[第一篇、git安装 2](#_Toc332367698)

[1、安装 2](#_Toc332367699)

[2、注册 2](#_Toc332367700)

[3、保存key 2](#_Toc332367701)

[4、生成key 2](#_Toc332367702)

[5、使用key 3](#_Toc332367703)

[第二篇、使用git 5](#_Toc332367704)

[1、如何使用其他资源 5](#_Toc332367705)

[2、如何向一个空仓库中上传代码 6](#_Toc332367706)

[3、如何使用上游分支进行开发 8](#_Toc332367707)

[4、如何删除github.com上的一个分支 9](#_Toc332367708)

[5、如何多用户使用 9](#_Toc332367709)

[6、常用命令 12](#_Toc332367710)

[7、常用方法 13](#_Toc332367711)

[8、常用网址 13](#_Toc332367712)

# 第一篇、git安装

## 1、安装

$sudo apt-get install git-core git-gui git-doc //ubuntu安装

$yum -y install git //fedora安装

## 2、注册

到<https://github.com/>注册一个帐号（假定changyukun），以后客户端登录的时候要使用的。

## 3、保存key

在客户端安装SSH Key，方便我们在客户端安全登录：

cd ~/.ssh

如果提示“没有那个文件或目录”，那么说明以前没有生成过SSH Key，直接跳到第4步，否则保存从前的SSH Key：

ls

mkdir key\_backup

cp id\_rsa\* key\_backup

rm id\_rsa

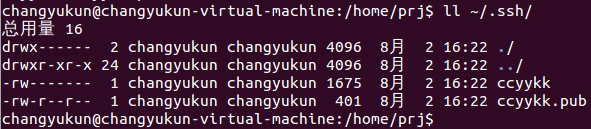
## 4、生成key

ssh-keygen -C "changyukun@hualu.tv" –f ~/.ssh/ccyykk

注意：上面要填你注册的邮箱；

然后会询问保存key的位置，我们选择默认，一直enter就可以了

命令完成后会在 ~/.ssh/ 目录下生成两个文件 ccyykk、ccyykk.pub



## 5、使用key

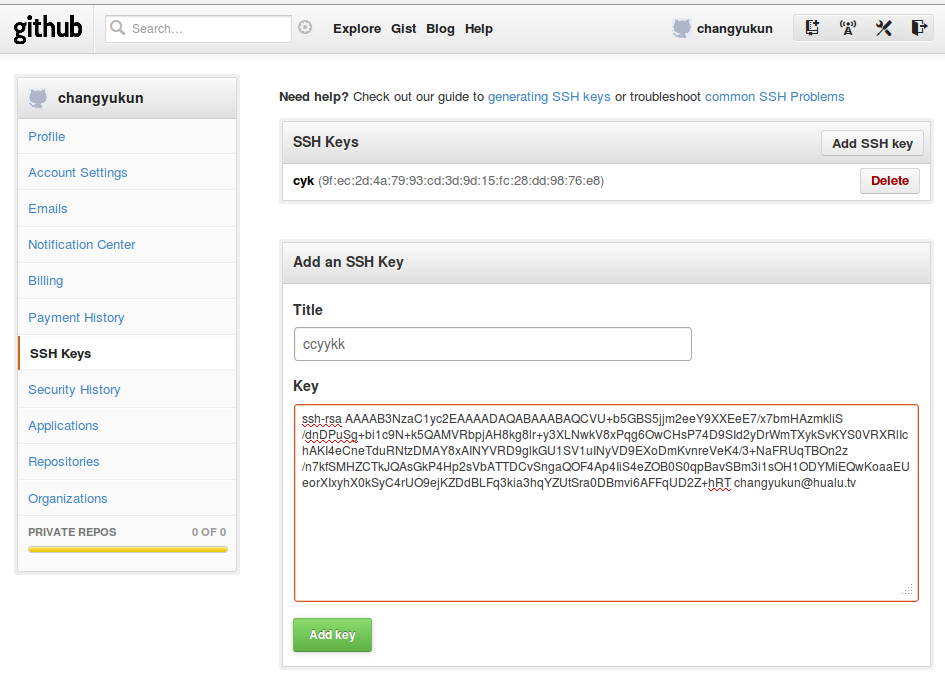
也可以用ssh-keygen命令以不同的名称创建多个公钥，当拥有多个Github账号时，非常重要。这是因为虽然一个GitHub账号允许使用多个不同的SSH公钥，但反过来，一个SSH公钥只能对应一个GitHub账号。

接下来就将~/.ssh/ccyykk.pub文件内容拷贝到剪切板。公钥是一行长长的字符串，若用编辑器打开公钥文件会折行显示，注意在copy时一定不要在其中插入多余的换行符、空格等，否则在公钥认证过程因为服务器端和客户端公钥不匹配而导致认证失败。然后将公钥文件中的内容粘贴到GitHub的SSH公钥管理的对话框，即上图key对话框中，并为这个SSH Key起个名字并保存。设置成功后，再用ssh命令访问GitHub，会显示一条认证成功信息并退出。在认证成功的信息中还会显示该公钥对应的用户名。

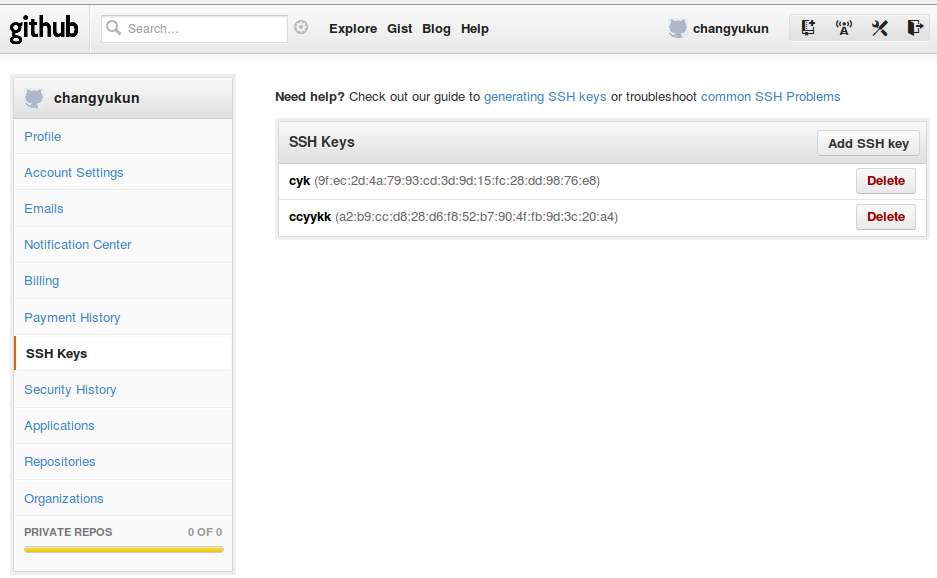
ssh -T [git@github.com](mailto:git@github.com)

执行后提示：Hi github! You've successfully authenticated, but GitHub does not provide shell

通过以上的设置之后，我们就能够通过SSH的方式，直接使用Git命令访问GitHub托管服务器了



添加成功的图片（增加了一个ccyykk，网站会发一封邮件到邮箱changyukun@hualu.tv）



# 第二篇、使用git

使用github进行项目开发（账户：changyukun）

## 1、如何使用其他资源

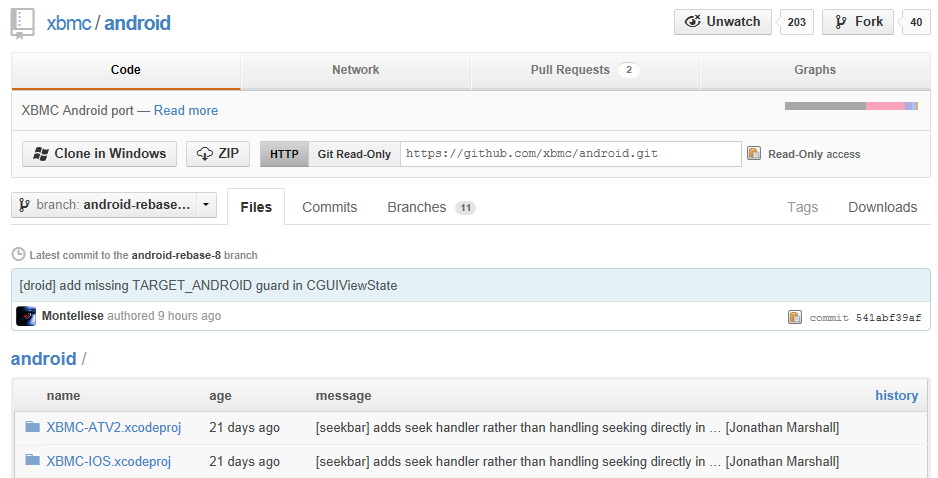
A、首先针对我们要做的项目，在github上查找相关系的资源，如我们想做一个xbmc的android版本的项目，然后我们就在github的搜索中进行相应的搜索



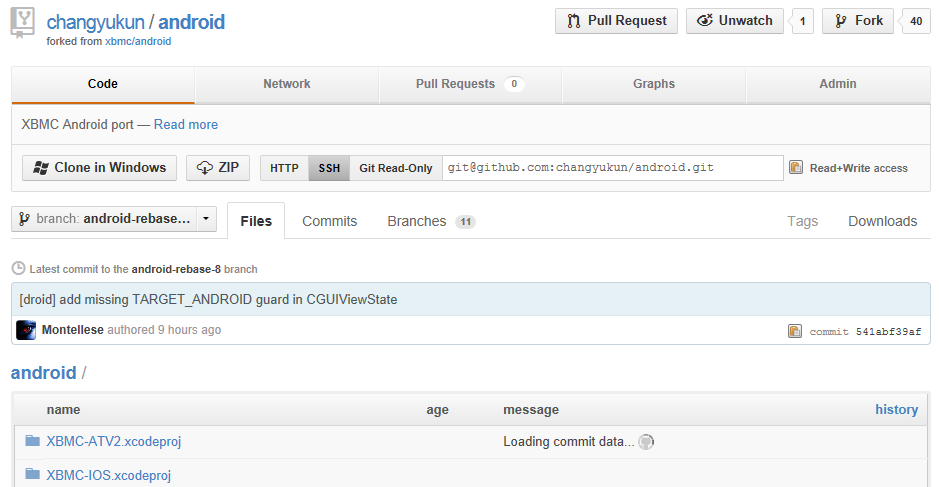
然后就能搜索到如下的一个项目



1. 然后点击进入这个xbmc/android的项目中，大概如下图：

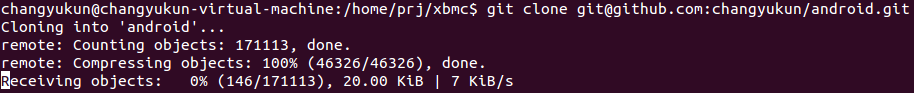


1. 然后点击fork一个到自己的账户中（当然点击fork前可以选在一个branch分支）



D、下载到本地

mkdir xbmc

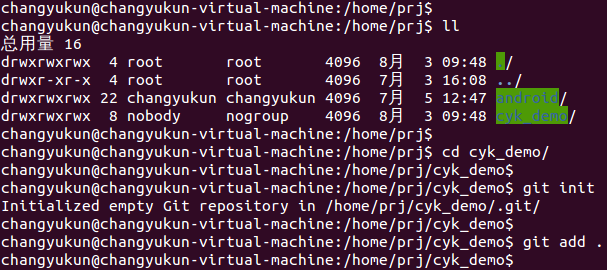
克隆到本地（利用上步骤中的github网站提供的ssh连接地址下载）

## 2、如何向一个空仓库中上传代码

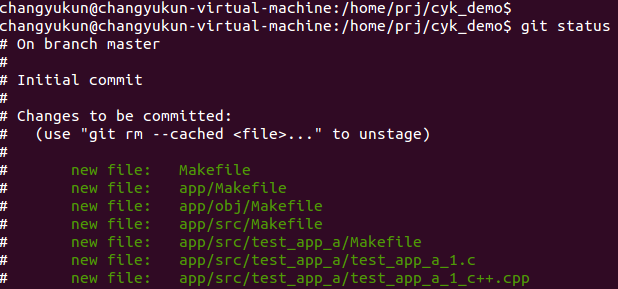


上面命令需要注意：如果创建的仓库与工程文件名字不相同，可能会在执行git push的时候出现错误，通常可使用强制上传命令 git push -f origin master

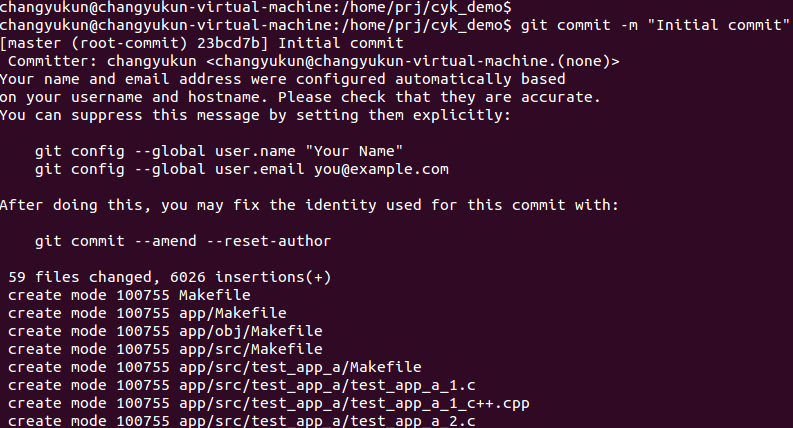
下面为一个实际的例子，将本地的cyk\_demo目录下的所有代码上传到cyk-source仓库



可以查看一下状态（本地的）

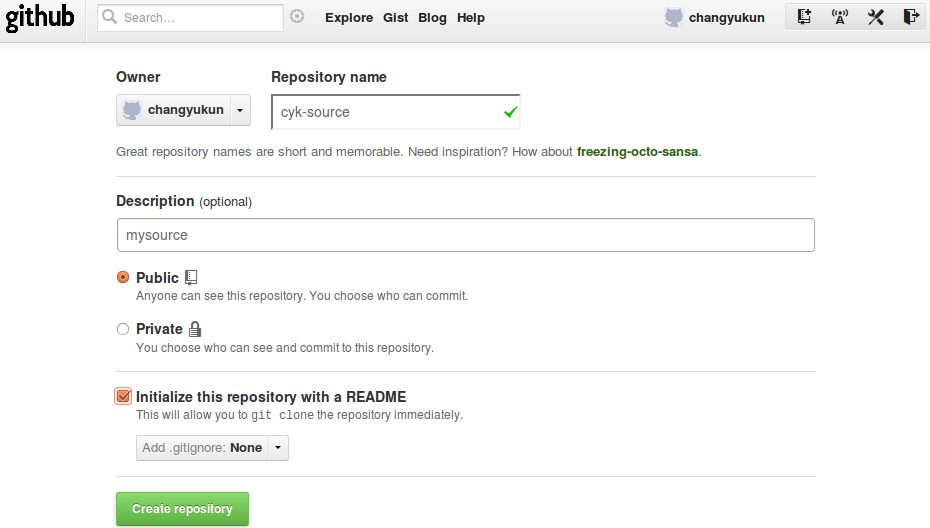


继续执行命令

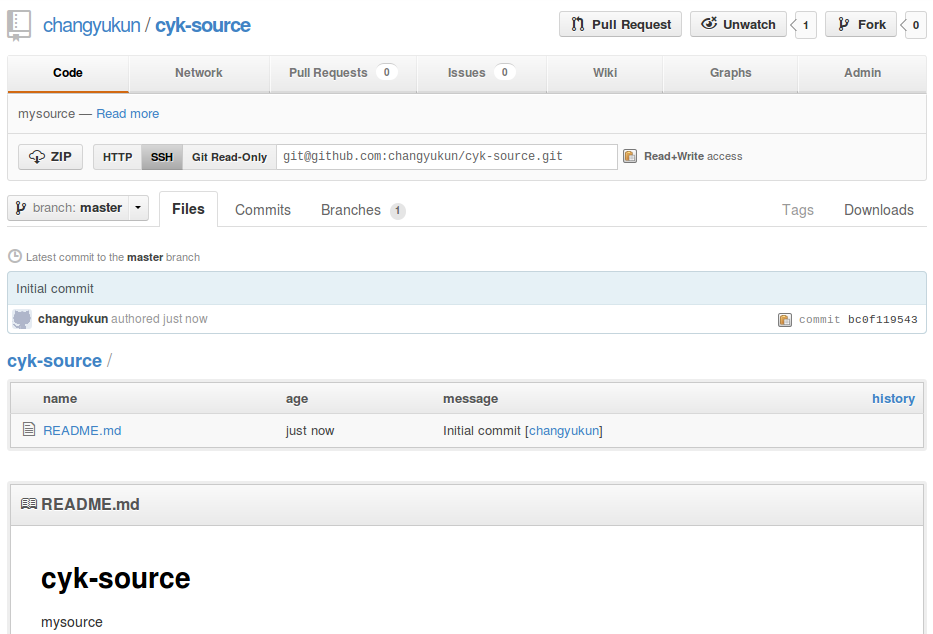


执行此上述步骤之后、或者一开始就需要登录github网站创建一个空的仓库，名字为cyk-source(注意此仓库的创建下面的勾选上了[如下图所示]，与后面的没选上勾创建cyk-source-1仓库时所使用的上传命令略有不同，没打勾的上传命令见创建仓库cyk-source-1成功的提示)，如下：

**创建仓库cyk-source**

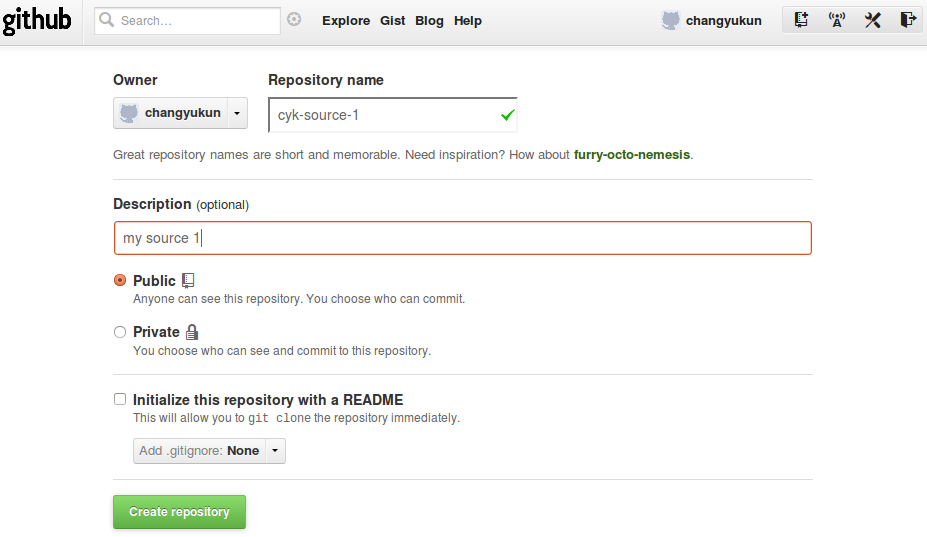


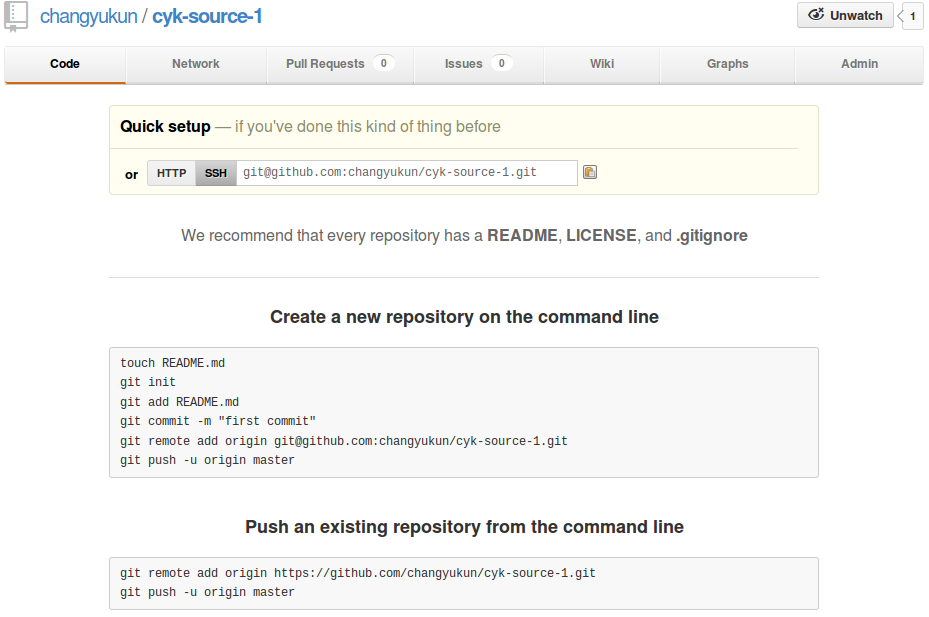
点击“Create repository”，创建仓库成功后形如下图所示：



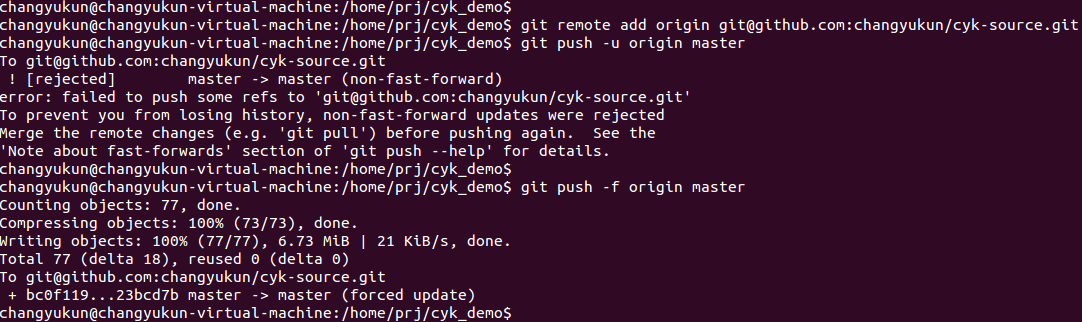
注意：创建成功后cyk-source仓库中只有master分支

**创建仓库cyk-source-1**





继续执行命令将代码传到cyk-source仓库中



至此，已经成功将代码上传至github的changyukun/cyk-source仓库中（master分支）

## 3、如何使用上游分支进行开发

本地的代码是从changyukun/android上克隆下来的，而changyukun/android又是在xbmc/android上fork出来的，则xbmc/android项目中的所有分支就称为上游分支。因为可能很多人更新了xbmc/android，但我们不知道，而我们又想使用xbmc/android的最新代码，即我们想用xbmc/android上的来对本地的进行更新，则可以用如下的命令。

//为远程地址地址为xbmc/android项目添加一个别名xxx，供本地使用（书写简单）

$git remote add xxx [https://github.com/xbmc/android.git](https://github.com/xbmc/android.git%20)

$git pull xxx android-rebase-9 //android-rebase-9 为xbmc/android的一个分支，即用xbmc/android的分支android-rebase-9来实现对本地的更新

经过上述命令更新完代码后就可以对代码进行修改，修改完成后可执行如下的命令

$git ci //制作生成一个版本

$git push //将修改传回给自己的 changyukun/android的分支中（即传到github服务器中）

## 4、如何删除github.com上的一个分支

思路：

新建一个空的版本库，直接把空的版本库push到github.com的某个分支上。这个分支就会被自动删除。

代码：

#这是我想删除分支的仓库地址

remote\_repo="https://github.com/changyukun/cyk-source.git"

#这是我想删除的远端仓库分支的名字

remote\_bra[nc](http://www.linuxso.com/command/nc.html)h="libapi"

1. [mkdir](http://www.linuxso.com/command/mkdir.html) /tmp/git-empty
2. [cd](http://www.linuxso.com/command/cd.html) /tmp/git-empty
3. git init
4. git push $remote\_repo :$remote\_branch #有个冒号不能省略

## 5、如何多用户使用

github使用SSH与客户端连接。如果是单用户（first），生成密钥对后，将公钥保存至github，每次连接时SSH客户端发送本地私钥（默认~/.ssh/id\_rsa）到服务端验证。单用户情况下，连接的服务器上保存的公钥和发送的私钥自然是配对的。但是如果是多用户（first，second），我们在连接到second的帐号时，second保存的是自己的公钥，但是SSH客户端依然发送默认私钥，即first的私钥，那么这个验证自然无法通过。不过，要实现多帐号下的SSH key切换在客户端做一些配置即可。

多用户定义：

1. 终端在不同电脑上使用同一个key与github的一个用户进行连接，即不用每换一台电脑（linux虚拟机）就要重新生成一个新的key，还需要将新生成的公钥key添加到github上用户的ssh中那么麻烦，而只是拷贝已有的key到不同的电脑上来实现的，具体如下：

假定：github账户为changyukun，一个仓库为cyk-source

1. 在a电脑上

ssh-keygen -C "changyukun@hualu.tv" –f ~/.ssh/ccyykk

通过上述命令后会在~/.ssh/目录下生成ccyykk、ccyykk.pub两个文件，然后将ccyykk.pub文件的内容添加到github上账户changyukun的ssh中，即设置公钥。

然后就可以在a电脑上正常使用git了

如：

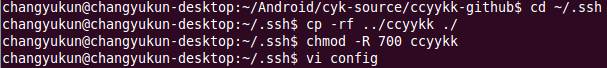
cd /xxx/

git clone [git@**github.com**:changyukun/cyk-source.git](mailto:git@github.com:changyukun/cyk-source.git)

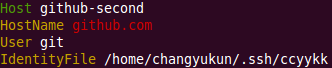
1. 在b电脑上

在b电脑上并不执行ssh-keygen命令重新生成key、设置公钥等操作，而是直接使用A步骤中即a电脑上创建的ccyykk私钥文件，具体操作如下：

mkdir -p ~/.ssh



在config文件中添加如下内容：



则在b电脑上就可以使用github-second（config文件中的Host值）来访问github上的changyukun用户了，即实质是使用a电脑上创建的公钥、私钥来配对的

如：

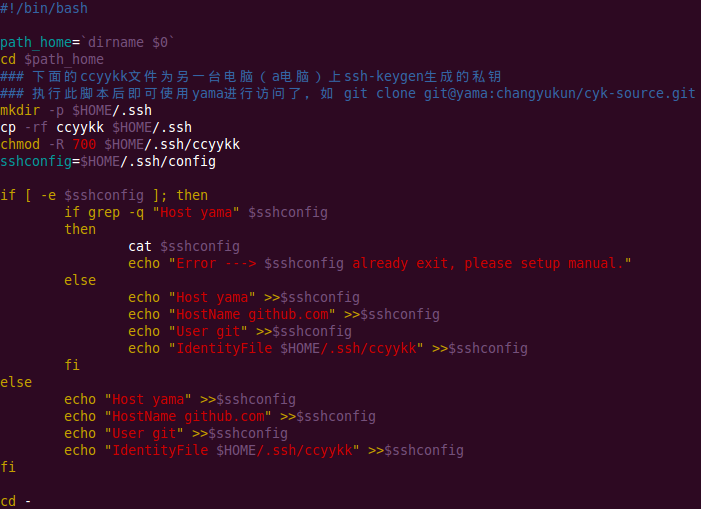
cd /xxx/

git clone [git@**github-second**:changyukun/cyk-source.git](mailto:git@github-second:changyukun/cyk-source.git)

附加说明1（b机如何使用默认）



附加说明2（b机上使用脚本，自动添加配置，手动麻烦，脚本大致如下）

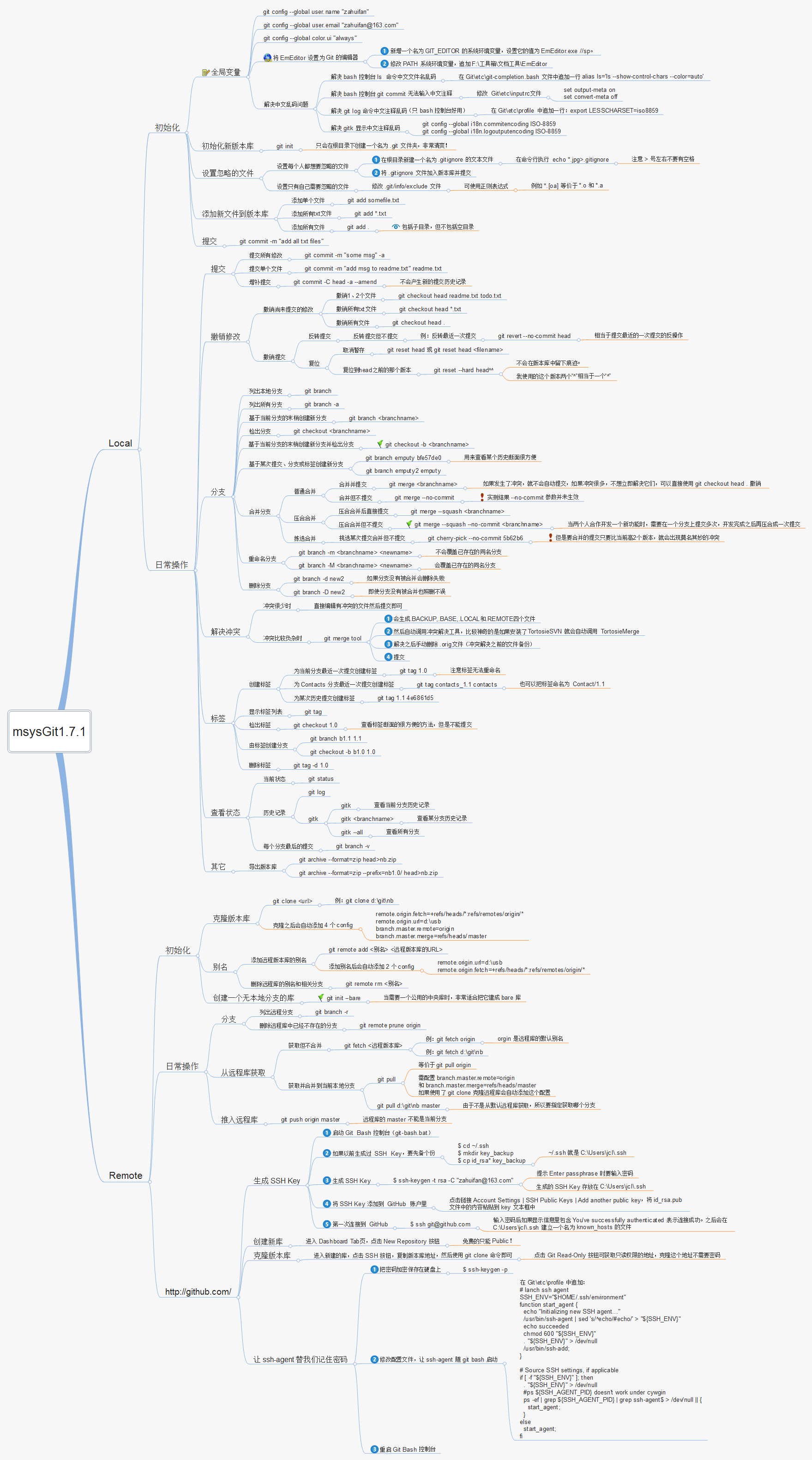


1. 在同一台电脑上访问github上面的多个用户

多个用户访问于1中的不同电脑相互类似。

## 6、常用命令

如下图：



## 7、常用方法

本地库初始化一般有如下几种方法，针对不同的初始化方法，其操作的命令也略有不同

**第1种方式：git init**

创建一个空文件夹，然后进入此文件夹，执行git init，即会在此目录下生成.git目录，即生成本地的仓库。此方法只是初始化了一个空的仓库，并没有与任何远程仓库进行上连接，因此要想获取远程仓库上的数据，或者是想将更新的数据推送到远程仓库上等等操作，都需要先和远程的一个仓库先进行连接，命令如下:

git remote add xxx [git@github.com:changyukun/cyk-source.git](mailto:git@github.com:changyukun/cyk-source.git)

此命令为远程的cyk-source仓库添加一个本地的别名，供后面使用方便，即后续命令中使用此别名就相当于使用此url，即此命令相当于实现了与远程仓库的连接。

**注意：**添加一次远程别名xxx之后，下次开机仍然可以继续使用，因为添加的内容是保存到本地仓库中了的，即应该是在.git文件夹中了。除非执行了git remote rm xxx之后xxx别名才会失效的。

下载数据：

git pull xxx branch-1 //将xxx的分支branch-1现在到本地仓库中

上传数据：

假设git checkout branch-local, 即本地选择的branch-local分支

git branch //查看本地分支，带\*号的表示当前选中的分支

git branch –a //查看所有分支，包括远程仓库与本地仓库对应的分支都会显示

1. 只是上传所有修改的文件

git commit –m “message1” -a //将所有修改上传

git push xxx branch-local //将本地的将本地的branch-local分支上传至远程

1. 添加了新文件

git add file1

git commit –m “message2”

git push xxx branch-local //将本地的branch-local分支上传至远程

1. 删除文件

**第2种方式：git clone**

本地建一个空的文件夹，然后直接使用clone命令直接克隆一个远程的库到本地，如：git clone [git@github.com:changyukun/cyk-source.git](mailto:git@github.com:changyukun/cyk-source.git)

执行完此命令后就会将远程的cyk-source的仓库完全克隆一份到本地的目录中，即.git目录页clone过来了（有本地的仓库了）

## 8、常用网址

<http://help.github.com/win-set-up-git/>

<http://help.github.com/multiple-ssh-keys/>

<http://my.oschina.net/u/559701/blog/63442>

<http://blog.csdn.net/cazicaquw/article/details/7441408>

<http://www.cnblogs.com/fnng/archive/2012/01/07/2315685.html>

GitHub使用简介

1. 关于GitHub的常见使用方法，请参考官网上面的说明。本篇文档只是针对于常用的，而且容易犯错的用法进行说明。仅仅是个人观点。

2. 如何使用GitHub？

(1) 常用的是首先在GitHub上创建一个版本库，然后接下来的工作是在本机创建一个版本库，注意不一定要和原来那个版本的文件名相同。比如原来创建的版本为 HelloWorld，那么本机完全可以使用一个文件名为HelloRyan这样的文件夹。只是需要在后续操作中配置相应的信息即可。

(2) 本机创建文件夹后，需要进行Git的配置。首先git init来初始化一个版本库，然后需要建立和远程版本库的链接。这里使用git remote add命令来添加。

git remote add origin url

其中origin是你指定的url的别名，你完全可以取其他的名称。而url则是版本库信息对应的路径，这里在GitHub上面你创建的版本库上面，有一个SSH/HTTP/Git Read-Only这些选项，你需要的做的就是选择一个url，填写即可。比如

git remote add helloworld git@github.com:username/repname.git

(3) 建立起本机和远程的联系之后，还可以对Git进行简单配置，比如邮箱，用户名，比较工具啥的，由于本文不是初级教程，所以此部分内容略。详见官方文档。

(4) 本机进行工程开发，比如新建文件，然后提交。等等

(5) 保证和远程同步，因此需要将本机的版本库push到远程上去，为此使用：

git push origin branch

其中，origin就是刚刚取的别名，比如是helloworld，而后面的就是要推送到的分支名，比如master。具体就看你本机和远程的分支，对应的别名了。

(6) 完成了上述修改后，可能会在另外一个场合拉取代码，此时还是相同的步骤，创建一个文件夹，git init来初始化版本库，然后同样git remote add xxx xxx。

接下来的任务就是执行pull命令：git pull xxx branch(其中xxx就是前面的origin,helloworld这些别名)。

简单的用法就介绍到这里。

下面说说一些用到的体会，总结。

(1) 首先是关于git remote add xxx xxx这个用法。这个做法是取别名，避免了每次都要输入完整的路径名，当然，你也可以这么做，当然没问题。

比如：git pull url master，就是把远程的url这个拉取到本机的master分支上。

可是每次都这么做多麻烦，为此使用别名，以后就直接通过别名来push，pull。

实际上，昨晚这个工作后，会在.git目录下面的config文件中写入一个项，用来指示该别名的信息,如下所示：

[remote "origin"]

url = git@github.com:xxx/rep.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

上面就是别名origin这个别名的信息。记录了目标远程版本库的地址。

(2) SSH方式还是HTTPS方式。

在版本库的SSH方式和HTTPS方式是不同的，具体来说就是url信息的不同，但是，实际的认证机制也是不同的。当建立了本机密钥之后，使用ssh方式实际上是不需要再次认证的，而https则每次需要输入密码。

比如，上面的url = git@github.com:xxx/rep.git使用的是ssh方式，那么我进入一个版本库中，push到远程的时候，是不需要输入用户名和密码的，这个只在你初始时候使用到。而如果我的url是以https方式来做的话，如url = https://xxxx，这样我在进行更新，push或者pull的时候，都会提示你输入GitHub的帐号和密码的。实际上，GitHub是鼓励使用 ssh方式的，在官方文档也有说明，不过对于无法使用ssh服务的地方来说，https又多了一种方式，给用户也带来了方便。

(3) 常用的命令和Git的使用似乎并不需要有太多的说明，但是要想掌握这个工具还是需要很大的尝试和努力。这里推荐下图形化使用工具 TGit（Windows）以及gitg(Linux)版本。通过图形化工具能够帮助我们熟悉提交图，掌握基本的操作，有时候图形化界面能够极大的方便我们操作，比如右键可以打Tag，查看分支，所有分支，Diff等等。

在Windows下面，先要安装Git程序，然后安装TGit，指定Git路径，此时可以利用Cygwin的命令行界面来操作Git，同时可以在右键菜单中使用TGit来查看提交图。

在Linux下面，就直接使用命令行来工作，同时利用gitg等图形化工具来观察提交图等。

(4) 协同工作。在GitHub上面有很多开源项目可以参与到开发学习，可以搜索相关项目，fork该项目，然后在自己的版本库中就会出现，此时通过pull可以更新到本机，然后再进行开发，当你开发之后push上去，再通过向开发者发送请求让其了解到你进行的修复，然后他根据自己的判断是否采纳，更新版本库来决定是否让你的这次提交并入到版本库中。

(4)更多关于Github的使用请参见官网：https://github.com,里面讲述了如何从建立版本库到创建分支，管理版本库的知识。

**Git分支管理**

Git的分支管理是Git的神器。拥有了它就会使我么管理代码更加游刃有余。那么什么是Git的分支管理？为什么要使用Git的分支管理？Git分支管理怎么用？  
在集中式版本控制中，冲突的合并是可怕的，是令人恶心的。所以很多版本控制软件通过加锁来拒绝多个人同时访问一个文件；而有的版本管理软件，则不是通过加锁的方式，第一个提交的人会很顺畅，但是如果第二个人提交，那么面临它的将是恶心的冲突解决。  
而在分布式管理软件中，冲突解决、合并、衍合，则是一种容易的事情，它是版本管理中的常态。  
 而合并、衍合的主体就是分支。  
分支其实就是指向某种代码状态的一个指针。而合并其实就是将两种代码状态合并到另一种代码状态中。  
 在Git中，正确的使用方法中，无处不在使用分支。比如，提交实际上就是本地分支合并到远程分支，更新实际上就是将远程分支合并到本地分支，在开发过程中，每加入一个功能或特性，都加入一个分支，当实验成功后合并到主分支...

**为什么要使用分支管理？**我们来设想下面几种情况：

1、我们在基于一个稳定的版本在进行开发，突然在稳定版本上有一个紧急的bug需要我们解决。

2、我们在软件中加入了一个小的特性，但是开发到一半的时候，发现开发组的另一个的想法更有创意，所以我们想废弃自己的更改。

3、自己想在软件中同时加入多个特性，但是希望并行开发开发，而不是依次开发。

如果采用单分支形式的话，以上可能也可以实现，但是实现的复杂度可能就会加大。而应用多分支管理时情况就变的简单了。  
如果我们开发新功能时是基于一个新的分支的话，如果稳定版本有一个紧急bug需要处理，那么我们就可以切换到稳定版本的分支，然后修改bug，修改之后，我们再次切换到原先的分支继续工作，最后我们将该分支合并到稳定分支即可。如果我们想废弃正在开发的某个特性，如果该特性在一个单独的分支上，只需要简单的删除该分支即可。如果我们想并行开发多个特性，我们可以创建多个分支，分别开发，然后将每个分支都合并到稳定分支上即可。  
多分支管理，我们可以维护一个稳定的分支，然后某些特性或实验性的开发可以单独作为一个分支，这样开发过程就不会影响到稳定的版本。而且Git中分支的创建和切换基本上没有多少消耗。

**Git如何进行分支管理？  
1、创建分支**创建分支很简单：git branch <分支名>

**2、切换分支**git checkout <分支名>  
该语句和上一个语句可以和起来用一个语句表示：git checkout -b <分支名>

**3、分支合并**比如，如果要将开发中的分支（develop），合并到稳定分支（master），  
首先切换的master分支：git checkout master。  
然后执行合并操作：git merge develop。  
如果有冲突，会提示你，调用git status查看冲突文件。  
解决冲突，然后调用git add或git rm将解决后的文件暂存。  
所有冲突解决后，git commit 提交更改。

**4、分支衍合**分支衍合和分支合并的差别在于，分支衍合不会保留合并的日志，不留痕迹，而 分支合并则会保留合并的日志。  
要将开发中的分支（develop），衍合到稳定分支（master）。  
首先切换的master分支：git checkout master。  
然后执行衍和操作：git rebase develop。  
如果有冲突，会提示你，调用git status查看冲突文件。  
解决冲突，然后调用git add或git rm将解决后的文件暂存。  
所有冲突解决后，git rebase --continue 提交更改。

**5、删除分支**执行git branch -d <分支名>  
如果该分支没有合并到主分支会报错，可以用以下命令强制删除git branch -D <分支名>