Github 使用说明

1. 第一次使用github方法

1.进入对应目录用

git init 命令把这个目录变成Git可以管理的仓库

2. 用命令git add告诉Git，把文件添加到仓库：

git add readme.txt可以使用git add -A添加所有更新的文件

git commit -m "wrote a readme file"

3.由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

ssh-keygen -t rsa -C [hxhaawt@126.com](mailto:hxhaawt@126.com)

需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

Generating public/private rsa key pair.

// 第一次提示信息 按第一次回车 不输任何信息

Enter file in which to save the key (/Users/liangkuiyuan/.ssh/id\_rsa):

// 第二次提示信息 按第二次回车 不输任何信息

Enter passphrase (empty for no passphrase):

// 第三次提示信息 按第三次回车 不输任何信息

Enter same passphrase again:

Your identification has been saved in /Users/liangkuiyuan/.ssh/id\_rsa.

Your public key has been saved in /Users/liangkuiyuan/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:IhNkHkezvAbVAITk2h1lcIqPK22mBYqU1RsS33Uw/Ts kerain.liang@jiuzhang.net

The key's randomart image is:

+---[RSA 2048]----+

| ..+B+Xo +o. |

| ..=+X.oo o. |

| oo\*+o. . |

| oo+.+o. . |

|ooo =.+ S . |

|+o . + . E |

|+ \* . |

| \* |

|. |

+----[SHA256]-----+

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容。

具体操作在这个页面有相关说明：

http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001374385852170d9c7adf13c30429b9660d0eb689dd43a000

2 创建本地仓库和提交到远程github上的方法

第一步：在ginhub网站上创建一个repository；

第二步：点击需要上传的本地工程目录，右击-->>Git Bash，进入命令行；

第三步：$ git init

$ git add --all //添加该文件夹下的所有文件

$ git commit -m ‘这是一个示例’ //引号内是对仓库的描述

$ git remote add origin git@github.com:xxx/yyy.git

//添加远程仓库路径（ssh路径或者http路径）

$ git push -u origin master

由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

注：如何找路径：点击你刚创建的respiratory（仓库），在右下角会有一片区域，复制里面的内容就可以了。

1. Github相关命令

1). 查看状态：git status

2）. 添加所有的文件： git add . / git add –A / git add –all 三个命令都可以

3）. 创建与合并分支

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name> / git push orange :name(分支名)

4). 提交新建分支到远程

git push --set-upstream origin dev

5). 用分支提交代码

Git push origin 分支名 如：git push origin dev

6). 从远程库克隆

在git bash中进入到对应的本地目录，执行以下命令会在 本地创建仓库

git init

然后执行下面命令，克隆hxhaawt用户下 gittest这个项目的所有文件

git clone git@github.com:hxhaawt/gittest.git

或者克隆分支： **git clone -b 分支名仓库地址 如：**

**Git clone –b dev git@github.com:hxhaawt/vue-test.git**

git clone后如何保持与原项目同步？

1).如果不是fork项目的话： git pull

2).如果你是fork的话，把原来的项目添加到 git remote add usp [主项目地址]

然后 git fetch ups && git merge ups/master

7). 同步远程github仓库

1). git fetch / git fetch origin

进入到对应的仓库或者克隆的目录下，然后执行 gti fetch

即可更新项目对应Github 上的文件到本地目录（可能要等待一些时间）。

再执行 git merge 或者 git merge origin/master，即可将文件更新到本地目录。

2). git fetch origin dev 可以只更新dev的分支内容

再执行 git merge 或者 git merge origin/master，即可将文件更新到本地目录。

8). Github 命令设置别名

git config --global alias.st status

为git status 设置了别名 git st

更长的别名设置，如：

git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

创建别名 git lg

9）. 如果远程修改了一个文件同时本地也对这个文件作了修改，这时就要在本地用 git pull

把远程修改的文件down下来，然后对文件这个文件进行修改，再git add

也可以只pull特定分支如 git pull origin dev

10）. 创建本地标签和推送本地标签到远程服务器命令

Git tag v1.0 创建标签

git push –tag 根据需要推送本地标签到远程服务器

1. github 多人协作

情景1：多人协作之如何建立本地分支？

假若你已经clone了别人的仓库并且需要修改，最好的办法是建立自己的分支然后在合并，具体步骤如下：

1.建立一个自己的分支

Git branch mybranch

此时可以使用git branch查看当前的分支情况，如不 出意外，则显示一个master一个mybranchi分支，而你当前在master分支上。

2.切换到新建的分支

git checkout mybranch

此时我们就可以在myabranch随意的修改，这并不会影响刚才的主分支master。注意：以上全部操作都不影响远程仓库，都是本地的备份。

3.假若我们已经在mybranch上做了很多修改。那么我们就可以在mybranch下进行commit

git commit -m "i had do some thing in mybranch"

4.此时我们回到原来的master分支

git checkout master

5.将之前在branch上的修改合并到master

git merge mybranch

6.此时mybranch分支对我们来说已经没用了，删除

git branch -d mybranch

7.此时我们可以一次性将本地master的修改(merge后的)push到远程

git push

注意：如果想新建和切换branch一次完成，可以

git checkout -b branchname

情景2：多人协作之如何建立远程分支？

1.假如我和另一位同事一起操作一个rep(https://github.com/kkkkkxiaofei/trygit.git)，情景1对于我本地来说是很保险的，因为我本地新建的branch修改好了去merge本地的master而后才push到远程，但问题是如果我push的远程的还是有很多问题那就得在远程的master上去修改，其实也可以在远程上新建分支，全部push到这个新的分支上去。

git checkout -b shopping\_cart

此时在本地新建并切换 到了shopping\_cart分支。

2.将本地分支链接到远程分支

git push origin :shopping\_cart

在操作这一步之前远程上是没有shopping\_cart这个分支的，而现在有了。

3.对当前这个分支做些操作然后push

git add 2.html

git commit -m "add 2.html in shopping\_cart branch"

git push

这三行全是在shopping\_cart分支上操作的，远程上现在已经有对应的分支去被push，可以去github上看看在shopping\_cart分支上确实多了一个2.html文件。

4.我觉得我的代码应该没问题，所以我要合并到主分支。

git checkout master

git merge shopping\_cart

git push

5.现在轮到我另外一个同时操作了，他首先克隆可该rep然后查看分支

git clone https://github.com/kkkkkxiaofei/trygit.git

git branch

此时只显示了\* master,为什么呢？因为对于这个同事来说，他的本地的确只有master分支。

6.查看远程分支

git branch -r

显示：

origin/HEAD -> origin/master

origin/master

origin/shopping\_cart

这回他看到了我在远程上建立的shopping\_cart分支。

7.他想操作shopping\_cart分支

git checkout shopping\_cart

此时checkout是把远程的shopping\_cart放到了本地，并且切换到了该分支。

查看本地分支状态可以发现本地的确有了该分支：

master

\* shopping\_cart

此时如果git remote show origin将会看到远程和本地同步了（都各自有两个分支）。

8.此时该同事查看了我在master上push的代码觉得没问题，那么远程的临时分支shopping\_cart就没用了

git push origin :shopping\_cart

显示：

To https://github.com/kkkkkxiaofei/trygit.git

- [deleted] shopping\_cart

这条指令想必很熟悉吧，没错，建立和删除远程分支指令相同。执行指令，没有时就建立，有时就删除。

9.删除了远程分支，对于他而言本地的shopping\_cart更没有用了，删除吧

git branch -d shopping\_cart

如果报错试试 git branch -D shopping\_cart

当然了，如果他觉得我的代码有问题，还可以在8.的时候修改自己的shopping\_cart然后继续push到远程的shopping\_cart。

情景3：撤销已git add的文件？

1.新建一个rep用以测试本例子，并且clone到本地

git clone https://github.com/kkkkkxiaofei/trygit.git

2.进入trygit文件夹下，该文件夹下只有一个README.md文件，手动修改该文件内容后查看diff

git diff

此时由于我修改了该文件，git显示差异如下：

diff --git a/README.md b/README.md

index 8686c6e..62a73fe 100644

--- a/README.md

+++ b/README.md

@@ -1,2 +1,2 @@

-trygit

+trygit by zxf

======

很显然，我是将trygit改为了trygit by zxf

3.此时的README.md文件的状态肯定是unstaged的，可以查看

git status

4.我们现在把该文件stage一下

git add README.md

5.现在查看修改的差异

git diff

发现没有任何内容，为什么呢?因为git diff是检测unstaged文件的差异。

6.现在查看staged的差异

git diff --staged

这时就输出了和2.一样的结果。此时文件的状态是changes to be committed.

7.哦，糟糕了，我想git add的文件并不是README.md，肿么办？

git reset HEAD README.md

此时出现如下提示：

Unstaged changes after reset:

M README.md

说明我们重置了HEAD指针后，README.md变为了unstaged.当然了，现在又可以用git diff了。

8.既然我都git add了错误的文件，那是不是之前的修改也要撤销呢？

git checkout -- READM.md

此时出现：

On branch master

Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

意思太明显了，该文件又回到了以前的原始状态(内容为trygit)，利用checkout可以将一个已经track的file变为untrack.

ps：什么是track呢？只要在当前git工作目录下文件是untrack的，然后利用git add就可以将件变为staged的，而当该文件最终的修改被push后，它就会回到原来的状态，这个原来的状态并不是指它刚建立的时候，那个时候它是untrack，而现在它被push后是变成了track状态了。举个最简单的例子，你新建的文件无论如何修改，你都看不到git diff，但当你把他git add 后，这时git 就可以track到了。

git reset --hard HEAD^ 回退到add distributed版本时

情景4：撤销已经commit的文件？

1.修改1.html然后提交（注意默认该文件已经是track状态）

git commit -a -m "add 1.html file"

-a可以捕捉所有已tack的文件的changes，这样就省去了git add xxx1 xxx2 xxx3的繁琐。

2.哦，糟糕了，我没有把我修改的内容注释写在提交里，我想补充，肿么办？

git reset --soft HEAD^

该指令会将commit状态转变为staging状态，此时git status：

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: 1.html

提示你有修改，可以commit的，那太明显了，这就是staging状态啊。

3.重新修改commit

git commit --amend -m "add 1.html file and modify its content"

这时git log可以看到只有后面这条commit记录，之前的没了。

注意，在这个情景中，其实如若只是把commit的注释写错了，直接使用3.即可，而你如果不单单是修改注释这么简单，还想git add其他文件作为这次commit，那么就需要2.

其他相关指令：

撤销上一次的commit和all changes

git reset --hard HEAD^

撤销上二次的commit和all changes

git reset --hard HEAD^^