[Git 1](#_Toc6507)

[● 新建仓库，本地推到远程 1](#_Toc19046)

[● 1. 本地最新分支覆盖： 1](#_Toc29732)

[● 2. 远程覆盖本地： 1](#_Toc18952)

[● 3. 创建本地新分支： 2](#_Toc29553)

[● 4. 远程分支修改名称 2](#_Toc18600)

[● 5. 合并修改工程文件 2](#_Toc7328)

[● 6. 还没有push上去，取消本地的commit 3](#_Toc3603)

[● 7. 已经push上去了，修改push上去的commit 3](#_Toc31000)

[● 8. git log 3](#_Toc9116)

[● 特定项目配置-邮箱、用户名 3](#_Toc9285)

[● git pull 和git fetch 4](#_Toc6365)

[● git stash/ git stash pop 4](#_Toc12747)

[● git stash list 恢复指定缓存 4](#_Toc13203)

[● 回滚到某个commit版本 5](#_Toc30322)

[● Git 设置代理 5](#_Toc2909)

[● git remote -v 5](#_Toc20850)

[● 删除本地origin名字 5](#_Toc22625)

[● 一个本地仓库连接gitee和github 5](#_Toc16637)

## Git

* 新建仓库，本地推到远程

|  |
| --- |
| git remote add origin git@github.com:你的账户名/远程仓库名.git  git branch -M master  git push -u origin master |

* 1. 本地最新分支覆盖：

|  |
| --- |
| git fetch --all  git reset --hard |

* 2. 远程覆盖本地：

|  |
| --- |
| git fetch --all  git reset --hard origin/master  //即使此时我的本地分支不是master，也可以用master 覆盖 |

* 3. 创建本地新分支：

|  |
| --- |
| git branch :查看分支  git checkout -b newLocalBranch :本地创建新分支newLocalBranch  > > 切换分支是：git checkout newLocalBranch  git add/commit/  git push --set-upstream origin newLocalBranch :创建远程分支 |

* 4. 远程分支修改名称

|  |
| --- |
| 远程分支重命名 (已经推送远程-假设本地分支和远程对应分支名称相同)  > a. 重命名远程分支对应的本地分支  ```  git branch -m oldName newName  ```  > b. 删除远程分支  ```  git push --delete origin oldName  ```  > c. 上传新命名的本地分支  ```  git push origin newName  ```  > d.把修改后的本地分支与远程分支关联  ```  git branch --set-upstream-to origin/newName  ``` |

* 5. 合并修改工程文件

|  |
| --- |
| > 1. 先将自己的分支<mark>crouch</mark>先push完  >  > 2. git checkout master //切换到主分支  >  > 3. git pull //git fetch 是将远程主机的最新内容拉到本地，用户在检查了以后决定是否合并到工作本机分支中。 而 git pull 则是将远程主机的最新内容拉下来后直接合并，即： \*\*git pull = git fetch + git merge\*\* ，这样可能会产生冲突，需要手动解决。  >  > 4. git merge <mark>crouch</mark> //解决冲突  >  > 5. git diff 查看  >  > 6. git push （不用commit，commit是最新的分支的那次commit） |

* 6. 还没有push上去，取消本地的commit

|  |
| --- |
| > 1. git commit --amend  >  > 2. 进入vim界面，按i进入编辑commit信息，按esc返回，输入 :wq 保存退出。 |

* 7. 已经push上去了，修改push上去的commit

|  |
| --- |
| > 跟上面一样，不过:wq之后，可能需要pull 一下， 用 git diff 可以查看没有问题。之后直接push。中间会提示你输入一个修改的说明原因，一样是vim编辑。  >  > 注意，网上远程修改amend其实是一个push一个一样的新分支，但是显示的话是仓库主页显示你上一次的push，并且commit信息已经修改的。 |

* 8. git log

|  |
| --- |
| > 查看历史，可以用q键退出。 |

* 特定项目配置-邮箱、用户名

|  |
| --- |
| > 1、\*切换到.git目录：\*  > git config user.name XieJiaying  >  > 2、cat config  > 3、git config credential.helper store  > 4、cat config  > 5、\*切换到根目录：\*  > git pull origin (或者其他命令) |

* git pull 和git fetch

[面试官：说说对git pull 和 git fetch 的理解？有什么区别？ · Issue #224 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/224)

|  |
| --- |
| 相同点：   * 在作用上他们的功能是大致相同的，都是起到了更新代码的作用   不同点：   * git pull是相当于从远程仓库获取最新版本，然后再与本地分支merge，即git pull = git fetch + git merge * 相比起来，git fetch 更安全也更符合实际要求，在 merge 前，我们可以查看更新情况，根据实际情况再决定是否合并 |

* git stash/ git stash pop

|  |
| --- |
| git stash将当前未提交的修改(即，工作区的修改和暂存区的修改)先暂时储藏起来，这样工作区干净了后，就可以切换到master分支下拉一个fix分支。在完成线上bug的修复工作后，重新切换到dev分支下通过git stash pop命令将之前储藏的修改取出来，继续进行新功能的开发工作。  **注意，是在dev分支stash后去到其它分支开始操作，好了再回dev分支stash pop。不可以在dev分支stash后再在dev分支pull或者其它操作。（最好再dev选择commit，但不push就好，之后可以选择回退commit版本）** |

* git stash list 恢复指定缓存

|  |
| --- |
| 查询stash 列表：  > git stash list 如下：  stash@{0}: WIP on emcache: c13f985 Merge pull request #12 from username/fixResponseType  stash@{1}: WIP on fixResponseType: 8ba2b28 add and change some logic  可以看到 stash@{0} 是最新的缓存，stash@{1} 是第一次的stash，所以选择指定的缓存恢复的操作如下：  > git stash pop stash@{1}  这个是恢复第一次的缓存 |

* 回滚到某个commit版本

|  |
| --- |
| git log 查看日志  git reset --hard HEAD^ 回退到上个版本  git reset --hard HEAD~3 回退到前3次提交之前，以此类推，回退到n次提交之前  git reset --hard commit\_id 退到/进到 指定commit的sha码 |

* Git 设置代理

|  |
| --- |
| git config --global https.proxy <http://127.0.0.1:10809> git config --global https.proxy <https://127.0.0.1:10809>  git config --global http.proxy 'socks5://127.0.0.1:10808' git config --global https.proxy 'socks5://127.0.0.1:10808'  Git代理取消  git config --global --unset http.proxy git config --global --unset https.proxy |

* git remote -v

|  |
| --- |
| 查看该仓库的所有origin的情况（origin名字和连接远程仓库的地址） |

* 删除本地origin名字

|  |
| --- |
| git remote rm origin  （origin是之前remote add origin自定义的） |

* 一个本地仓库连接gitee和github

|  |
| --- |
| 本质就是设置不同的origin 的名字关联不同的仓库地址  假设我已经配了gitee的情况下，再连接上GitHub仓库  git remote add githubOrigin https://github.com/shAdow-XJY/SCUT2022\_Portal.git  接下来使用命令就要指定origin的名字了  git pull githubOrigin master  git push githubOrigin master |