目录

[1、git基本操作 1](#_Toc19801160)

[2、使用总结 4](#_Toc19801161)

# 1、git基本操作

git init 本地初始化一个git仓库

git clone  [git@github.com:lavor-zl/Github-Git.git](mailto:git@github.com:lavor-zl/Github-Git.git) 从远程克隆一个仓库仓库到本地

git add modify-file 添加工作目录的改动文件到暂存区

git add . 添加工作目录所有的改动到暂存区

git status 显示工作树的状态，一般有三种状态

Untracked files 未被跟踪的文件，表示是工作目录新增加的文件

Changes not staged for commit 工作目录中修改了文件，但是没有被添加到暂存区

Changes to be committed 添加到暂存区的文件，等待提交

git config --global –e：给命令创建别名

## 1.1 Commit 命令

git commit –m “提交备注” 提交暂存区的文件，带有提交信息

git commit –a –m “提交信息” 跳过暂存区，直接提交工作做目录中所有改过的文件，但是不能提交工作做目录中新增的文件

git commit --author =lavor –m “提交信息” 提交暂存区的文件，并重写提交作者

git commit --date= 06.13.2016T09:00:00 -m "提交消息"：提交暂存区的文件，并重写提交日期

git commit --amend –m “提交信息” 修改最后一次commit提交的功能,不增加提交历史，也可写为git commit --amend --no-edit，表示提交信息不修改，继续使用上一次的提交信息

## 1.2 reset 命令

git reset 将暂存区的所有文件重置到当前分支的HEAD，将HEAD指针指向上一次提交的点（位置）

git reset <commit> files :将暂存区的指定文件重置到指定的<commit>,<commit>既可以是commit的hash（或者hash的前七位）也可以是HEAD及其祖先，HEAD~1表示HEAD的父节点，即上一次提交，没有指定commit则默认为HEAD

git reset [--hard|soft|mixed|merge|keep][<commit>]:将当前分支指定到commit，并根据mode有可能更新暂存区和工作做目录，mode的可取值可以是hard、soft、mixed、merge、keep

hard：重置暂存区与工作目录到指定提交，删除commit之后的所有提交并将HEAD指向该提交，此操作危险指数较大

soft：暂存区与工作目录不会变化，仅仅删除commit之后所有的提交并将HEAD指向该提交

mixed：默认的，重置暂存区到指定提交，删除commit之后的所有提交并将HEAD指向该提交

merge：keep暂无

## 1.3 rm删除命令：

git rm files: 删除工作目录的文件

git rm --f files: 强制删除工作目录的文件，不做更新检查，之后不做跟踪

git rm --cached files:删除暂存区的文件，之后不做跟踪，主要用作停止跟踪某些不需要上传的文件

**mv命令**

git mv oldfile newfile:为文件重命名

git mv files dir:移动文件到指定目录

git mv –f oldfile newfile :强制为文件重命名，即使目标文件已存在

git mv –f files dir :强制移动文件到指定目录，即使目标文件已存在

**分支与合并**

## 1.4 Branch命令

Git branch :查看所有分支

Git branch branchname :创建分支

git branch branchname <commit>:以特定提交为基本创建分支

git branch –d branchname :删除分支

git branch –D branchname :强制删除分支

git branch –m [oldbranchname] newbranchname :移动或者删除分支

git branch –M [oldbranchname] newbranchname :强制移动或者删除分支

git branch –r :列出所有被跟踪的远程分支

git branch –r –d branchname :删除被跟踪的远程分支

git branch –a :列出所有本地分支与被跟踪的远程分支

git branch --set-upstream-to=origin/master(远程分支) master(本地分支)

git branch -m oldName newName：修改分支名称，未推送到远程仓库

## 1.5 checkout命令

git checkout :检查本地仓库分支与远程仓库分支的差异

git checkout branchname|tagname :切换分支或切换到指定tag，同名时优先切换分支

git checkout tags/tagname :切换到指定tag

git checkout –b branchname :新建分支并切换到该分支上

git checkout –B branchname :强制新建分支并切换到该分支上

## 1.6 merge命令

git merge branchname : 将指定分支合并到当前分支上（默认附带提交操作作），在merge后面使用-m<message>,表示提交信息

git merge <commit> :将指定提交合并到当前支上

git merge oldbranchname newbranchname :将oldbranchname分支合并到newbranchname分支上

git merge --squash branchname:将指定分支上所有修改放入当前分支工作目录与暂存区，不提交

git merge no--squash branchname :将指定分支 上所有的修改放入当前分支工作目录与暂存区，并提交

**mergetool命令：**

git mergetool :在merge命令执行之后出现冲突时执行该命令，用来解决合并的冲突，冲突解决完了不要忘了提交，需要使用三方对比工具

## 1.7 fetch命令：

git fetch :下载远程仓库“origin”到本地，**不会自动merge**

git fetch remoterepository ：下载指定远程仓库到本地

git fetch remoterepository branchname :下载在指定远程仓库分支到本地

## 1.8 pull命令（pull命令相当于fetch与merge命令的组合）：

git pull 从远程获取最新版并merge到本地

git pull remoterepository branchname[:localbranch]:拉取指定远程仓库指定分支到本地仓库指定分支（默认是当前分支）

## 1.9 push命令：

git push remoterepository localbranch[:remotebranch] [--tags] :推送本地指定分支到远程仓库指定分支（默认是与本地分支同名的远程分支），默认是不推送标签到远程仓库的，加上—tags才会推送标签

git push origin localbranch :如果远程分支被省略，则表示将本地分支推送到与之存在追踪关系的远程分支（通常两者同名），如果远程分支不存在则会被新建

git push origin :remotebranch :如果省略本地分支，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支，等同于git push origin –delete remotebranch

git push origin ：如果当前分支与远程分支存在追踪关系，则本地分支与远程分支都可以省略，将当前分支推送到origin仓库对应的分支

git push ：如果当前分支只有一个远程分支，那么主机名都可以省略

## 2.0 remote命令 ：

git remote :查看所有与本地仓库关联的远程仓库

git remote –v:查看所有与本地仓库关联的远程仓库，并显示url

git remote add remote-name remote-url:添加与本地仓库关联的远程仓库

git remote rename oldname newname:为远程仓库重命名

git remote remove remote-name:移除远程仓库

## 2.1 rebase命令：

rebase命令一定要在push之前使用，可以使提交历史更简洁，被合并分支的记录会被删除

git rebase branchname :将指定分支上所有修改应用到当前分支之上

git rebase branchname branchname :将第一个指定分支上所有修改应用到第二分支上

git rebase –i commit编号：i代表交互式操作，以第一个不是自己的提交为基准，合并基准线之后的所有提交为一个新的提交（1-2-3-4四次提交，合并234，则以1的commit编号为基准线，则234合并）

pick：保留该commit（缩写:p）  
　reword：保留该commit，但我需要修改该commit的注释（缩写:r）  
　edit：保留该commit, 但我要停下来修改该提交(不仅仅修改注释)（缩写:e）  
　squash：将该commit和前一个commit合并（缩写:s）  
　fixup：将该commit和前一个commit合并，但我不要保留该提交的注释信息（缩写:f）  
　exec：执行shell命令（缩写:x）  
　drop：我要丢弃该commit（缩写:d）

## 2.2 stash命令

git stash: 把现有的修改藏起来，不包含未被跟踪的文件

git stash –m :“测试”: 把现有的修改藏起来，并添加注释

git stash save :“测试”: 把现有的修改藏起来，不能精确的指定暂存某个文件

git stash push:文件 –m “测试”：精确指定暂存某个文件

git stash push –u: 将未跟踪的的文件也暂存

-a：将所有改动的文件暂存

-k：表示不重置暂存区

git stash list:查看多次暂存的内容

git stash apply :应用最新的暂存内

git stash apply stash@{1}:指定应用某个暂存内容

git stash pop :应用最新的暂存内容，并将暂存删除

git stash pop stash@{1}：指定应用某个暂存内容，并删除

git stash pop --index：可实现重置暂存区功能

git stash drop stash@{1}:删除指定的暂存

git stash clear:删除所有的暂存

# 2、使用总结

## 1、开发中需要切换分支，但已开发部分暂不提交

开发中如修改不提交只能切换到母分支，且所作修改会一同带到母分支，当前分支还原为初始状态，如果母分支进行提交操作后则只有母分支拥有已改动部分，当前分支保持未修改状态，亦或者当前分可以以自身为母模板，创建一个新分支亦可，但改动只体现在提交模板上，若当前分支创建一个子分支，在最后提交由当前分支操作，则子分支为当前分支未修改时的状态，除此两种情况当前分支在未提交的情况下不能切换到其它分支

## 2、合并分支的使用总结

如果合并分支时母分支已经修改或者合并其它分支了，此时再合并分支如果分支与已经修改或者合并的分支没有修改重合的文件则可以直接合并没有冲突，如果有重合的文件，那就解决冲突再重新提交吧