Git常用命令教程

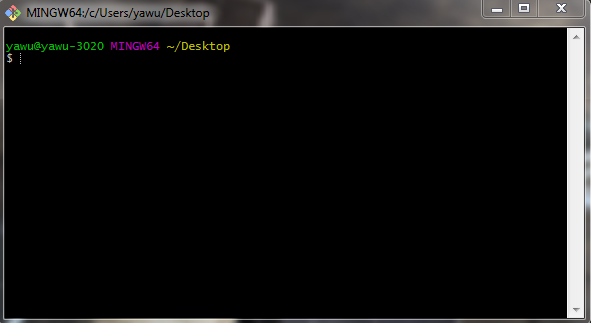
# Git简介

Git是目前最流行的版本控制系统。什么是版本控制系统？比如你写好了一个文档，取名为“文档01”，之后你又增添了一些内容，取名为“文档02”，后来你发现自己改的乱七八糟，想回头重新改，这时这个版本控制系统就可以直接回退到“文档01”，而不是自己手动的删除新增的内容。

# Windows 下Git安装

下载地址：<http://msysgit.github.io/> ，按照默认选项安装即可。

安装完成后鼠标右击，选择Git Bash Here，出现以下窗口则安装成功。



安装完成后在命令行输入：

$ git config --global user.name "Name"

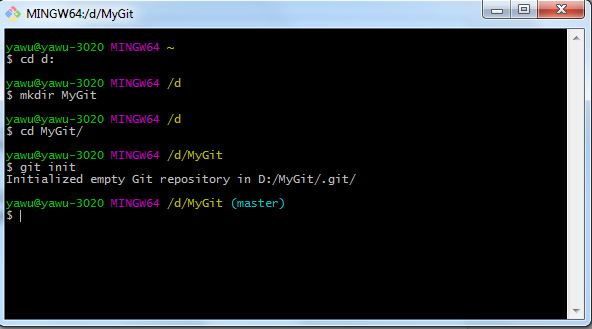
$ git config --global user.email "email@example.com"

这样，当你提交代码时其他人就能知道这是谁提交的了。

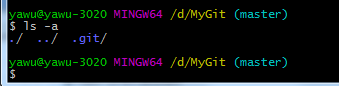
# 初始化一个版本库

首先创建一个文件夹MyGit，进入后右击，选择Git Bash Here，输入git init。

或者直接打开Git Bash，采用命令行的方式。



上图中前三步为创建文件夹并进入，git init 为初始化一个git 仓库。



Ls –a用于查看文件，可以看到，MyGit下多了一个git隐藏目录。这个目录用于跟踪文件变化，不要修改此目录，否则很可能会破坏这个仓库。

# 工作区和版本库

什么是工作区？就是你工作的地方，例如你是在MyGit目录下工作的，MyGit就是一个工作区。那版本库呢？之前有提到的.git隐藏目录，这个就是一个版本库，版本库下还有一个暂存区和当前分支，暂存区和分支待会儿就会提到。

# 提交一个版本

当你在工作区完成工作后，现在你要把它提交。假设你的工作是创建一个名为README.txt文件，现在你已经创建好了，那么如何提交呢？

首先你要在你的工作区打开Git Bash，分两步输入：

$ git add README.txt

$ git commit -m 'add README.txt'

第一步将工作区的README.txt文件加入到版本库的暂存区，第二步则是将暂存区的内容提交到当前分支，-m 后的内容是对本次提交的说明。你也可以“add”多次文件，然后一次性提交。当你需要add工作区所有文件时，可以用 git add . 来简化操作。

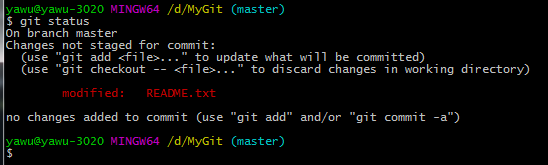
# 版本控制

## Git status

Git status是一个常用的功能，可以让我们了解git当前的状态，现在在README.txt下添加以下内容:

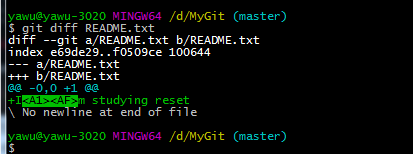
I’m studying reset

使用git status 可以查看git的状态：



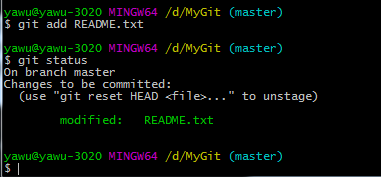
上面提示说README.txt已经被修改，但是没有被提交，可以使用git add 来上传到暂存区，也可以使用git checkout -- <file>来放弃此次修改。

上图只说了README.txt被修改，但并没有说明修改了那些内容，这时我们可以使用git diff 命令。



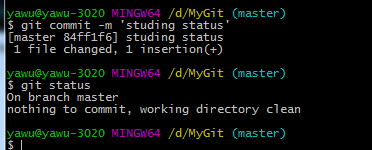
从上图可以看出，添加了一行内容。

现在将修改add 到暂存区。再使用git status查看状态：



可以看到提示，该改变准备被提交，modified那一行变成了绿色，表示README.txt已经被添加到了暂存区。

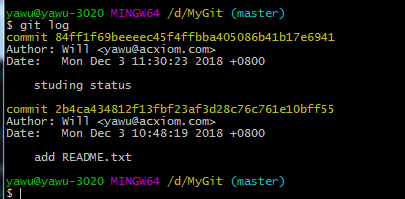
提交并查看状态：



提交后可以看到工作区是干净的没有需要提交的文件。

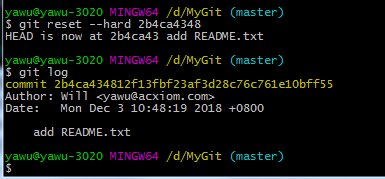
## 版本回退

到现在为止我们有几个版本呢？可以通过git log 命令查看。



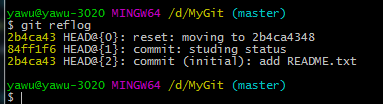
看到了吗，commit后面的是每次提交的版本号，Author后是在一开始就设置的用户名和邮箱，Date后是提交的时间，最后一行是对提交的版本说明，也就是git commit -m 后的内容。

现在我想回到上一个版本，也就是add README.txt的时候。可以用git reset --hard “版本号”,版本号可以只输前几位，只要这前几位已经可以唯一定位一个版本。例如我可以输入git reset --hard 2b4ca434 来回退到上一个版本：

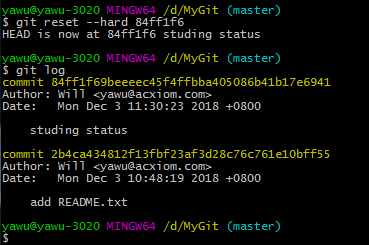


可以看到只剩一个版本了，现在我已经穿越到了过去，那么我想再回到未来呢，假使你关闭了该命令行窗口，找不到未来的版本号了，那怎么办呢？

别着急，你可以使用git reflog来查看你对版本的操作：



两次提交，一次回退，前面还记录着当时操作所在的版本号，终于可以回到未来了。

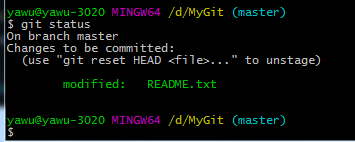


## 撤销修改

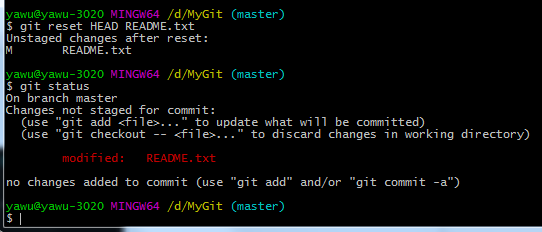
现在在README.txt中加入一段话：

“sun is black”

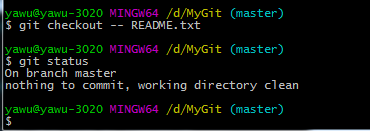
好了，然后你把它add到了暂存区，随后你发现了错误想要撤销怎么办呢？git已经给了我们提示，使用git status查看：



git提示我们可以用git reset HEAD <file> 来撤销



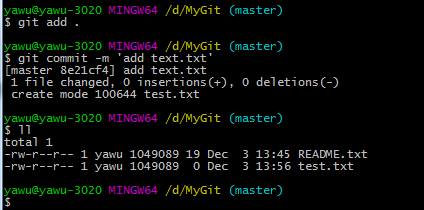
可以看到，暂存区已经没有了内容，只有工作区还保留着修改，此时使用git checkout -- <file>可以放弃工作区的修改，让其和版本库内容一致。



如果已经提交了呢？可以使用git reset –hard “版本号”，以上操作除了回退版本外都不需要背，直接git status都会有提示，方便！

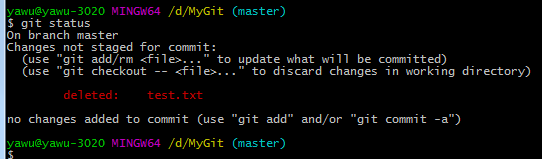
## 删除文件

Git 中删除文件和添加文件一样，都需要告知git ，我们先创建一个文件test.txt,并提交：

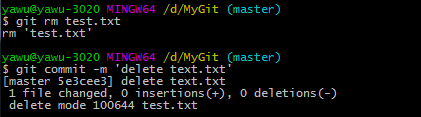


ll用于显示当前目录下的文件，可以看到已经有了test.txt

现在我删掉test.txt



可以看到，工作区删除文件后，还要用git rm <file> 命令上传到暂存区，最后再提交就完成了。



# 远程仓库

Git 是一个分布式的版本控制系统，自然不能只在一台机器上操作，还需要能够上传与下载。

能够通过设置一台服务器，大家都从这台机器上下载代码，完成各自的模块后还能够提交上远程服务器，其他人也能够看到你的修改。

这里我们用GitHub网站做测试，将它作为一个远程服务器。

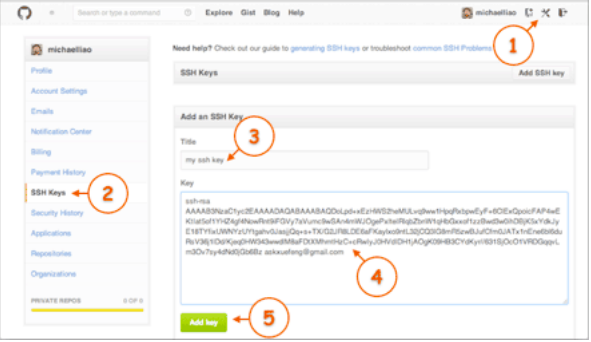
本地git 仓库与Github之间的传输是通过ssh加密的，需要先创建ssh key，检查一下自己的用户主目录下有没有id\_rsa和id\_rsa.pub两文件，如果有则跳到第二步。

1. 在git bash下输入：ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

然后一路回车即可。

1. 登陆GitHub，打开“Account settings”，“SSH Keys”⻚⾯： 然后，点“Add SSH Key”，填

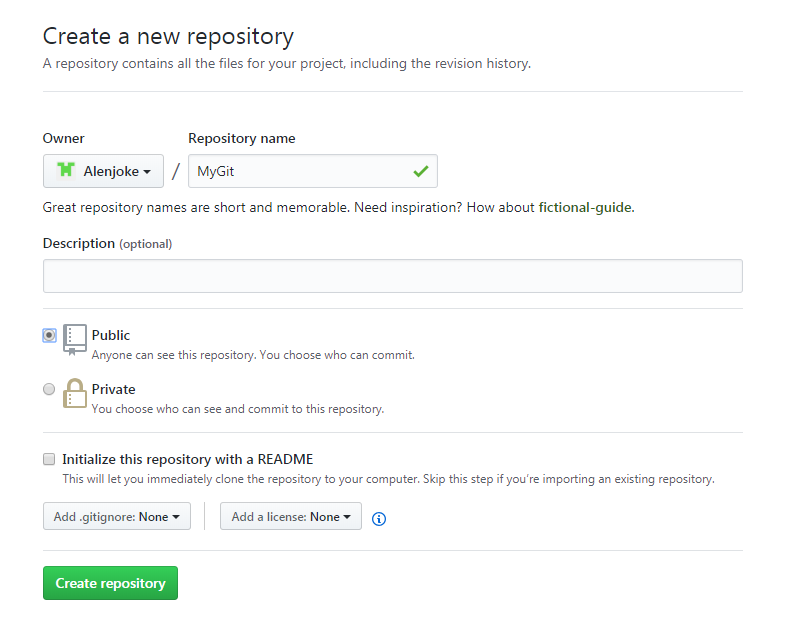
任意Title，在Key⽂本框⾥粘贴id\_rsa.pub⽂件的内容：



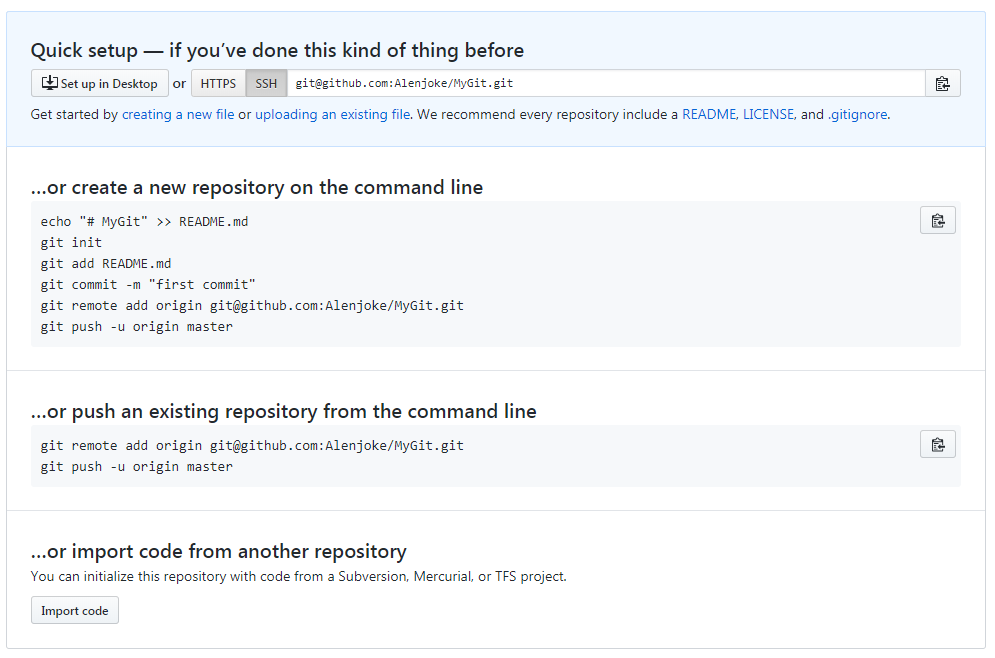
这样你就可以向github中提交了。

## 创建一个远程仓库

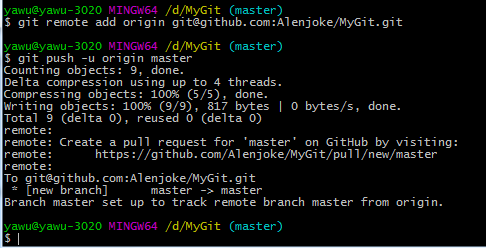
如下：



点击Create repository



它提示我们可以自己创建一个仓库，或者从本地推送一个，这里我们选择从本地推送。



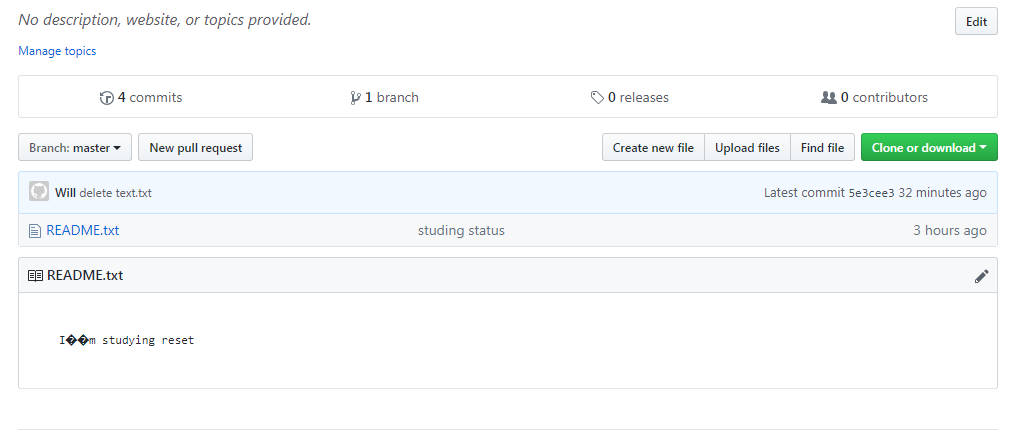
好了上传上去了，这里SSH地址换成你自己的。

$ git remote add origin git@github.com:Alenjoke/MyGit.git

用于与远程库关联。

$ git push -u origin master

用于将本地master 推送至远程的master，由于远程仓库是空的，第一次推送需要加-u指令，以后就不需要了。

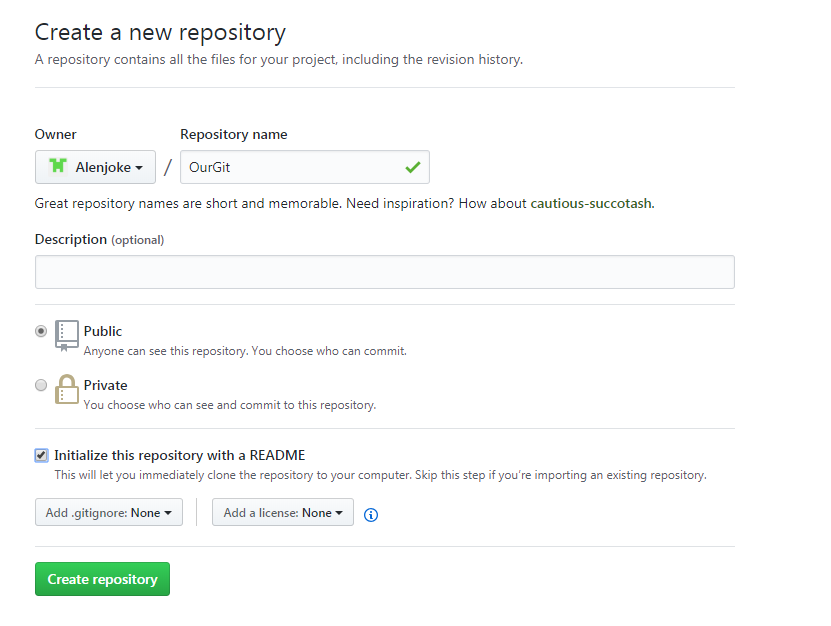


可以看到已经成功推送到远程了，一个README.txt，共有4次提交。

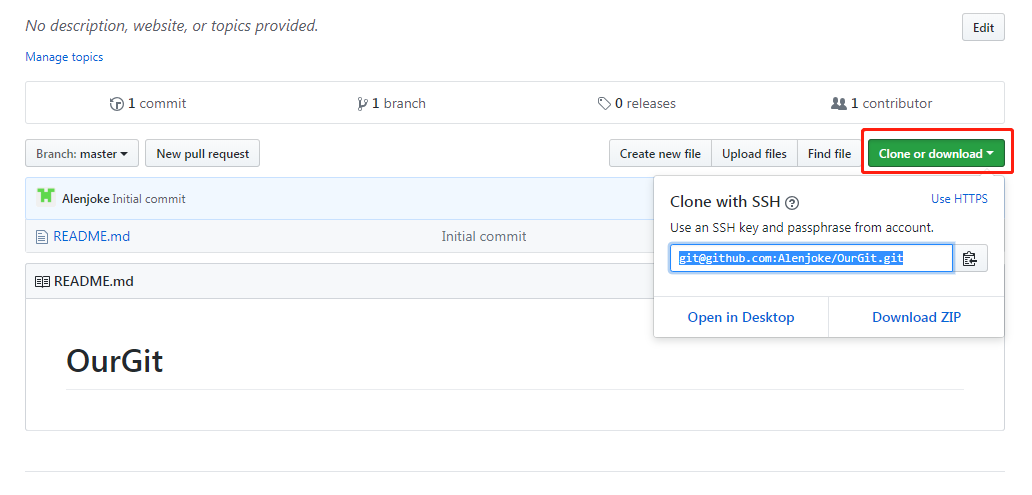
## 克隆

通常开发都是从远程创建仓库，然后本地克隆下来开发的，那这要怎么做呢？

首先创建一个远程仓库：



勾选 Initialize this repository，点击创建。



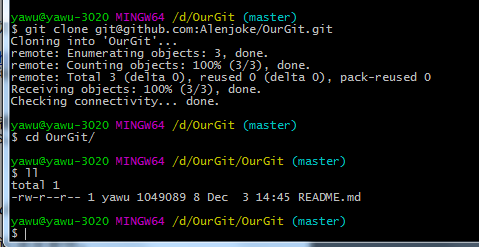
远程库创建好了，点击Clone or download，将下面的ssh地址复制。

在本地新建一个git 仓库并打开git bash.

首先关联远程库：



再输入 git clone git@github.com:Alenjoke/OurGit.git 即可。



# 分支

## 本地分支

分支有什么用，不知道你是否注意到了，git bash中有个蓝色的(master)，这代表我们现在在master分支上工作，这是一个默认的分支。假设我们只有这个分支，多人合作时，我开发了一个功能，却只完成了一半，这时如果上传到远程，别人拉下代码时，代码会无法运行，而如果我不提交，可能会面临代码丢失的风险。

这时分支就起作用了，你可以将代码提交到你自己的分支。

若你在master分支上工作，master 指向当前提交，而HEAD指向master。

创建分支：

Git branch dev

切换分支:

Git checkout dev

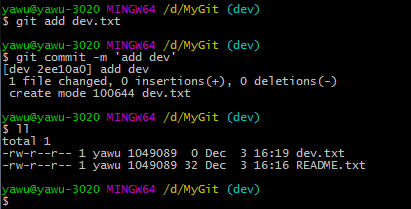
查看分支：

Git branch



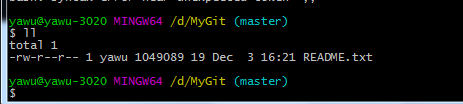
现在修改dev 中的README.txt，新建dev.txt

提交。

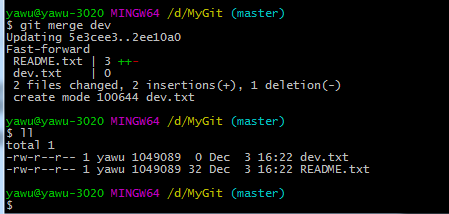


多了一个文件dev.txt。

现在切换到master



没有dev.txt，但是可以将master与dev 合并。



这样就和dev分支一样了。

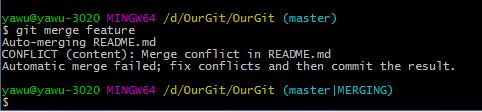
删除分支：

Git branch –d “分支名”

### 本地分支冲突

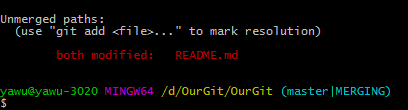
新建一个分支feature，修改README.txt，加入”feature”并提交，之后切换到master分支。

在README.txt中加入”master”并提交。这时将master与feature合并。出现以下信息：

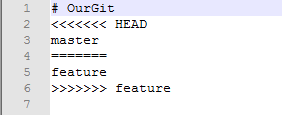


出现冲突了。

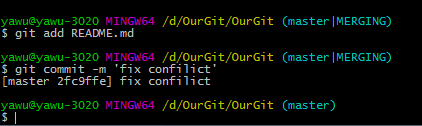
这时输入git status即可查看冲突。



冲突文件为README.md，打开它



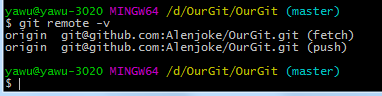
Git 使用<<<< HEAD的内容======feature的内容>>>>>来标记冲突，你可以对冲突做修改，然后add、commit



这样冲突就解决了。

## 远程分支

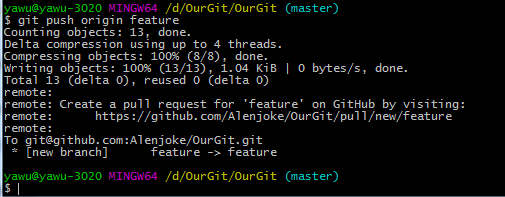
查看远程库信息:



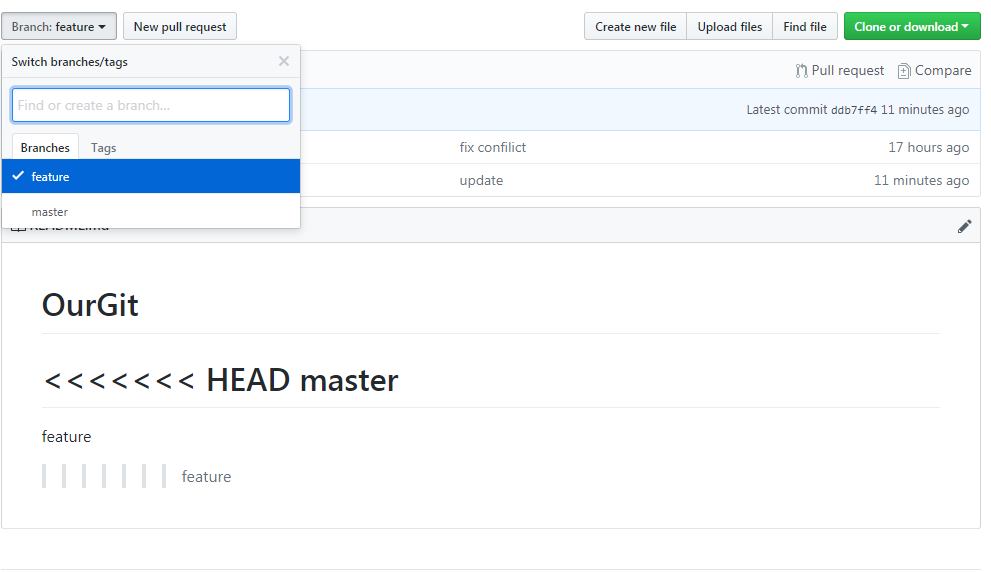
显示了推送和抓取的地址,若无推送权限,则push那一行看不到

### 推送分支

Git push origin “你的本地分支”



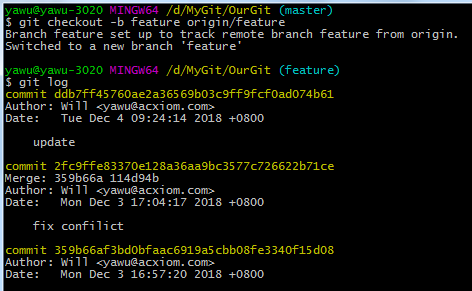
这里我直接向远程推送了feature分支，这样远程会自动创建一个feature分支与我本地的对应。



那当其他人克隆这个项目后，如何在feature分支上工作呢？



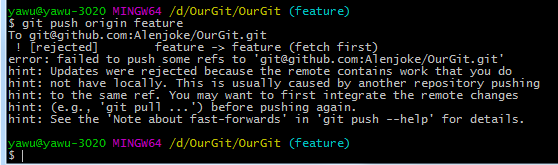
重新克隆项目后，本地只有master分支



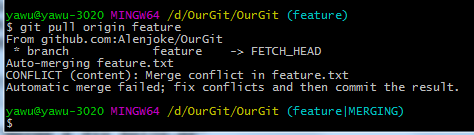
使用了$ git checkout -b feature origin/feature 后，可以使本地的feature与远程的feature对应起来。我们查看log发现，本地的feature分支已经和远程的一样了。

### 远程分支冲突

现在其他人本地有了feature分支，并且做了一些修改后提交到远程。此时，正好你在feature上的工作也完成了，然后提交到远程，这时就会有冲突。

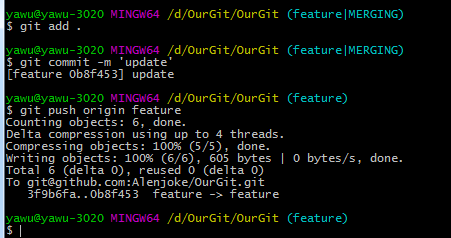


Git 提示我们使用git pull来解决冲突。



Git pull到本地会有冲突，我们自己在本地解决掉冲突，提交，push。

Git pull = GIt fetch + Git merge



可以看到，提交成功了