# git命令详解

<https://git-scm.com/docs> Documentation reference

<https://github.com/git-for-windows/git>

## 1、用git的命令行修改提交的用户名和邮箱：

git config --global user.name "Your Name"  
git config --global user.email youEmail@example.com

如果只是查看的话就不要加后面的用户名和邮箱了。

## 2、关于git ssh key密钥：

SSH key 的作用就是远程存放仓库的服务器要注册有你本地生成的ssh key ，这样你才能把你本地的仓库和远程的仓库进行关联，同步修改。不然你是没有权限把本地修改同步到远程的。

1、查看是否已经有了ssh密钥：cd ~/.ssh

存放密钥的文件夹：.ssh一般是自动生成在c盘的用户目录下的。如果上面的命令能定位到这个文件夹则表示已经生成好了。

还可以通过命令： ls 查看当前文件夹下的目录。

2、生成密钥：ssh-keygen –t rsa –C “623578381@@qq.com”

注意这里的C一定要是大写的。

之后会提示要你输入密码：直接3个回车就行了，表示密码为空。

3、在github上添加ssh密钥，找到github官网里的Settings -> SSH keys ,然后这里要添加的是“id\_rsa.pub”里面的公钥。Title 随便填。

4、github官网上是可以添加多个SSH key的，但是不同的电脑不能用同一个ssh key。都要重新独立生成。

## 3、远程仓库和本地仓库的关联同步：

有两种方式关联：

1. **使用添加远程仓库的方式：（注意远程仓库和本地仓库的根目录要同名）**

首先，在github网站上建好一个仓库（当然也可以是任何远程主机），github上新建的仓库会提示你是否新建时建一个README.md文件，可以不用勾选，建一个空的仓库就好了。

------------------------------

然后在本地也建好一个仓库：步骤流程如下：首先，用git bash 命令行cd定位到要创建仓库的目录里。

然后，运行命令：git init ——初始化。

然后，运行命令：git add README.md ——把仓库目录下的已有文件一次添加到缓存。可以一多次添加多个文件，还有注意文件的路径。

然后， 运行命令： git commit -m “一些提交的说明文字” ——这个说明文字一般是强烈建议要求填写的。

然后，git remote add origin **repository URL。 ——这里就是添加远程仓库与本次仓库进行关联。示列：**git remote add origin [git@github.com:liuyaoao/LearnNotes.git](mailto:git@github.com:liuyaoao/LearnNotes.git)

最后， 运行命令：git push -u origin master ——把本地提交的内容一次性推送到远程仓库去同步。由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

1. **把一个远程仓库克隆到本地。**

**这个远程仓库一般是已经有内容了的。**

**然后运行命令：**git clone [git@github.com:liuyaoao/test.git](mailto:git@github.com:liuyaoao/test.git)

拉取远程仓库：**git** pull [remoteName] [localBranchName]

推送远程仓库：**git push** [remoteName] [localBranchName]

## 4、有关分支的命令：

git branch ——查看当前项目的所有分支。并且会在当前分支前加一个\*号。

git checkout branch\_name ——切换分支。branch\_name 为分支名。

git checkout -b branch\_name ——新建一个新分支，并且切换到这个新分支。

删除分支：**git** branch -d [name] ---- -d选项只能删除已经参与了合并的分支，对于未有合并的分支是无法删除的。如果想强制删除一个分支，可以使用-D选项

合并分支： **git** merge [name] ----将名称为[name]的分支与当前分支合并

**1、创建远程分支(本地分支push到远程)**  **git push** origin [localBranchName:remoteBranchName] ——提交本地的一个分支localBranchName作为远程的remoteBranchName分支 //如果远程没有localBranchName，远程的github就会自动创建一个localBranchName分支作为新的远程分支。

删除远程分支：**git push** origin :remoteBranchName 如果”:”冒号左边的分支为空，那么将删除”:”冒号右边的远程分支。

**2、如果是没有“：”冒号的情况推送：**

git push origin master的意思就是上传本地当前分支代码到master分支。git push是上传本地所有分支代码到远程对应的分支上。也就是说master还可以是远程的其他分支，如：develop分支。

## 5、修改、添加、提交、push推送的流程。

在一个分支上修改或添加了文件时，也分为两种情况：

1. 这是个新的本地的分支，还没有推送到远程的分支，就是所远程还没有你这个分支的。这种情况下的流程是：

运行：git checkout –b temp ——temp是为新分支名。

修改或添加文件后。。。。。。。。。

运行：git add filepath/filename.xxx ——add后为根目录开始的文件路径和文件名。

运行：git commit –m “提交注释说明”

由于这个分支是新的本地分支，没有push到远程的，所以这个分支上的所有修改到提交这一步后是不能push（推送）到远程的，要先切换到远程已有的分支，然后再合并到这个远程已有分支，再在这个远程已有分支上push到远程。

提交之后的流程：

运行： git checkout master ——切换到其他远程已有的分支。

运行： git merge --no-ff -m "提交注释说明" temp —— （--no-ff）表示不适用fast forward模式合并。还有一种合并方式是默认的使用fast forward模式合并的：

运行：git merge temp ——表示把temp分支合并到当前分支。

最后在push到远程，运行： git push origin master ——把master 上的修改推送到远程origin上。

当然这里的origin 和master 可以是远程已有的任何分支。到时最终还是要合并到master 分支，然后在push到origin 上的。

是否使用 fast forward 模式合并的区别：

通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

1. 在远程已有的分支上修改，添加文件时：

流程为：

运行：git add filepath/filename.xxx ——add后为根目录开始的文件路径和文件名。

运行：git commit –m “提交注释说明”

最后就是直接推送了，运行：git push origin master ——把master 上的修改推送到远程origin上。

当然这里的origin 和master 可以是远程已有的任何分支。到时最终还是要合并到master 分支，然后在push到origin 上的。

## 6、 关于贮藏当前分支的工作现场，修改临时紧急bug的流程。

应用场景： 如果你在当前分支的工作还没有完成，有来了一个临时紧急bug要改。

这样就可以用 stash (贮藏)的功能。可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作：

运行：git stash ——储藏当前分支的工作现场。

运行：git status ——查看工作区，就是干净的（除非有没有被Git管理的文件）。

然后切换到你要修改bug的那个分支，在这个分支上新建一个临时分支：

运行： git checkout master ——切换到master分支。

运行：git checkout –b issue-101 ——新建一个issue-101分支并切换到这个分支。

修改完bug后：

运行： git add filepath/filepathName.xx ——缓存当前修改了bug的一些文件。

运行： git commit –m “提交注释说明” ——在这个临时分支上提交（当然这个临时分支远程是没有的，所以不用push操作）。

运行： git checkout master ——bug修复好后，重新切换回master分支。

运行： git merge --no-ff -m "提交注释说明" issue-101 ——把有修改提交bug的那个临时分支合并到当前master分支。

运行：git branch -d issue-101 ——合并完以后就可以删除这个临时分支了。

当然，到这一步了可以先把这个合并到master分支的bug push推送到远程，也可以以后在push。

重点来了，怎么把以前贮藏了工作现场的分支重新恢复工作：

首先，切换到以前的那个分支：

运行： git checkout dev ——切换到以前的分支。

运行： git stash list ——可以先查看一下贮藏在这个分支的所有工作现场。

然后恢复某个工作现场，有两种方式：

1. 恢复后不删除：

运行：git stash apply ——恢复工作现场，如果有多个工作现场，可以用：

git stash apply stash@{index} ——指定恢复某个现场。

删除运行： git stash drop ——删除工作现场，如果有多个也可以指定删除那个工作现场。

1. 恢复的同时删除：

运行： git stash pop ——恢复工作现场，同时删除这个贮藏。如果有多个可以指定恢复并删除哪一个：运行：git stash pop stash@{index} 。

7.git 命令详解：

git add :

**git add –u or git add –update :** 表示将已跟踪文件中的修改和删除的文件添加到暂存区，不包括新增加的文件.

**git add –A 或者 git add –all :**  表示将所有的已跟踪的文件的修改与删除和新增的未跟踪的文件都添加到暂存区。

**还有个git add –i命令，或者是git add --interactive 暂时不清楚怎么用。**

**git commit [options]:**

**-a**

**--all**

Tell the command to automatically stage files that have been modified and deleted, but new files you have not told Git about are not affected.

**git commit –m “提交说明”:这是最常用的命令。提交本次修改or新增or删除文件，并添加说明。**

**git commit –a –m “message” ： 只提交修改或删除的，不提交新增的，而不管是否已经在暂存区。**

**git reset:**

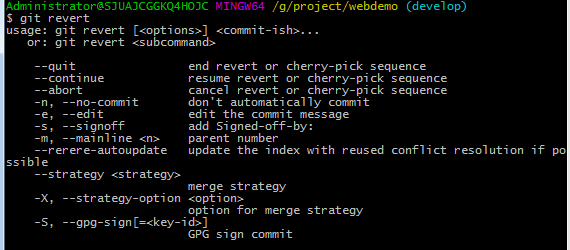
**git reset HEAD: 把所有暂存的文件改成未暂存的状态。**

**git reset --hard HEAD: 如果文件处于暂存状态，使用该命令则会直接丢弃所有文件的修改。**

**git reset --soft HEAD^ :把最后一次提交（还没push）撤销掉，把该次提交的文件改成为暂存状态。**

**git reset --hard HEAD^:这会把最后一次提交（还没push）直接撤销掉，并且会直接丢弃这些文件的修改。所以：注意：一般最好是不要使用--hard选项。**

**git revert :**



**git revert –n commit-ish: 撤销某次完整提交（指已经push了的），并且不会自动生成一个新的回滚提交，而是把这次提交回退到暂存状态，可以重新修改后再提交。**

**git log:**

**git log –n x或者git log --max-count x: 限制只显示最近的多少条提交记录。X:表示条数。不加x的话默认会显示6条。**

**接参数：**

**--since=<date>**

**--after=<date>**

Show commits more recent than a specific date. 查询某个时间之后的，也就是从当前现在到这个时间之间的提交记录。

**--until=<date>**

**--before=<date>**

Show commits older than a specific date. 查询某个时间点之前的记录。

**--author=<pattern> ：根据提交的作者查询提交记录。**

**--grep=<pattern> ：根据提交的message说明查询记录。**