使用git进行项目管理

一、Git与Github简介

> - Git是一种版本控制系统,是一种记录若干文件内容变化，以便将来查阅特定版本修订情况的系统。它具有极为丰富的命令集，对内部系统提供了高级操作和完全访问。Git诞生于2005年，由Linux开源社区（特别是Linux的缔造者Linus Torvalds）开发。Git 的特点： — 支持离线开发，离线仓库(Repository) — 强大的分支功能，适合多个独立开发者协作 — 速度块
> - Github是一个网站，给用户提供git服务。 这样你就不用自己部署git系统，直接用注册个账号，用他们提供的git服务就可以。所以只要到www.github.com申请一个github帐号，就可以免费使用git服务。

Github将每一个项目称作一个“仓库”(repository),在仓库里，你可以自由地添加、同步、删除文件，而且可以多人协作对一个仓库中的文件进行修改。横向上，github采用工作流的方式，你的本地仓库由git维护的三棵“树”组成。第一个是你的工作目录，它持有实际文件；第二个是暂存区（Index），它像个缓存区域，临时保存你的改动；最后是HEAD，它指向你最后一次提交的结果；纵向上，github采用主干-分支的流程控制方式，采用多分支实现单人多作和多人协作，可以方便地找回任何一个修改节点的记录。 本文主要介绍如何使用git进行合作项目管理，希望之前没有接触过git和github的朋友可以通过这篇介绍对如何使用git有一定的了解。另外，我们使用命令行的git操作方式，所以没接触过的朋友需要先学会使用简单的命令行操作；如果需要像本文一样进行Markdown文本格式编辑，需要使用相应的Markdown格式；涉及到在vim编辑器中进行代码或文本的修改，需要了解一些vim相关的使用命令。这些我就不再一一叙述，如果还不会的可以自行查阅网上相关资料进行学习。下面正式发车：

二、环境配置

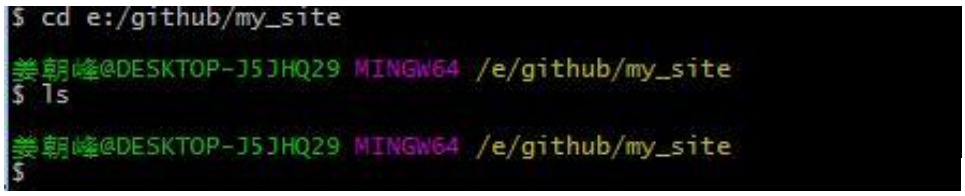
- 第一次接触git和github的朋友，需要先配置环境。首先，到https://git-scm.com/download，根据自己的操作系统选择下载对应的客户端（一般linux已经预安装）。下载后安装、打开，进入git（命令行界面）。 

- 然后，我在自己电脑的E盘建立了一个专门用于git操作文件的文件夹，这个文件夹就相当于github的本地站点，比如我建了e盘的github/my_site文件夹。在命令行中，去到对应的文件夹：

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 ~
$ cd e:/github/my_site
```

于是打开了根目录，输入ls命令，该目录下还没有文件：

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site
$ ls
```



图例2.2

- 接着，我们要建立本地库和github上对应代码库的连接，对应到群主的github代码库中，就需要进行下面的设置。在本地创建ssh key，建立和github服务器的连接：

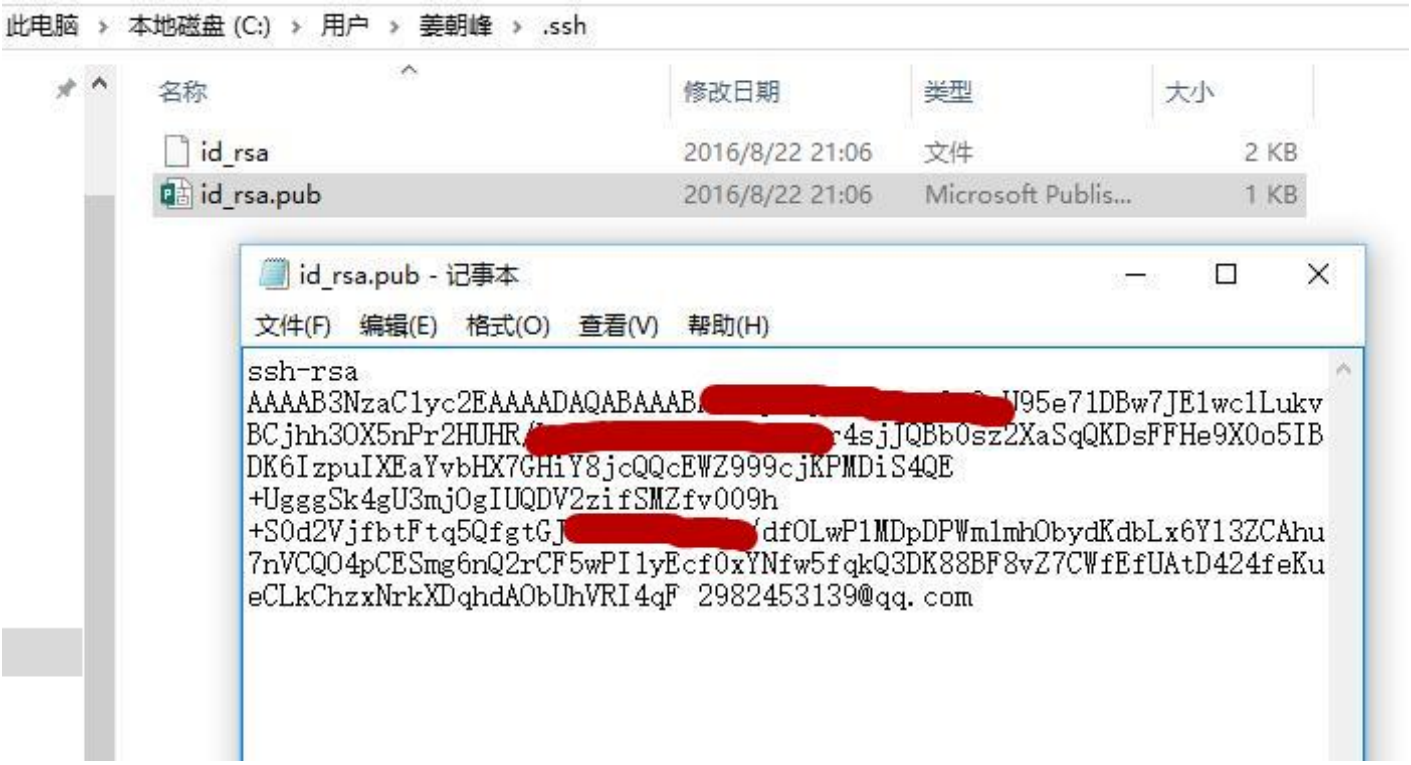
```
$ ssh-keygen -t rsa -C "your_email@youremail.com"
```

后面的your_email@youremail.com改为在github上注册的邮箱（如图），之后会要求确认路径和输入密码，我们默认一路回车就行。

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site
$ ssh-keygen -t rsa -C "2982453139@qq.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/姜朝峰/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/姜朝峰/.ssh/id_rsa.
Your public key has been saved in /c/Users/姜朝峰/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:x8cyUBn9ZssNozN6g9w5nCc2ZFtwtVRSOxQ/supAsFY 2982453139@qq.com
The key's randomart image is:
+---[RSA 2048]----+
|          o+  .=B|
|         .. .  o=|
|        ..E   o.oo|
|         +o .. X..|
|        oS.= oO = |
|       . .. +B + .|
|          o O B   |
|           * & .  |
|            + *   |
+----[SHA256]-----+

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site
$
```

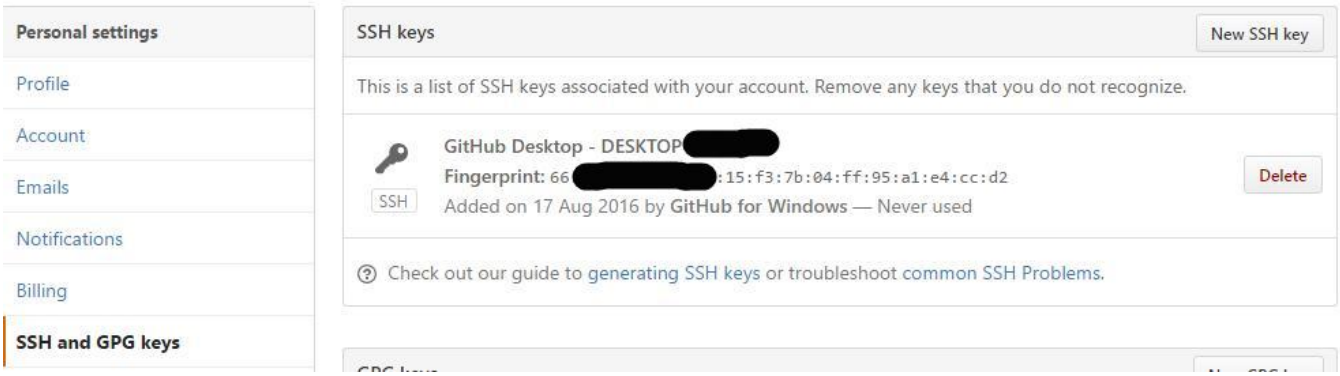
图例2.3

成功的话会在“C:\Users\XXX”下生成.ssh文件夹，进去并打开id_rsa.pub，复制里面的key。



图例2.4

回到浏览器打开github，进入Settings(配置)，左边选择“SSH and GPG Keys”，点击“New SSH Key”,随便填一下title，粘贴生成的key。

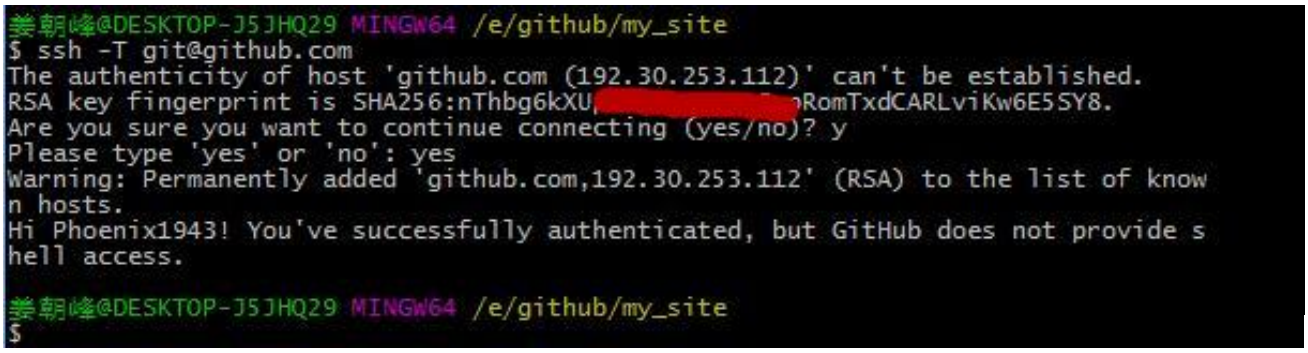


图例2.5

- 为了验证是否成功，在git bash下输入：

```
$ ssh -T git@github.com
```

如果出现下图所示的情况，这就表示已成功连上github。如果要多人协作对这个库进行修改，就需要把合作者的SSH KEY都添加到账户列表中。



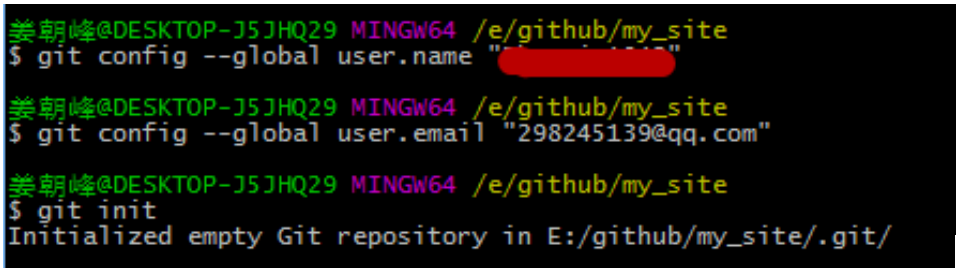
图例2.6

- 在我们能用git工作之前，我们需要还做个一次性的用户配置。为了git能跟踪到谁做了修改，我们需要设置用户名和账户。发送下面的命令，相应地替换掉其中的“your_username”和“your_email@server.com”，改成自己的信息：

```
git config --global user.name "your_username"
git config --global user.email your_email@server.com
```

- 设置好之后，再输入初始化命令，就把当前目录作为git根目录，创建了一个本地仓库：

```
git init
```



图例2.7

最后出现“Initialized empty Git repository in E:/github/my_site/.git/”的指令，说明git的初始化配置成功。Git会在my_site文件夹内创建一个名为.git的隐藏文件夹，那就是你的本地代码仓库。如果不进行初始设置，你就无法提交任何东西。设置好之后，下一次打开可以直接工作，不用再设置一次。不过要是换了电脑，那就需要再设置一遍。

我们希望为荔枝Pi的github项目库增加或修改内容，这就涉及到多人一起在github上的开发。一般有三种方法：

> - 第一种：把各位的公钥加到该项目的公钥列表里；
> - 第二种：在github建一个**orgnization**，然后建一个**team**，把大家加到team并把项目放到team下；
> - 第三种：就是你要修改的人**fork**,然后给群主发**pull request**，等群主通过。 第一种上面的环境配置中已经谈到，第二种可以自行到github了解，这里推荐第三种方式。

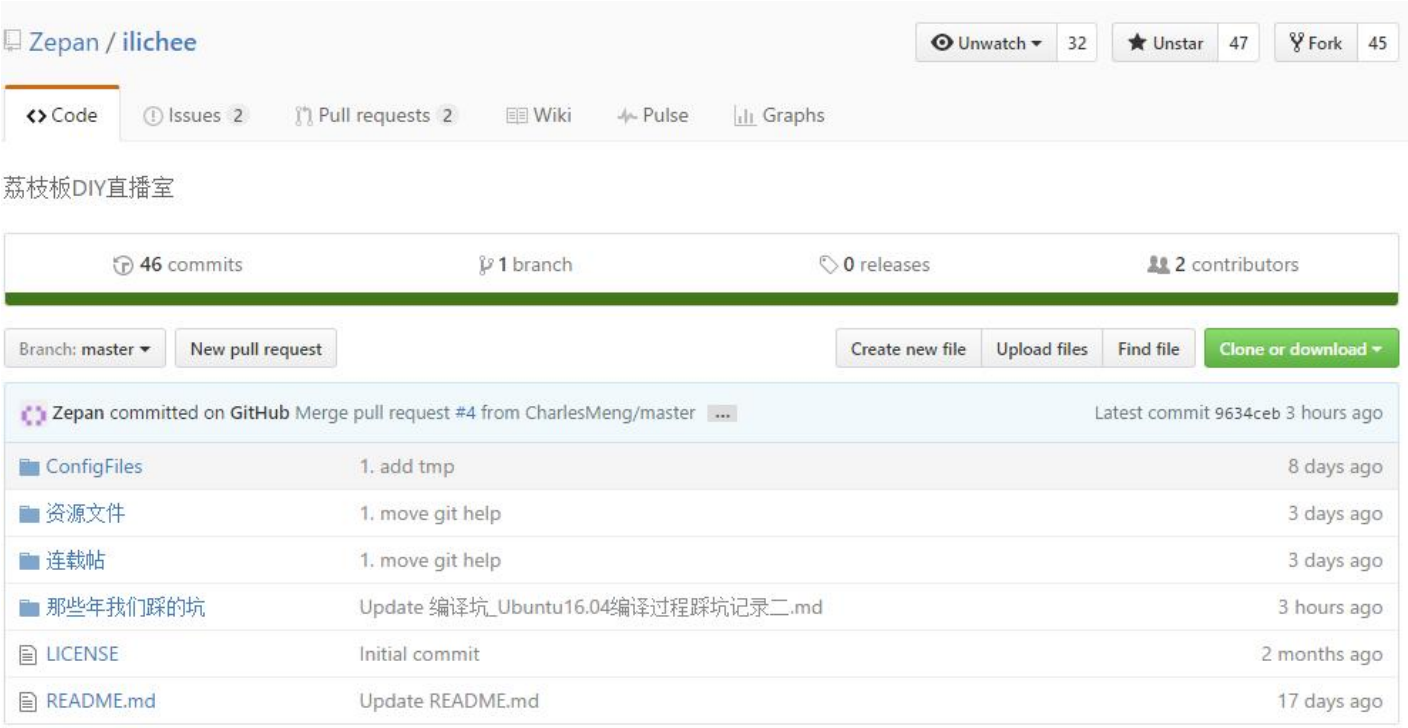
- 到此，就完成了git的环境配置，可以开始具体的工作流程。 ***

三、Git入门准备(以荔枝派为例)

提醒：下面有的操作要在命令行界面，有的要在浏览器中到github主页鼠标操作，注意切换。

1. 复制别人的代码库(fork)

如果想参与到对别人代码库内容的增加或修改中，可以先fork再pull request。**fork**就是复制别人这个项目代码库到自己帐号下；**pull request**后面会谈到。 比如群主A有一個代码库a(也就是荔枝派的github主页)，用浏览器打开：



图例3.1

你叫做B，看到这个觉得不错，所以就**fork**(点击图中右上角的fork图标)一個到自己的代码库中，现在暂且称为代码库b。这时，你查看自己的profile下面的就会多出了一个illichee的代码库。a和b的修改互不干预，大家可以随意修改自己的代码库b，群主的a不会受影响。另外，旁边的“Star”表示持续关注别人项目更新，“Watch”则是设置接收邮件提醒。

图例3.2

2. 复制远程代码库到本地(clone)

但现在，代码库b还是存放在github服务器上，你只能在浏览器中修改，不能本地修改。所以你想把这个远程代码库整个下载到自己的电脑上再修改的话，你需要拷贝**(clone)**它。发送(注意，此时要在根目录下)：

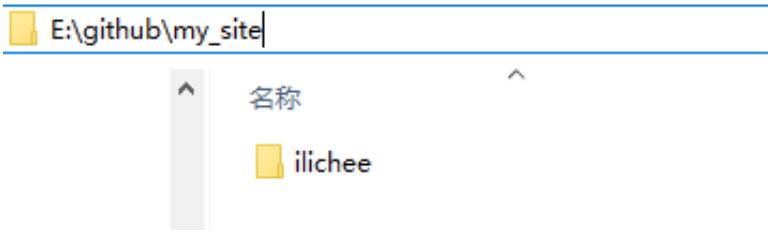
```
git clone git@github.com:your_username/name_of_remote_repository.git
```

得到如图：

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git clone git@github.com:Phoenix1943/ilichee.git
Cloning into 'ilichee'...
remote: Counting objects: 176, done.
Receiving objects:  60% remote: Total 176 (delta 0), reused 0 (delta 0), pack-re
used 176
Receiving objects: 100% (176/176), 8.25 MiB | 32.00 KiB/s, done.
Resolving deltas: 100% (72/72), done.
Checking connectivity... done.
```

图例3.3

果然在这台电脑本地库目录下多了一个“ilichee”的文件夹：



图例3.4

这样你就把代码库b都下载到本地了。以后，可以用相同的方法在不同的电脑上、由不同的人一起完成一个代码库。 另外，如果已经在本地的项目上工作了，只是想从远程代码库上取得它最新的版本，那我移动到项目的根目录下(注意，不在根目录下无法获取)，并发送：

```
git pull first master
```

3. 推送到远程代码库(push)

在第一次你想推送一个本地代码库到远程代码库时，你需要把它添加到你的项目配置里。像这样做：

```
git remote add origin https://your_username@bitbucket.org/your_username/name_of_
```

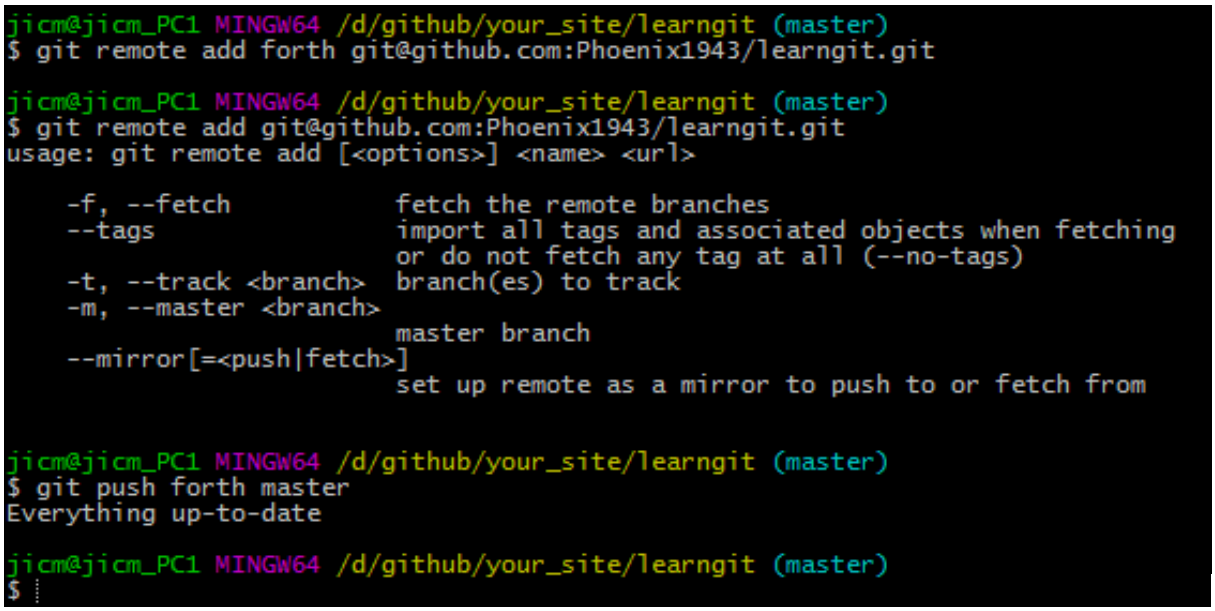
```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git remote add origin3 git@github.com:Phoenix1943/learngit.git

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git push -u origin3 master
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 731 bytes | 0 bytes/s, done.
Total 9 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To github.com:Phoenix1943/learngit.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin3.
```

图例3.5

注意这里的“origin”只是一个习惯。它是你的远程代码库的别名，但是你可以用其他任何你喜欢的词。你甚至可以有多个远程代码库，你只需要给它们起不同的别名。 之后，推送你的本地代码库的主干分支到你的远程代码库：

```
git push origin master
```

下图是对该库文件修改后提交并推送的示例：



图例3.6

4.同步代码库

假设按照上述的流程复制别人的代码库到自己的GitHub(fork)，别人的代码库已经更新，自己复制的代码库有没有新代码可以提交，这时候就需要同步复制别人的源代码库。 进入clone到本地的代码库文件夹中，然后增加源分支地址到你项目远程分支列表中；

```
git remote add source https://github.com/originalL_owner/original_repo.git
```

这里的source是自己给源仓库起的名字，可以起自己喜欢的。

fetch源分支到本地

```
git fetch source
```

然后，切换到本地 master 分支：

```
git checkout master
```

合并两个版本的代码

```
git merge source/master
```

把合并后的代码push到你的Github项目上去

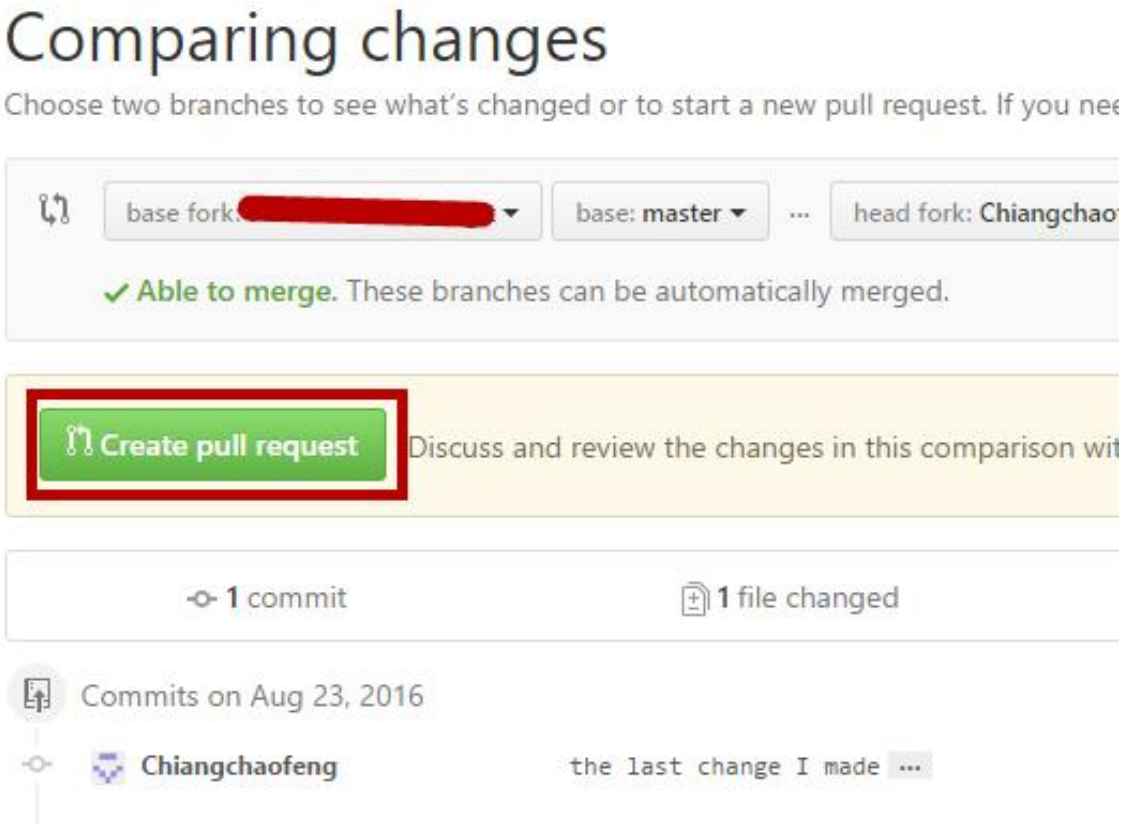
```
git push origin master
```

```
fanwenl_@fanwenl_PC MINGW32 /e/prj/ilichee (master)
$  git remote add source https://github.com/Zepan/ilichee.git

fanwenl_@fanwenl_PC MINGW32 /e/prj/ilichee (master)
$ git fetch source
From https://github.com/Zepan/ilichee
 * [new branch]      master     -> source/master

fanwenl_@fanwenl_PC MINGW32 /e/prj/ilichee (master)
$ git checkout master
Already on 'master'
Your branch is up-to-date with 'origin/master'.

fanwenl_@fanwenl_PC MINGW32 /e/prj/ilichee (master)
$ git merge source/master
Merge made by the 'recursive' strategy.

fanwenl_@fanwenl_PC MINGW32 /e/prj/ilichee (master)
$ git push origin master
fatal: unable to access 'https://github.com/fanwenl/ilichee.git/': Failed to con
nect to github.com port 443: Timed out

fanwenl_@fanwenl_PC MINGW32 /e/prj/ilichee (master)
$ git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 758 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To https://github.com/fanwenl/ilichee.git
   7309dfe..ce5eef7  master -> master

fanwenl_@fanwenl_PC MINGW32 /e/prj/ilichee (master)
```

图例4.1 这样 fork 源仓库、fork 仓库副本 和 local 仓库实现了同步更新

5. 推送请求(pull request)

假设就按照上述流程，初始clone代码库到本地主机上，B就可以尽情修改code(branch、commit、merge、push)，每次 B push 更新，都只会更新自己的代码库b，并不会影响到到A的代码库a。 如果哪天B觉得自己新增加的内容很不錯，可以帮助到群主A，想跟A分享，那就可以发一个**pull request**，问问A要不要这一份。 下面我用两个帐号模拟fork和pull request：我用帐号B fork了A的一个代码库a，成为了自己的代码库b，修改提交之后，然后想把改动增加到库a里，于是点击“Create pull request”：

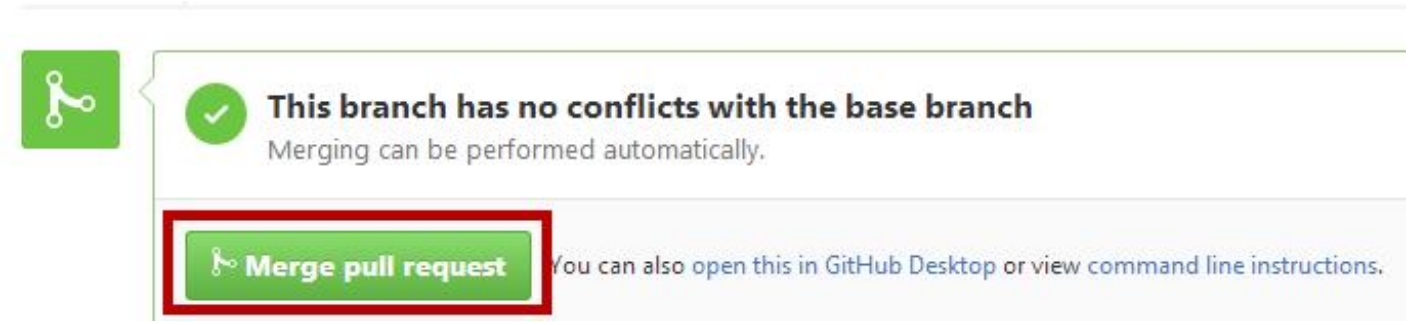


图例2.7

A帐号马上收到了邮件通知，当然github主页也有timeline消息通知。

图例3.8

A收到这则pull request之后，如果覺得ok，用线上merge，就會將代码库b合并到代码库a上。



图例3.9

四、Git工作流程简介

- 建立目录
- 加载文件(stage)
- 提交文件(commit)
- 创建分支(branch)
- 合并分支(merge)
- 丢弃分支
- 删除分支(delete)
- 回滚到之前的提交状态(back)
- 复制和推送请求(fork/pull request)
- 推送到远程代码库(push)
- 取得远程代码库的拷贝(pull)
- 更多……

首先，你需要去github上将别人的代码库复制到自己的库中，也可以先在自己的电脑上建立目录，相当于自己的个人站点，对应着github服务器上的站点。接下来，你可以在这个目录下进行创建新文件、修改文件等等操作，修改完之后，就需要加载文件，也就是把修改过的文件正式放进自己的项目里。而提交文件则是把文件上传到github服务器，进行文件同步。接着，我想加入新的代码进行测试，那么就可以创建分支。建立分支是你创建代码的独立版本的动作，独立于你的主干分支，相当于进入了另外一条“时间河流”。默认情况下，每次你提交到Git的文件都会被储存到主干分支。满意的话将新分支和主干合并分支，不满意的话“时光倒流”回到主干，并删除分支。如果发现新增的改动有问题，可以回滚到之前的提交状态。终于，本地的修改结束，想要上传到github，就可以推送到远程代码库；相反，如果想把github仓库里的文件下载到本地，可以取得远程代码库的拷贝。

五、具体操作

下面，我们以windows上的操作步骤为例进行示范（linux类似）。

1. 加载文件

我用vim指令创建并保存了text1.txt和markd.md两个文本文件，但它们还没有放入中。现在加载(stage)所有项目文件仓库(repository)，输入：

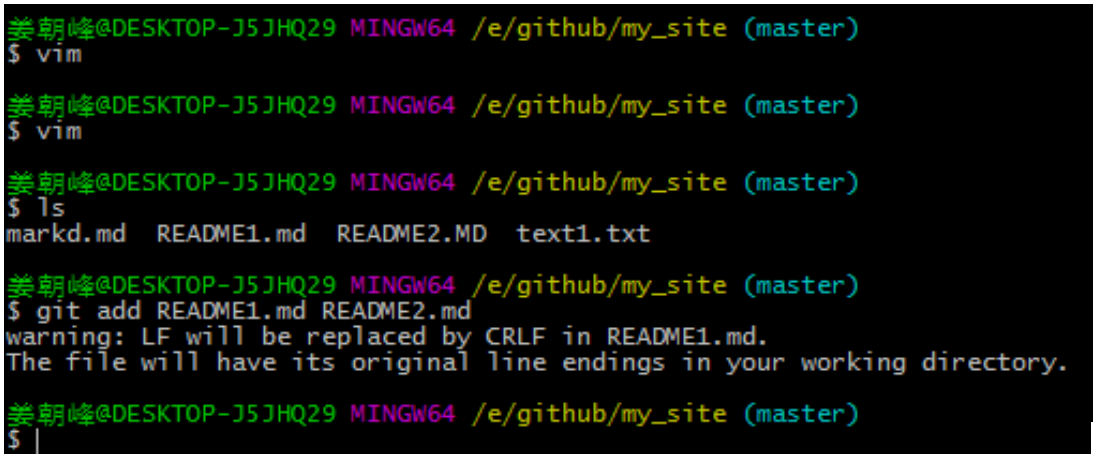
```
git add .
```

最后的“.”符号的意思是“所有文件、文件夹和子文件夹”。

图例2.1

假如我们只想要把特定文件添加到源代码控制中去，我们可以指定要添加的文件。比如，我用vim创建了两个markdown文件------README1.md、README2.md，发送：

```
git add README1.md, README2.md
```



图例2.2

2. 提交文件

现在，我们想要提交已加载（staged）的文件。提交文件时，我们需要给这个状态一个备注，

所以我们提交我们的文件时，总是附带着有意义的注释，描述了它们现在的状态。比如用“first commit”来作为第一个提交的注释，如下：

```
git commit -m "first commit"
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git commit -m "first commit"
[master (root-commit) 614e0c0] first commit
 3 files changed, 3 insertions(+)
 create mode 100644 README1.md
 create mode 100644 markd.md
 create mode 100644 text1.txt
```

图例2.3

但发现至提交了3个文件，检查可知我把其中一个文件README2.md打成了README2.MD，修改后重复上面的操作即可。 这样我们就用“first commit”代表这个时间点提交的修改，后面还可以再回滚到这个提交状态。 提交之后，如果你想查看现在已加载、未加载的文件及状态，可以用以下命令：

```
git status
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git status
On branch master
nothing to commit, working tree clean
```

图例2.4

3. 创建分支

如果要写或者测试新的部分，还不想直接加入到程序中时，就可以创建分支，就像暂时踏入另一条时间的河流一样。建立分支是你创建代码的独立版本，独立于你的主干分支。默认情况下，每次你提交到Git的文件都会被储存到“master（主干）”分支。 创建并同时切换到你新建的分支，发送：

```
git checkout -b new_feature
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git checkout -b new_feature
Switched to a new branch 'new_feature'

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (new_feature)
$
```

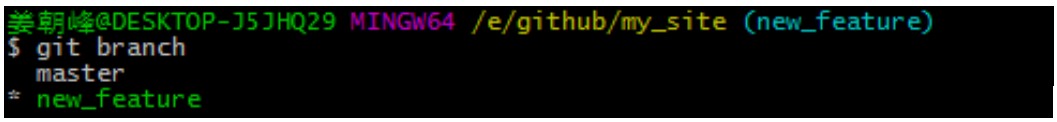
图例2.5

或者，你可以先创建一个分支然后手动切换，输入命令：

```
git branch new_featuregit checkout new_feature
```

要看你现在项目下所有的分支，输入如下指令：

```
git branch
```

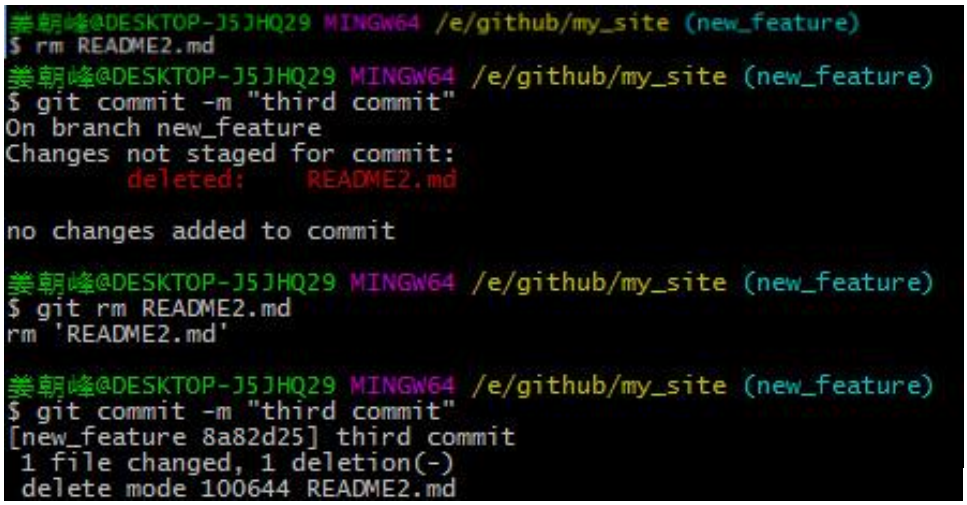


图例2.6

现在你可以在你的项目上无所顾忌地做任何你想做的：任何时候，你都可以回到你创建分支前的状态。你同时可以有多个分支，甚至可以从一个分支上再创建一个分支。

4. 合并分支

现在我在新的分支new_feature做一些修改，比如删除了README2.md。



图例2.7

在这个分支上修改得差不多的时候，如果想要把这个分支加回到主干(master)上，首先需要加载(stage)并且提交(commit)你的文件：

```
git add .git commit -m "third commit"
```

然后你移到主干分支：

```
git checkout master
```

最后像这样合并：

```
git merge new_feature
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (new_feature)
$ rm README2.md
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (new_feature)
$ git commit -m "third commit"
On branch new_feature
Changes not staged for commit:
        deleted:    README2.md

no changes added to commit

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (new_feature)
$ git rm README2.md
rm 'README2.md'

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (new_feature)
$ git commit -m "third commit"
[new_feature 8a82d25] third commit
 1 file changed, 1 deletion(-)
 delete mode 100644 README2.md
```

图例2.8

此时，主干分支就和新功能分支会变成一样的了。

5. 丢弃分支

相反，如果你打算丢弃你在分支里做的修改，你首先需要加载你的文件并且在分支里提交：

```
git add .git commit -m "feature to be discarded"
```

然后，你移到主干分支：

```
git checkout master
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (new_branch1)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin3/master'.

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$
```

图例2.9

现在，你的代码处于你创建分支之前的状态了。

6. 删除分支

如果你要把你的分支合并到主干分支，从主干（master）分支上发送：

```
git branch -d new_feature
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site ((8a82d25...))
$ git branch
* (HEAD detached from cc87a2d)
  master
  new_feature

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site ((8a82d25...))
$ git branch -d new_feature
Deleted branch new_feature (was 8a82d25).

姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site ((8a82d25...))
$ git branch
* (HEAD detached from cc87a2d)
  master
```

图例2.10

发现"new_feature"这个分支确实被删除了。 假如修改已经合并了，它只会删除分支。假如分支没有合并，你会得到一个错误信息。删除一个未合并的分支（通常你不想保留的修改），你需要发送一样的命令附带一个大写D。意思是“强制删除分支，无论如何我不想要它了”。

```
git branch -D new_feature
```

7. 回滚到之前的提交状态

在某些时候，你可能想要回到之前的代码版本。首先，你需要找到你想回到哪个版本。要看所有的完成了的提交，发送：

```
git log
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git log
commit 8a82d255ede7db1b3e975b22cda7e49e7fdc5773
Author: Phoenix1943 <298245139@qq.com>
Date:   Tue Aug 23 02:08:16 2016 +0800

    third commit

commit cc87a2d42a208203037d77e8239f56f7cb41b6d6
Author: Phoenix1943 <298245139@qq.com>
Date:   Tue Aug 23 01:26:51 2016 +0800

    second commit

commit 614e0c06c29bf3dc24768033c1b6b7b09f081b30
Author: Phoenix1943 <298245139@qq.com>
Date:   Mon Aug 22 23:39:01 2016 +0800

    first commit
```

图例2.11

这会输出你的提交的历史记录。

如果你想回到“second commit”这个提交，简单地用提交的ID做签出(checkout)（可以只用到ID开头的9个字符）

```
git checkout cc87a2d42
```

```
姜朝峰@DESKTOP-J5JHQ29 MINGW64 /e/github/my_site (master)
$ git checkout cc87a2d42
Note: checking out 'cc87a2d42'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at cc87a2d... second commit
```

图例2.12

你也可以签出到一个新的分支，像这样：

```
git checkout -b my_previous_version cc87a2d42
```

但是，要注意保持分支的清晰，太多的分支会导致整个仓库的混乱，让整个项目失去控制。

六、更多

- 如果对命令行的git使用方式很不适应，还可以下载git的**GUI**版本，比如GitHub Desktop和Gitbox(见网站https://git-scm.com/downloads/guis)，GUI版本的git确实要比命令行更直观更容易入门。
- 当然，还有比这些更多的**Git**的相关知识，本文也是我参考网上的资料进行整理、修改和实践得到的。如果有疑惑或者需要更进一步了解的地方，可以自己搜索“**Git**”相关资料或者和我联系，推荐廖雪峰的官方网站/Git教程，相信你会有更多收获。