

Github进行fork后如何与原仓库同步

问题场景：

公司要求所有的代码提交都要先通过自己的仓库提交到主repo上去，所以登录到自己的github网页并fork公司的repo主仓库到自己github，在本地电脑克隆自己github仓库代码到本地工作区，修改代码后提交到自己的github仓库上，然后在自己github网页上发起一个merge request请求，最后等待主repo主人review之后合并到公司的主仓库。

问题是同时也有其他同事在主repo合并代码，所以我要定期和主repo保持同步。

整体思路如下：

在自己的本地电脑添加公司的主repo为上游代码库，注意只是配置原仓库的路径，并没有真正clone原仓库，然后将远程公司主repo同步到自己本地的机器，最后本地的机器再push到自己的远程的fork仓库所有的操作都要在本地命令行完成

步骤 一

fork ---->自己github---->本地工作区

- 1) 首先登录到自己的远程仓库，fork主仓库；这样自己的远程仓库与公司主仓库一致
- 2) 在本地克隆自己github上刚才fork的仓库

```
git clone https://xxxxxx.git //自己的远程github仓库
```

- 3) 然后在本地的机器修改代码再push到自己的远程的fork库

```
git push
```

以上操作只是完成从公司仓库到个人远程仓库的过程，在协同开发的时候还需要下面的神操作

照着这个操作，完全没问题：

步骤 二

我们在进行Github协同开发的时候，往往会去fork一个原仓库到自己的Github中，过一段时间以后，原仓库可能会有各种提交以及修改，很可惜，Github本身并没有自动进行同步的机制，这个需要我们手动去执行，现在我来演示一下如何进行自己的仓库和原仓库进行Gith同步的操作。

Note：以下操作时已经完成步骤一的请款下

- (1) 使用终端 命令行操作。进入项目目录，执行如下命令：查看你的远程仓库的路径。

```
$ git remote -v
origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
```

- (2) 配置原仓库的路径：

下面这步操作即添加公司主repo为上游代码库

注意一定要cd到你self fork出来的库里面去，比如工程名叫luoluo，那要先cd到luoluo中去，然后才能操作

```
git remote add upstream
https://github.com/original_owner/original_repository.git （公司的github主仓库地址）
```

```
$ git remote add upstream https://github.com/ORIGINAL_OWNER
/ORIGINAL_REPOSITORY.git
```

(3) 再次查看远程目录的位置:

```
git remote -v
```

```
$ git remote -v
origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
```

(4) 抓取原仓库的修改文件:

```
git fetch upstream （指定的分支） //如不加指定的分支，默认是master分支执行效果如下所示
```

执行效果如下所示

```
$ git fetch upstream
warning: Permanently added the RSA host key for IP address '54.255.254.175' to the list of known hosts.
remote: Counting objects: 193, done.
remote: Compressing objects: 100% (137/137), done.
remote: Total 193 (delta 72), reused 102 (delta 35)
Receiving objects: 100% (193/193), 36.90 KiB | 169.00 KiB/s, done.
Resolving deltas: 100% (72/72), done.
From git.liebaopay.com:nest/pegasi
* [new branch]      2018_Branding_index -> upstream/2018_Branding_index
* [new branch]      619css -> upstream/619css
* [new branch]      Jira-TIANMA-223 -> upstream/Jira-TIANMA-223
* [new branch]      Jira-TIANMA-286-FRONT -> upstream/Jira-TIANMA-286-FRONT
* [new branch]      TIANMA-619_CSS -> upstream/TIANMA-619_CSS
* [new branch]      TIANMA-703_appLock -> upstream/TIANMA-703_appLock
* [new branch]      TIANMA-715_AdNetwork -> upstream/TIANMA-715_AdNetwork
* [new branch]      TIANMA-748_format -> upstream/TIANMA-748_format
* [new branch]      branch_fe -> upstream/branch_fe
* [new branch]      dateCon -> upstream/dateCon
* [new branch]      feature -> upstream/feature
* [new branch]      master -> upstream/master
* [new branch]      streaming -> upstream/streaming
* [new branch]      unit -> upstream/unit
```

(5) 切换到master分支。

```
git checkout master
```

```
$ git checkout master
Switched to branch 'master'
```

(6) 合并本地的master分支:

下面这行代码执行结束之后，本地代码会立刻和公司repo主库保持同步，非常神奇

```
git merge upstream master
```

```
$ git merge upstream/master
Updating a422352..5fdff0f
Fast-forward
 README          | 9 -----
 README.md       | 7 ++++++
 2 files changed, 7 insertions(+), 9 deletions(-)
 delete mode 100644 README
 create mode 100644 README.md
```

Note:

```
git merge upstream/master  (这是mac系统的指令)
git merge upstream master  (这是linux/win系统的指令)
```

(7) 此时，你的本地库已经和原仓库已经完全同步了。但是注意，此时只是你电脑上的本地库和你的远程的公司github主仓库同步了，你自己的远程github仓库还没有同步，此时需要使用“git push”命令把你本地的仓库提交到自己的远程github中。

-----这是华丽的分割线-----

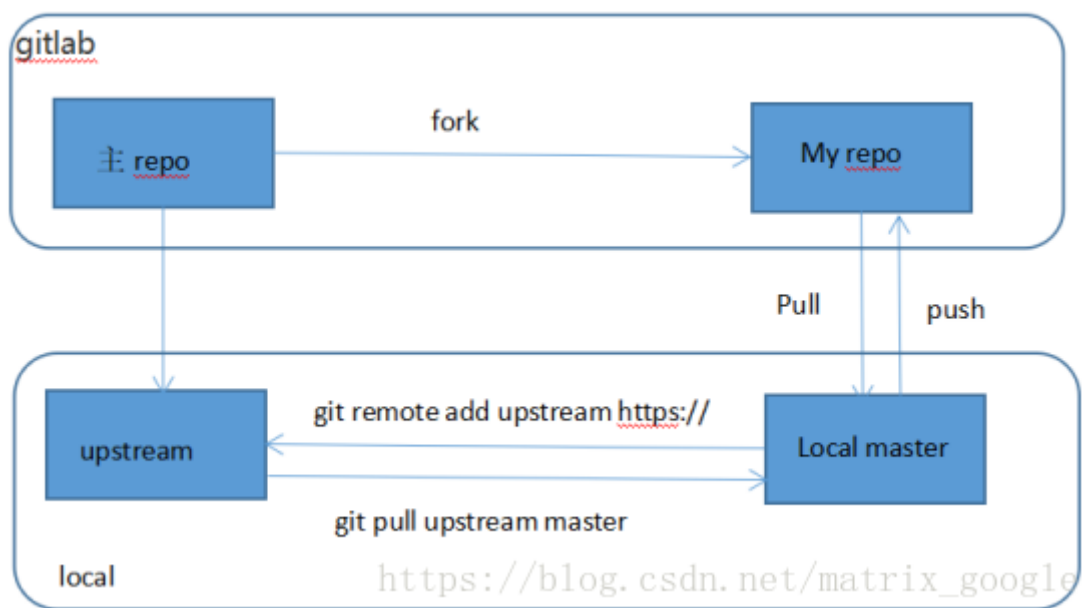
其实(4)(5)(6)可以合并成一条命令

git pull upstream master/(或其他分支)

第一个参数upstream 表示远程主repo

第二个参数master 表示自己fork库的master分支

为了说清楚这个问题，做了一张图



注：每天上班就要做至少一次下面的操作

```
git pull upstream (分支名) //本地工作区代码更新与公司主仓库代码同步
git push //把本地工作区更新后的代码推送到自己个人github, 保持与公司主仓库同步
//假如有在本地更新代码
git add . //把跟新的代码放到本地暂存区
git commit -m "xxxxxx" //备注说明
git push //把更新的代码推送到自己的个人github
登录自己的github, 请求PR
```

如何git 同步代码到另外一个分支

例如: 将master分支的代码同步到develop分支 (从同步本地仓库到推送至远程仓库)

(以下指令都是在本地电脑的仓库目录操作)

```
git checkout develop //转换到develop分支
git merge master //更新到与master一致
git push
```

上面三个指令操作, 就完成了master分支的代码同步到develop分支并推送到自己的远程develop的仓库

删除远程分支指令(也是在电脑终端上执行)

```
git push origin -delete 分支名 //删除远程仓库的分支
git branch -r //查看执行删除分支指令后的分支列表
```

在本地仓库建立了一个分支 (Develop) , 并把这分支推送自己的远程仓库

```
git checkout -b Develop //在本地建立一个Develop分支
git push origin Develop //把本地的Develop分支直接推送到远程主仓库, 自己有操作远程主仓库的权限
```

从远程主仓库克隆代码之后的修改或更新并推送到自己的远程仓库

```
//执行更新远程代码之后 -->git fetch upstream/(对应的分支)
git add . //提交到本地缓冲区
git commit -m "xxx" //添加备注"xxx", 提交到本地文档
```

完成以上两个步骤; 就可以用以下指令推送到自己的远程仓库了

```
git push //吧自己改动或更新的代码推送到自己的远程仓库
```

请求与主仓库合并, 是要登录到自己的远程仓库的操作界面进行PR请求, 并等候审核

注意: 在删除远程分支时, 同名的本地分支并不会被删除, 所以还需要单独删除本地同名分支
如果发生以下错误:

error: unable to delete 'origin/xxxxxxxx-fixbug': remote ref does not exist

error: failed to push some refs to '[git@github.com](https://github.com):xxxxxxxx/xxxxxxxx.git'

解决办法: git checkout xxxxx-fixbug 切换到当前分支上, 然后再进行 git push --delete origin origin/xxxxx-fixbug 此时将不会再发生错误

获取远程主仓库的分支

因为用git clone指令只能克隆master分支，那么可以用以下指令获取主仓库的其他分支

```
//查询远程主仓库的分支指令  
git branch -a
```

```
E:\project_repo\Pressure-level>git branch -a  
* master  
remotes/origin/Develop  
remotes/origin/HEAD -> origin/master  
remotes/origin/master
```

要在本地仓库建立与自己的远程主仓库一致的Develop分支


```
git fetch origin Develop:Develop
```

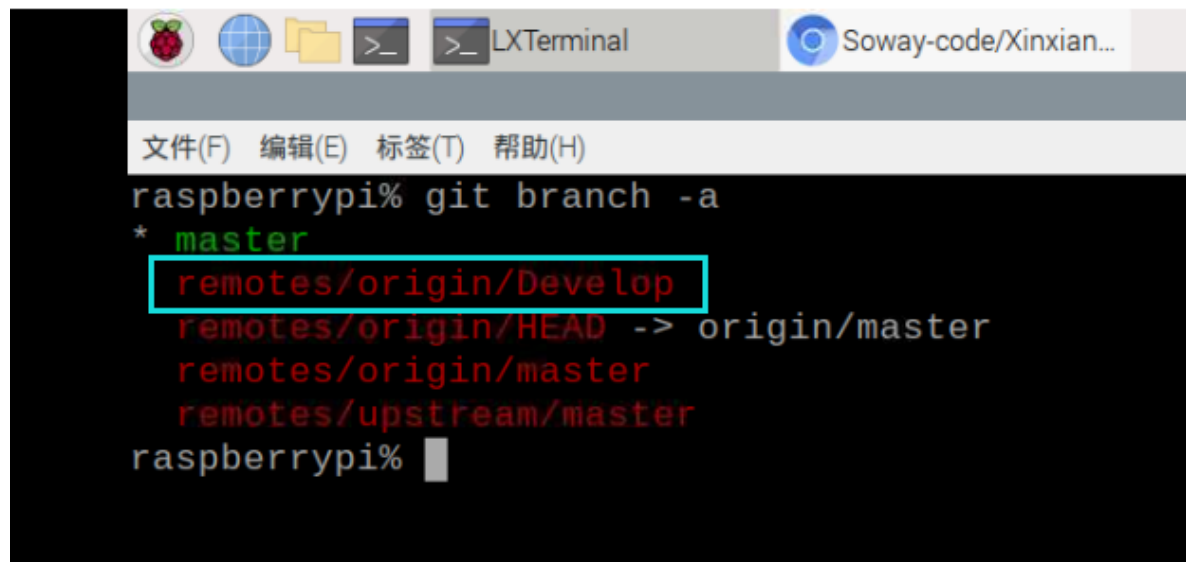
```
E:\project_repo\Pressure-level>git fetch origin Develop:Develop  
Enter passphrase for key '/c/Users/Loo George/.ssh/id_rsa':  
From github.com:Soway-code/Pressure-level  
* [new branch]      Develop    -> Develop
```

```
E:\project_repo\Pressure-level>git branch  
Develop  
* master
```

这样就建立了本地仓库与自己远程主仓库一样的Develop分支（多个分支同样操作）

要在本地工作区（本地仓库）建立与公司远程仓库一致的分支,分两个步骤

 pi (raspberrypi) - VNC Viewer



步骤一：更新公司远程分支到自己的本地分支，还更新到自己的远程分支（登录自己的github发现自己的GitHub已经添加了新的分支）

```
git fetch upstream 分支名称  
或者  
git pull upstream 分支名称
```

```
raspberrypi% git pull upstream Develop
来自 https://github.com/Soway-code/Xinxiang
* branch          Develop    -> FETCH_HEAD
* [新分支]        Develop    -> upstream/Develop
已经是最新的。
raspberrypi% git branch -a
* master
  remotes/origin/Develop
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/upstream/Develop
  remotes/upstream/master
raspberrypi% █
```

把本地工作区建立与自己的远程仓库一致的分支

```
git fetch origin Develop:Develop
git pull origin Develop:Develop
```