## 本地库初始化

|  |  |  |  |
| --- | --- | --- | --- |
| 命令 | 作用 | 效果 | 注意事项 |
| git init |  | 当前文件下生成一个隐藏文件.git | .git目录下存放的是本地库和相关的子目录、文件。不要删除和随意修改 |
| 1.git config user.name xxx  git config user.email [xxx@xxx.xxx](mailto:xxx@xxx.xxx)  2.git config --global user.name xxx  git config --global user.email xxx@xxx.xxx | 区分不同开发人员的身份  项目级别签名信息保存在.git/config，系统级别签名保存在 home目录下的.gitconfig中(cd ~进入home目录) | 签名：用户名和邮箱地址  1.项目级别/仓库级别：仅在当前本地库范围内有效 2.系统用户级别:登录当前操作系统的用户范围(范围更大) | 这里设置的签名和登录远程库也就是代码托管中心的账号密码没有关系 |
| git status | 查看工作区、暂存区的状态  （工作区是写代码的地方，暂存区是临时存储，历史版本在本地库存储） | No commits yet:没有提交记录,指本地库里面没有提交过的东西  nothing to commit：没有可提交的文件，指暂存区没有文件  Untracked files:未追踪的文件,指新建的文件或者在工作区未提交过的  提交上去的放在本地库，可提交的放在暂存区 |  |
| git add xxx.c | 将文件提交至暂存区 | 暂存区已有该文件，再修改工作区的这个文件后使用这个命令，暂存区的这个文件会更新 | 已经追踪过的文件(提交到缓存区)修改后可以进行提交暂存区或者直接提交本地库的操作 |
| Git rm --cached xxx.c | 将暂存区的文件删除，不会删除工作区文件 |  |  |
| 1.Git commit xxx.c  2.git commit -m “注释” <file> | 将暂存区的文件提交到本地库 | 成功后进入vim编辑器中。在第一行直接输入自己对这次提交的内容的注释  退出编辑器显示修改的内容变化信息。如几个文件改变，改变了几行  2.可以直接在命令行模式下直接加上注释 |  |
| Git log  Git log --pretty=oneline  Git log --oneline  Git reflog | 查看历史版本记录 | 1.显示最完整的HEAD ->master  代表当前版本  2.当版本数多时，以一行的方式显示所以版本  3.以一行显示信息且简写哈希值显示,且只显示该版本以前的版本，后面不显示  4.多了显示指针移动多少步回到某一版本 |  |
| **控制版本前进和后退的方法有三种:**   1. **基于索引值操作[推荐]** 2. **使用^符号 只能往前历史版本走也就是回退历史版本** 3. **使用~符号 也是只能回退历史版本** | | | |
| Git reset --hard <哈希值> | 选择前进或者回退版本 |  | 哈希值不必全写，git reflog打印出来的就可以确定唯一性 |
|  |  |  |  |
| Git reset --hard HEAD^ |  | 几个^往回退几版本 | Git reset 还有其他参数 –soft: 不会改变暂存区和工作区，仅仅1.在本地库移动指针  --mixed：会重新设定暂存区不会碰工作区，所以在1.本地库移动HEAD2指针2.且会重置暂存区  --hard:1.本地库移动HEAD指针，2.重置暂存区 3.重置工作区 |
| Git reset --hard HEAD~3 |  | ~后是几回退几版本 | 删除工作区文件并且只提交到了暂存区  使用Git reset --hard HEAD进行刷新，恢复文件 |
| Git diff <file> | 比较文件的差异。 | 1.当修改一个文件后，使用该命令，会和暂存区比较，打印出修改。  2.提交到暂存区后，再使用该命令没什么效果。  3.然后再次提交到本地库时(git diff HEAD <file>)，比较工作区和本地库某个版本。如果不带file，就会比较工作区所有文件 |  |
| **分支管理:在版本控制中使用多条线同时推进多个任务**  Master :主干  Feature\_xxx:分支1  Feature\_xxx:分支2  Hot\_fix:热修复分支。当出现bug时，创建的分支 | | | |
| Git branch <分支名> | 创建分支 |  |  |
| Git branch -v | 查看分支 |  |  |
| Git checkout <分支名> | 切换分支 |  |  |
| Git merge <有新内容的分支名> | 合并分支 | 1. 切换分支到被增加新内容的上面 2. 执行合并命令 | 例如要将分支完成的任务合并到master，那么现在必须处在master分支上 |
|  | 解决冲突 | 例如当几个分支恰好同时修改同一个文件同一个位置，但不同内容，会产生合并冲突.这时需要人为决定取舍 | 怎么解决：  当出现合并冲突时，1.执行合并命令后，打开合并冲突的文件。  2.然后自行选择删除哪些内容。  3.然后执行git add <file>来标记完成合并，  4.再次使用git commi来退出合并状态（**不能带文件名，注释可以要**） |
| **本地库远程库交互**  **Github邀请别人加入团队的操作: 1.点击Setting 2.点Collaborators然后输入邀请的账号 3.点击添加**  **在git 命令行下输入过一次GitHub账号后不用再次输入是因为账号密码信息记录在windows的凭据管理器。如果想要登录另一个账号那么在管理器里面删除保存的信息即可** | | | |
| Git remote add <别名> <远程库地址> | 为远程库地址起别名。方便记住并向远程库推送 |  |  |
| Git remote -v | 查看所有别名 |  |  |
| Git push <地址或别名> <分支名> | 向远程库推送自己的本地库 | 如果push失败可先执行Git pull –rebase <别名> <分支名> 来进行合并后  再执行这个命令  Push失败的原因可能是远程库先初始化有了一个ReadMe的文件。所以要先进行合并在进行推送 | 如果是另一个人克隆自己的远程库并且想要修改后推送到远程库，那么远程库的作者必须邀请这个人进入自己的团队后，才可以更新远程库 |
| Git clone <远程地址> | 克隆远程库信息 | 1. 完整的把远程库下载到本地 2. 创建”origin”远程地址别名 3. 初始化本地库   这是包含的三个过程 |  |
| Git fetch <别名> <远程分支名> | 远程库的读操作 | 不需要验证身份  **1**.此时从远程库读取的数据保存在 [别名/分支名] 的分支。  **2**.git checkout [别名/分支名]切换到那个分支可以查看 | 这两个经常一起操作  Pull = fetch+merge |
| Git merge <别名/分支名> | 合并远程库和本地库 |  |
| Git pull <远程地址别名> <远程库分支名> | 直接读取远程库信息合并到本地 |  | 当修改的较少时可以使用这个方式来合并，当修改的地方较多时，最好使用上面两个命令。  可以先查看有没有冲突，再进行合并 |
| **跨团队协作**  **修改方：**   1. **点击远程库的fork** 2. **Clone到本地，然后修改后提交到本地库再push** 3. **在GitHub项目点击Pull requests 再点New pull request.最后点击Create pull request** 4. **来到发送邮件界面. 写好后再次带年纪Create pull request**   **要求方：**   1. **收到修改完成消息后，点击项目的 Pull requests** 2. **然后看到邮件，点击邮件，可以发送消息或者直接点merge pull request 进行项目合并** 3. **合并之前可以点击files changed 来进行审核. 合并完成后再pull到本地** | | | |
| SSH登录  **1.进入当前用户的家目录 $ cd ~**  **2.删除.ssh 目录**  **$ rm -rvf .ssh 3.运行命令生成.ssh 密钥目录**  **$ ssh-keygen -t rsa -C atguigu2018ybuq@aliyun.com 4.进入.ssh 目录查看文件列表**  **$ cd .ssh**  **$ ls -lF 5.查看 id\_rsa.pub 文件内容**  **$ cat id\_rsa.pub 6.复制 id\_rsa.pub 文件内容， 登录 GitHub， 点击用户头像→Settings→SSH and GPG keys 7.New SSH Key 8.输入复制的密钥信息 9.回到 Git bash 创建远程地址别名**  **git remote add origin\_ssh git@github.com:atguigu2018ybuq/huashan.git 10.推送文件进行测试 ,git push 时，输入yes确认** | | | |
|  |  |  |  |

## 部署到GitHub步骤

**先确保你的GitHub账户中已经添加本机的SSH Key!**

1. 在GitHub创建一个新的仓库
2. git切换到本地项目所在目录，执行git init 初始化一个本地仓库
3. 关联远程库：git remote add <别名> <远程仓库地址>
4. 进行代码合并，将README.md文件下载到本地仓库.

git pull --rebase <刚才的别名> master(代表分支名这是主)

注意：如果你所在的项目目录下已经存在README.md文件，执行以上命令会报错。提示你，在合并之前，先把之前的README.md移除

5、将项目所有文件先添加到本地仓库中，然后再向远程提交

git add –A

git commit –m “首次提交”

git status