上去，会导致严重后果，即远程库的一部分提交也被全部删除，所以实际开发中git push -f被禁止使用**

****

### git restore（撤销本地add修改）

git restore <file>：将在工作空间但是不在暂存区的文件撤销（只修改未add的文件内容被删除）

git restore --staged <file>：将在暂存区的文件从暂存区撤回，但不会改变文件（撤销add操作）

### git stash

<https://www.cnblogs.com/zndxall/archive/2018/09/04/9586088.html>

### git log

git log **-2** 显示最近两次提交

git log **-p** 显示每次提交的具体差异

git log **--stat** 显示每次提交的粗略统计

git log **--oneline** 每次提交只显示一行（每次提交的SHA-1校验和、commit注释）

git log **--pretty=format:”%cd %s”** 以指定格式显示，%cd提交时间 %s 提交注释

git log **--graph** 以图形显示提交记录

git log **--befor --afte**r

**git log --graph --oneline** 或者 **git log --graph --pretty=format:”%cd %s”**