Git学习笔记

1. 初始化一个Git仓库，使用 **git init** 命令，将**当前文件路径作为Git仓库(repository)**；
2. 使用 **git add <file>** 命令添加文件到文件仓库，**可以反复多次提交**；
3. 使用 **git commit –m <message>**，提交文件到文件仓库，其中message为本次的提交说明，用 **双引号””** 引起来；
4. 使用 **git commit –am <message>** 命令用于提交跟踪过的文件，也就是可以跟踪文件夹内的文件路径以及内容变化，可以先使用 **git status** 命令查看git的状态后再决定使用 **–m** 还是 **–am** 选项命令；
5. 使用 **git status** 查看当前仓库工作区的状态；
6. 使用 **git diff** 查看文件被修改的详细内容，可使用 **git diff HEAD -- readme.txt** 命令对工作区与版本库里面最新版本的**readme.txt**文件的区别；
7. 使用 **git log** 查看从最近到最远的提交日志，在后面添加 **–pretty=oneline** 参数可以简化结果显示，最前面的列为 commit id (版本号)，是经过SHA1计算出来的一个数字，用十六进制表示；
8. 在Git中，使用 **HEAD** 表示当前版本，上一个版本为 **HEAD^**，上上一个版本为 **HEAD^^**，以此类推，当^数量过多时简化显示，如往上100个版本显示为 **HEAD~100**；
9. 使用 **git reset** 回退版本，后面添加 **--hard HEAD^** 表示回退到上一个版本，以此类推；
10. 倘若需要**撤销回退操作**，则需要在命令行窗口没有关闭的时候往前寻找到需要回撤到的那个**版本号(commit id)**，例如版本号为**1094adb……**，则需要通过 **git reset --hard 1094ad** 进行回撤；
11. 如果命令行窗口已经关闭，则需要通过 **git reflog** 命令对你操作的每一次命令进行查看（查看命令历史），寻找到响应的**commit id** 进行恢复；
12. Git的工作区和暂存区（相关文件名词解释）：<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013745374151782eb658c5a5ca454eaa451661275886c6000>；
13. 使用 **git checkout -- <file>** 命令对工作区的修改全部撤销，即将文件恢复到最近一次 **git commit** 或者**git add** 时候的状态，此命令必须要有 **--** ，否则会变成“切换到另一个分支”命令；
14. 使用 **git reset** 命令也可以将暂存区的文件修改回到工作区，例如 **git reset HEAD <file>** 命令可将位于暂存区的文件（已经在最新提交版本HEAD的基础上修改过并经过add操作添加到了暂存区）回退到最新提交的版本(HEAD)；再通过 **git checkout -- <file>** 命令可以丢弃对工作区的修改；
15. 使用 **git rm <file>** 命令对版本库中的文件进行删除操作；
16. 在**用户主目录下**使用 **ssh-keygen –t rsa –C “youremail@example.com”** 命令生成SSH Key文件，用于与远程仓库**github**或**码云**的传输加密操作，查看用户主目录下是否存在**.ssh**目录，在目录中是否有**id\_rsa**(**私钥，不可泄露**)以及**id\_rsa.pub**(**公钥，可公开**)这两个文件；
17. 添加github远程库：<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013752340242354807e192f02a44359908df8a5643103a000>；
18. 通过 **git push origin master** 命令提交本地master 分支的最新修改至GitHub；
19. 注意第一次**clone**或者**push**命令时会有SSH警告：



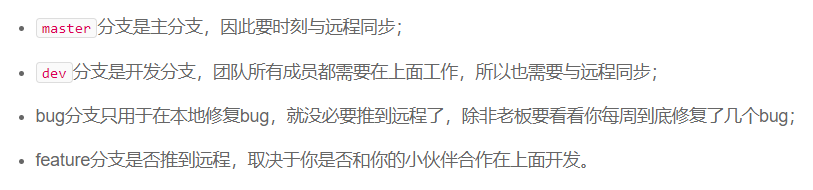
1. 要关联一个远程库，使用命令**git remote add origin git@server-name:path/repo-name.git** ，其中的**origin**可以修改，关联后，使用命令**git push -u origin master**第一次推送**master**分支的所有内容，此后，每次本地提交后，只要有必要，就可以使用命令**git push origin master**推送最新修改；
2. 使用 **git clone git@server-name:path/repo-name.git** 命令对github仓库进行克隆；
3. Git分支命令（Git使用指针来指向分支，因此对于分支的操作非常快）：

* **git branch** ，查看当前已有分支；
* **git branch <name>** ，创建名为name的分支；
* **git checkout <name>** ，切换到名为name的分支；
* **git checkout –b <name>** ，创建+切换到名为name的分支；
* **git merge <name>** ，合并某名为name的分支内容到当前分支；
* **gie branch –d <name>** ，删除名为name的分支；

1. 使用 **git log --graph --pretty=oneline --abrev-commit** 命令查看分支的合并情况；
2. 使用 **git log --graph** 命令可以查看分支合并图；
3. 使用 **git merge --no-ff –m <message> <name>** 命令表示**禁用Fast forward模式**合并分支（使用**普通模式**合并），其中message表示新创建的commit的描述，本次合并将名为name的分支合并到当前分支上（**普通模式**合并可使用 **git log --graph** 命令查看到合并的过程，而使用**Fast forward模式**合并则无法看到）；
4. 遇到合并冲突的时候，必须先处理冲突的文件内容，然后add到暂存区再进行commit操作，最终才会同步到当前分支上；
5. 使用bug分支（当前工作区存在未提交的已修改文件，但是又需要紧急抢修bug，因此需要创建新的bug分支对主分支进行bug修复）：

* **git stash** ，将当前工作现场进行“储藏”，等工作现场恢复后再继续工作；
* 保存工作现场后，切换回到需要修复bug的分支，创建新的修复bug分支，然后进行bug修复工作，修复完成后，合并到需要修复bug的分支中，再切换回到已经“储藏”了的工作分支，这个时候工作分支中使用 **git status** 命令查看状态并没有发现有修改过的痕迹；
* 使用 **git stash list** 命令查看已经“储存”了分支信息；
* 恢复stash内容方法 ：
  + 使用 **git stash apply** 命令恢复，恢复后，stash内容并不删除，需要使用 **git stash drop** 命令来删除；
  + 或者使用 **git stash pop** 命令来恢复，同时将stash中的内容也删除；
* 当已经使用多次stash进行保存的时候，可先使用 **git stash list** 命令进行查看，再使用 **git stash apply stash@{0}** 命令进行恢复；
* 小结：修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；当手头工作没有完成时，先把工作现场 **git stash** 一下，然后去修复bug，修复后，再 **git stash pop** ，回到工作现场。

1. 使用 **git branch –D <name>** 命令强行删除名为**name**的已经完成但是并未合并到主分支的分支（命令中的D为大写）；
2. 使用 **git remote** 命令查看远程仓库的信息，后面添加 **–v** 选项显示更加详细的信息，即使用 **git remote –v** 显示远程仓库详细信息；
3. 使用 **git push origin <branch-name>** 命令将名为branch的分支内容推送到远程仓库；



1. 当推送的分支与远程仓库中存在冲突时，需要先使用 **git pull** 命令将远程仓库中的内容抓取到本地，解决冲突后再进行推送；
2. 若 **git pull** 命令也失败了（提示：**no tracking information**，原因是没有指定本地分支与远程分支的链接），则需要使用 **git branch --set-upstream-to=origin/<branch-name> <branch-name>** 或 **git branch --set-upstream <branch-name> origin/<branch-name>** 或 **git branch --set-upstream-to <branch-name> origin/<branch-name>** 命令后再进行**pull**操作，其中**origin/branch-name** 为**远程仓库分支的名称**，**branch-name** 为**本地仓库分支的名称**，本地和远程分支的名称最好一致；
3. 使用 **git pull** 命令成功后，先合并冲突，然后再add、commit，最后在push；
4. 使用 **git log --graph --pretty=oneline --abbrev-commit** 查看历史提交的合并路径图；
5. 使用 **git rebase** 命令可以将本地未push的分支提交历史整理成直线；
6. 标签管理：

* 首先切换到需要打标签的分支上；
* 使用 **git tag <tagname>** 命令打上新标签，例如 **git tag v1.0** ，表示1.0版本；
* 同时可以使用 **git tag** 命令查看当前所有标签；
* 如果忘记了打标签，想要补上标签的时候，需要使用 **git log --pretty=oneline --abbrev-commit** 命令查看提交的历史日志，然后找到提交的commit id，再通过 **git tag <tagname> <commit id>** 命令对提交的内容进行补打标签；
* 使用 **git tag –a <tagname> -m <message> <commit id>** 命令，对id为commit id的提交记录打上标签名为tagname以及标签说明为message的标签；
* 使用 **git show <tagname>** 命令查看标签详细信息，包括说明文字在内；
  + **注意：标签与某个commit挂钩，若这个commit既出现在master分支，又出现在dev分支，则在这两个分支中都能看到这个标签；**
* 使用 **git tag –d <tagname>** 命令根据指定的标签名删除已有的标签；
  + **注意：所有的标签均储存在本地，不会自动推送到远程仓库；**
* 通过 **git push origin <tagname>** 命令将名为tagname的标签推送到远程仓库，或者通过 **git push origin --tags** 命令将所有本地标签推送到远程仓库；
* 若标签信息已经推送到远程仓库，且需要删除远程仓库的标签，则需要通过 **git push origin :refs/tags/<tagname>** 命令实现操作；

1. Git配置码云教程地址：<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/00150154460073692d151e784de4d718c67ce836f72c7c4000>。

**至此为止，Git的学习就暂告一段落啦！但是，学习是决无止境的，只要不断努力，我相信你还是会离自己的梦想越来越近哒，加油哦ヾ(◍°∇°◍)ﾉﾞ要记住，日积跬步，终至千里，不要让自己的努力白白浪费哦！**