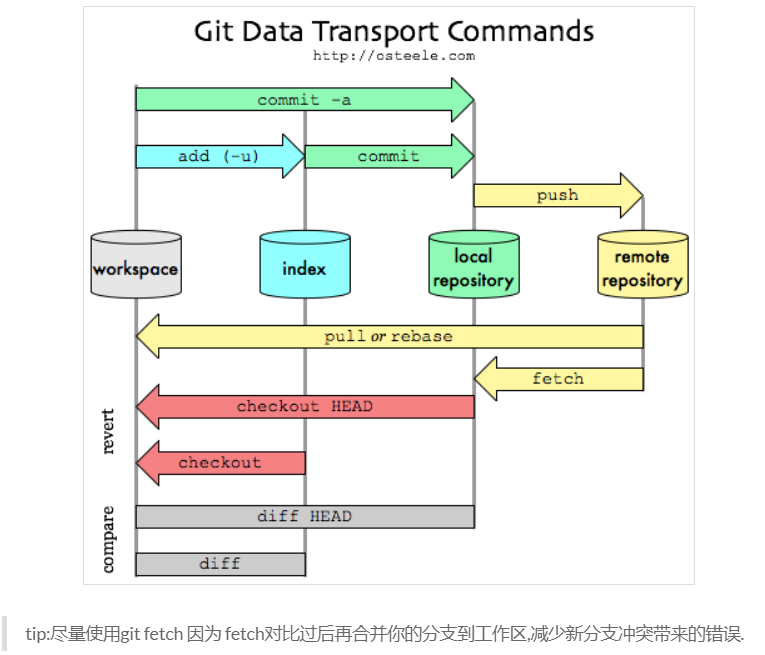
# Git安装



## 在Linux上安装Git

首先，你可以试着输入git，看看系统有没有安装Git：

$ git

The program 'git' is currently not installed. You can install it by typing:

su**do** apt-**get** install git

像上面的命令，有很多Linux会友好地告诉你Git没有安装，还会告诉你如何安装Git。

如果你碰巧用Debian或Ubuntu Linux，通过一条sudo apt-get install git就可以直接完成Git的安装，非常简单。

老一点的Debian或Ubuntu Linux，要把命令改为sudo apt-get install git-core，因为以前有个软件也叫GIT（GNU Interactive Tools），结果Git就只能叫git-core了。由于Git名气实在太大，后来就把GNU Interactive Tools改成gnuit，git-core正式改为git。

如果是其他Linux版本，可以直接通过源码安装。先从Git官网下载源码，然后解压，依次输入：./config，make，sudo make install这几个命令安装就好了。

## 创建本地项目目录

什么是版本库呢？版本库又名仓库，英文名**repository**，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

所以，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：

$ mkdir learngit

$ cd learngit

$ pwd

/Users/michael/learngit

pwd命令用于显示当前目录。在我的Mac上，这个仓库位于/Users/michael/learngit。

 如果你使用Windows系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

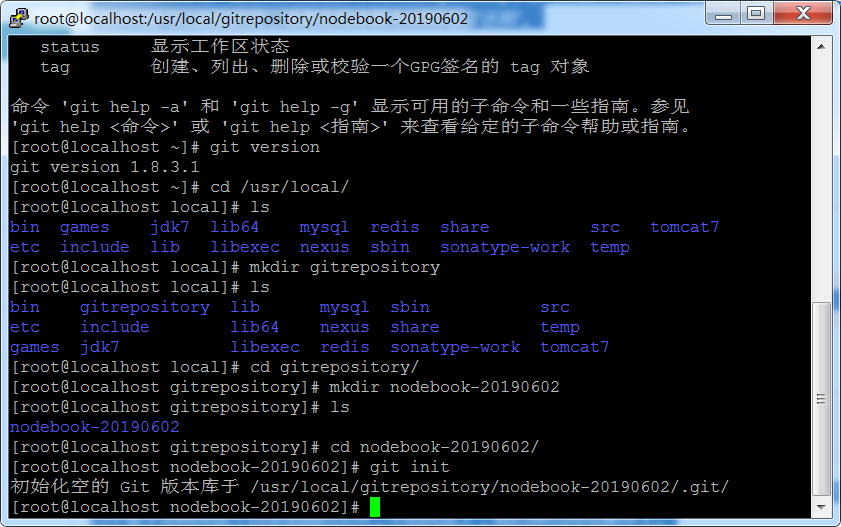
第二步，通过git init命令把这个目录变成Git可以管理的仓库：

$ git init

Initialized empty Git repository **in** /Users/michael/learngit/.git/

瞬间Git就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository），细心的读者可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

如果你没有看到.git目录，那是因为这个目录默认是隐藏的，用ls -ah命令就可以看见。



## 添加文件

文件一定要位于创建的本地项目目录文件夹下，否则git无法找到文件

和把大象放到冰箱需要3步相比，把一个文件放到Git仓库只需要两步。

第一步，用命令git add告诉Git，把文件添加到仓库：

$ git add readme.txt

执行上面的命令，没有任何显示，这就对了，Unix的哲学是“没有消息就是好消息”，说明添加成功。



第二步，用命令git commit告诉Git，把文件提交到仓库：

$ git **commit** -m "wrote a readme file"

[master (root-**commit**) eaadf4e] wrote a readme file

1 file changed, 2 insertions(+)

**create** mode 100644 readme.txt

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

嫌麻烦不想输入-m "xxx"行不行？确实有办法可以这么干，但是强烈不建议你这么干，因为输入说明对自己对别人阅读都很重要。实在不想输入说明的童鞋请自行Google，我不告诉你这个参数。

git commit命令执行成功后会告诉你，1 file changed：1个文件被改动（我们新添加的readme.txt文件）；2 insertions：插入了两行内容（readme.txt有两行内容）。

### 小结

现在总结一下今天学的两点内容：

初始化一个Git仓库，使用git init命令。

添加文件到Git仓库，分两步：

1. 使用命令git add <file>，注意，可反复多次使用，添加多个文件；
2. 使用命令git commit -m <message>，完成

* 要随时掌握工作区的状态，使用git status命令。
* 如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

git add 文件夹/            添加整个文件夹及内容

git add \*.文件类型       添加目录中所有此文件类型的文件

## 关联github远程仓库

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：

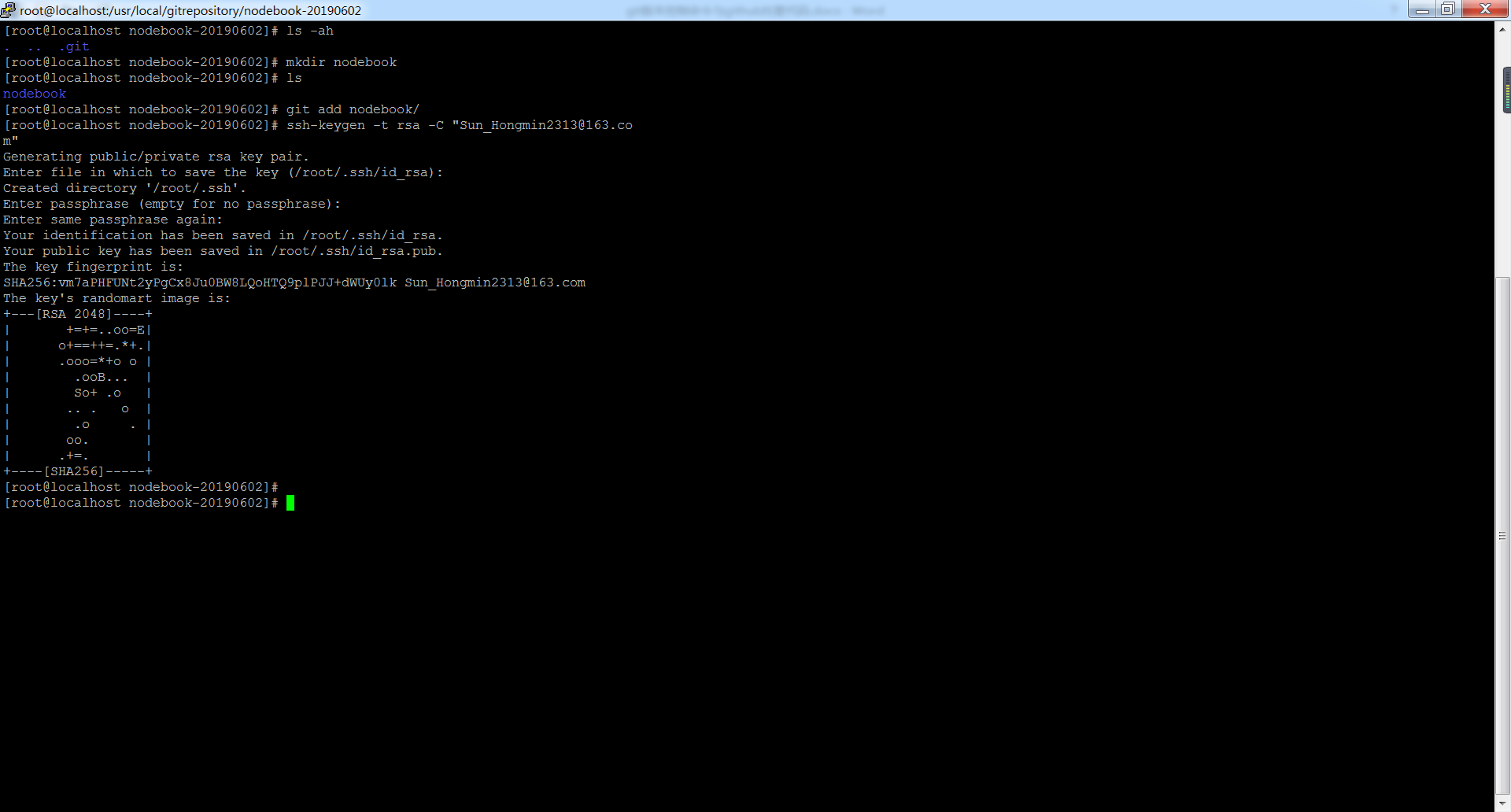
$ ssh-keygen -t rsa -C "youremail@example.com"

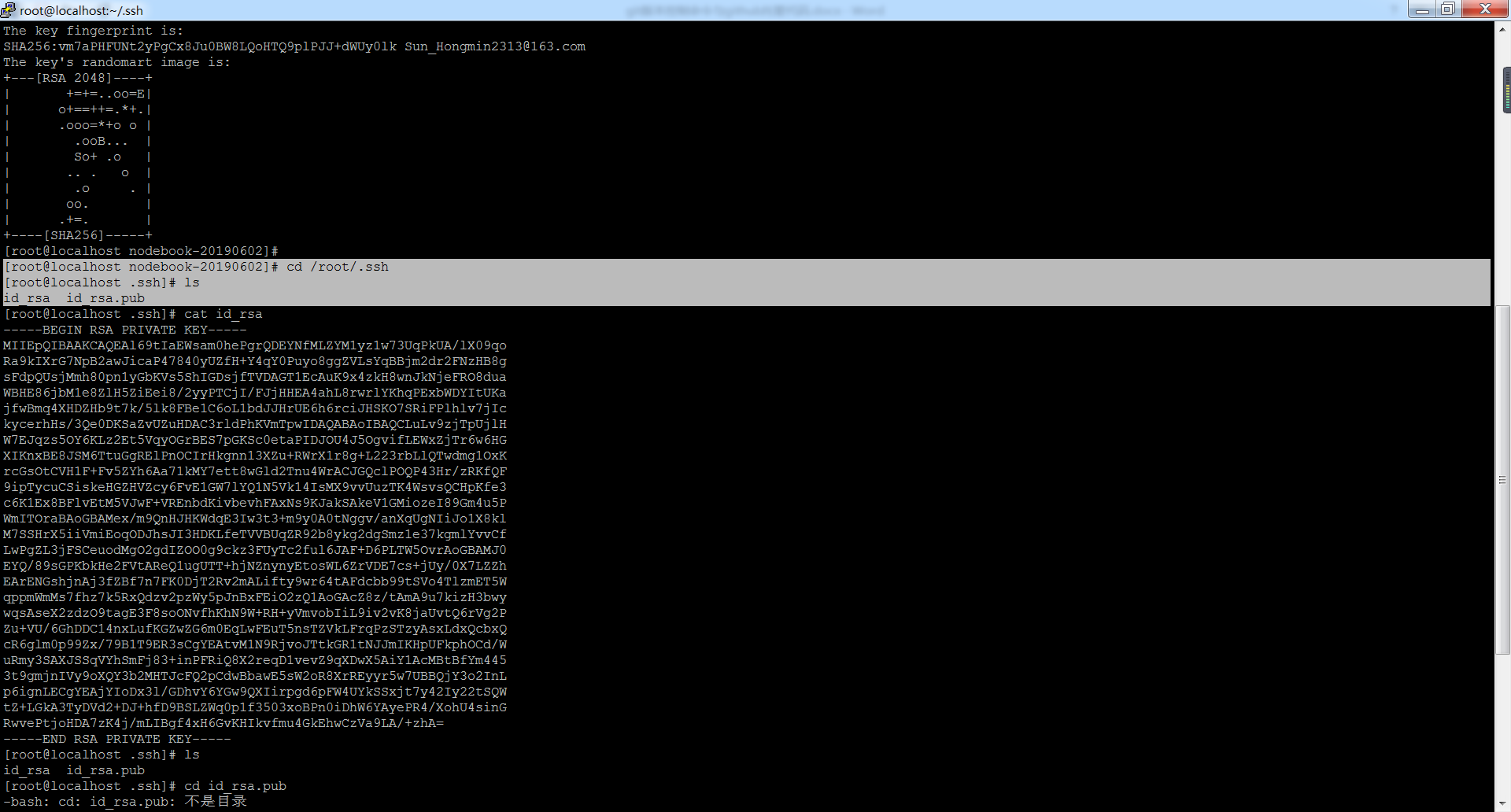
你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

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：





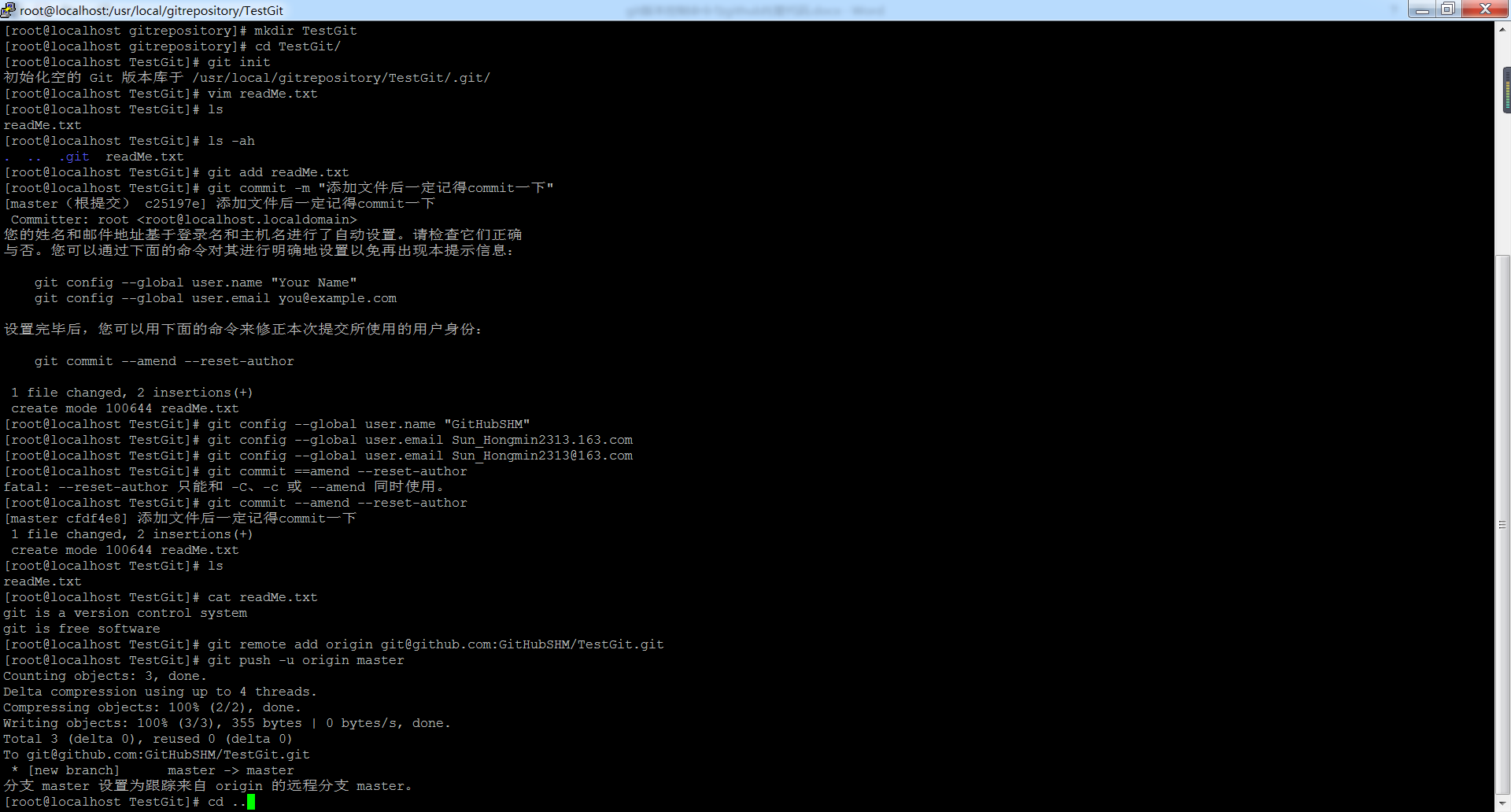
为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。

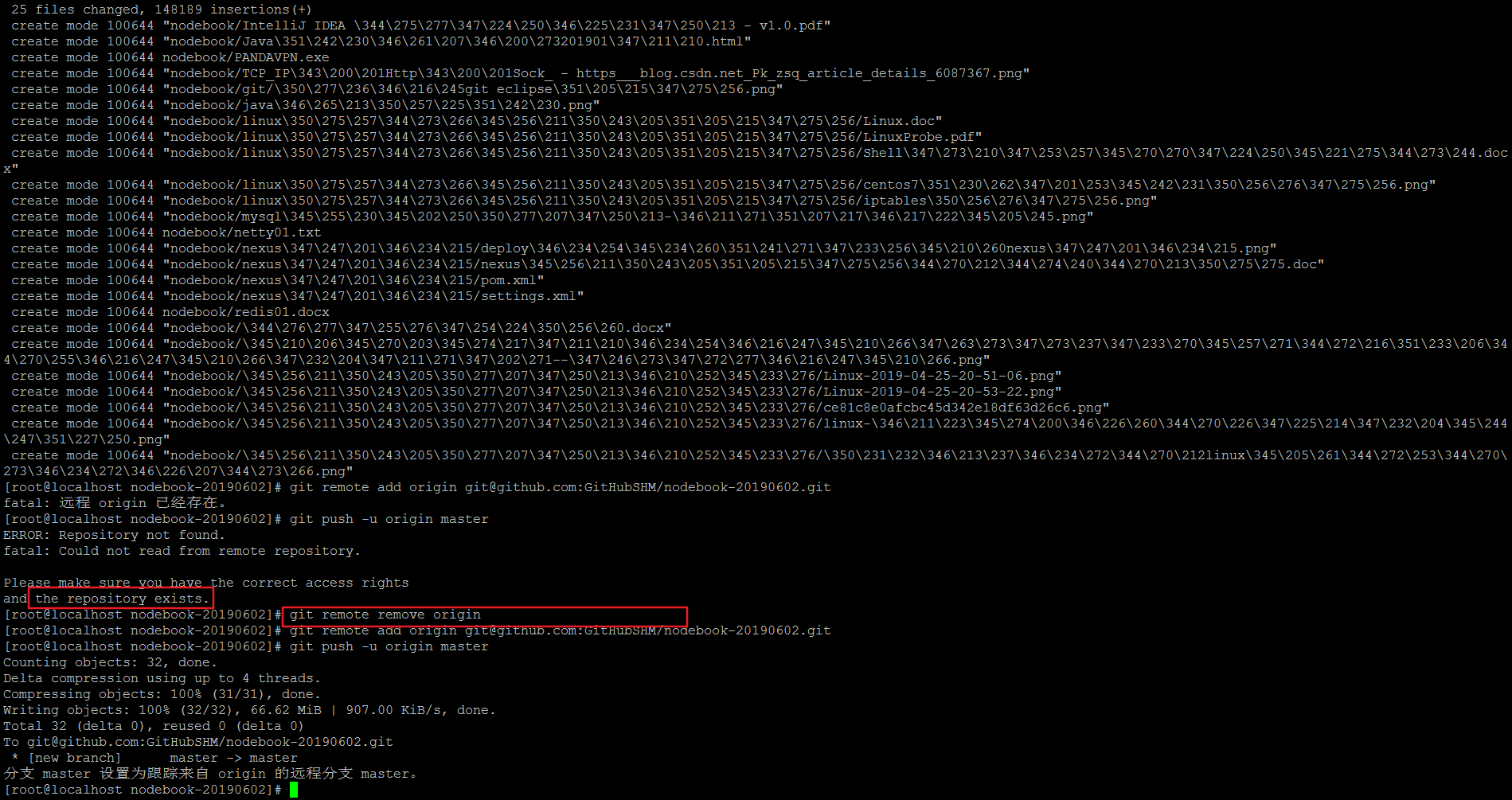
当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

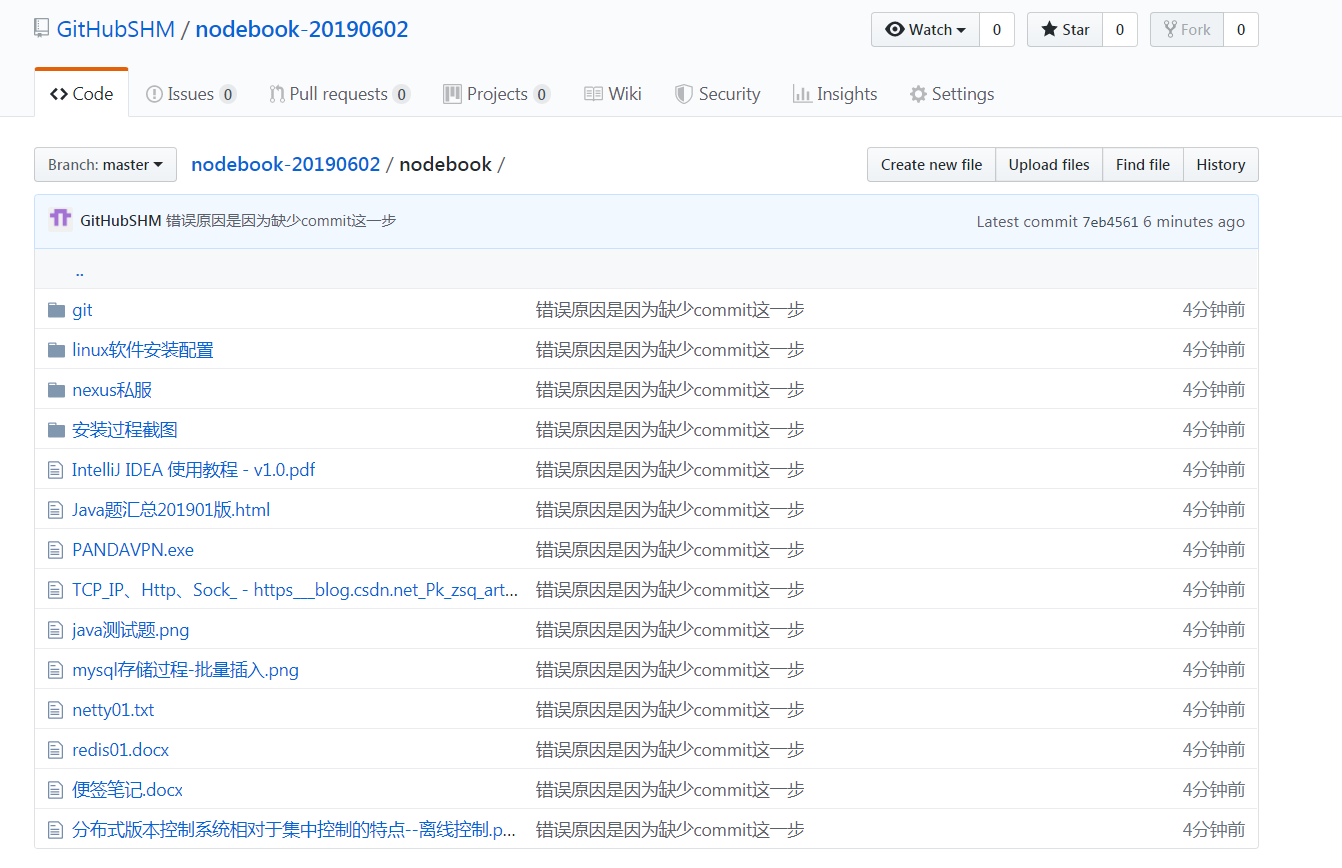
最后友情提示，在GitHub上免费托管的Git仓库，任何人都可以看到喔（但只有你自己才能改）。所以，不要把敏感信息放进去。

如果你不想让别人看到Git库，有两个办法，一个是交点保护费，让GitHub把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个Git服务器，因为是你自己的Git服务器，所以别人也是看不见的。这个方法我们后面会讲到的，相当简单，公司内部开发必备。

确保你拥有一个GitHub账号后，我们就即将开始远程仓库的学习。







## 前五点总结

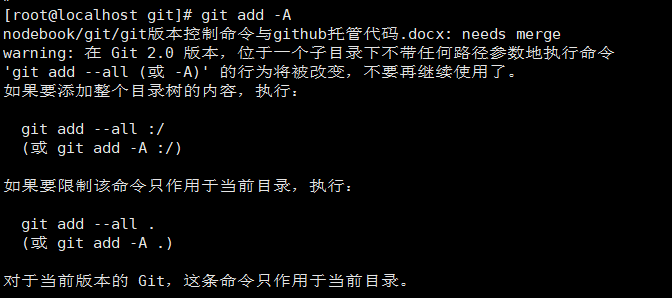
首先确保git是否已经安装好

在本地创建项目仓库路径

使用 #git init 初始化当前目录为项目版本库（项目节点）

添加文件后使用 #git add <文件名.拓展名> || <文件夹/> 将文件或文件夹添加到仓库中

\*#git add -A ---------同步本地所有文件到暂存区（批量add）



\*添加后一定记得提交 #git commit –m “提交注释” （该提交命令可以一次提交多个add命令添加的文件，不需要一次add 一次commit）

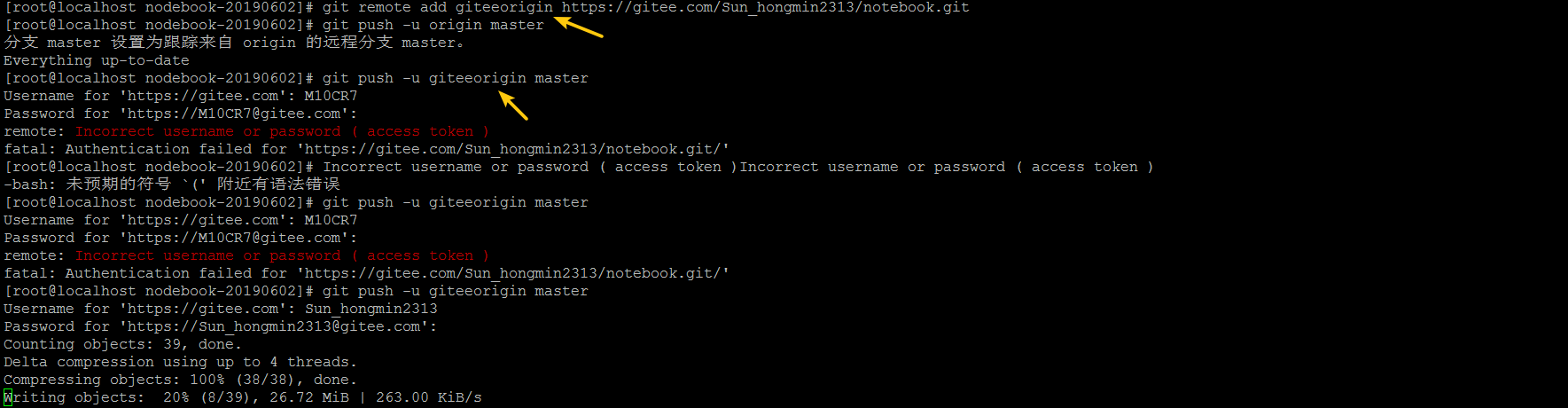
登录github，添加项目仓库

在本地项目仓库下（git init命令执行时的目录下）使用#git remote add origin [git@github.com:<github账户名>/](mailto:git@github.com:%3cgithub账户名%3e/)<仓库名>.git

使用#git push –u origin master 将本地文件推送到远程仓库中

后续的提交使用命令 #git push origin master

### git代码托管----码云



## 远程仓库克隆

克隆远程仓库：

#git clone [git@github.com:[用户名]/](mailto:git@github.com:[用户名]/)仓库名.git

$ git clone git@github.com:michaelliao/gitskills.git

Cloning into 'gitskills'...

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3

Receiving objects: 100% (3/3), done.

如果有多个人协作开发，那么每个人各自从远程克隆一份就可以了。

你也许还注意到，GitHub给出的地址不止一个，还可以用https://github.com/michaelliao/gitskills.git这样的地址。实际上，Git支持多种协议，默认的git://使用ssh，但也可以使用https等其他协议。

使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用ssh协议而只能用https。

### 关于SSH协议

一、什么是SSH

SSH是英文Secure Shell的简写形式。通过使用SSH，你可以把所有传输的数据进行加密，这样"中间人"这种攻击方式就不可能实现了，而且也能够防止DNS欺骗和IP欺骗。使用SSH，还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。SSH有很多功能，它既可以代替Telnet，又可以为FTP、Pop、甚至为PPP提供一个安全的"通道"。

原文：https://blog.csdn.net/macrossdzh/article/details/5691924

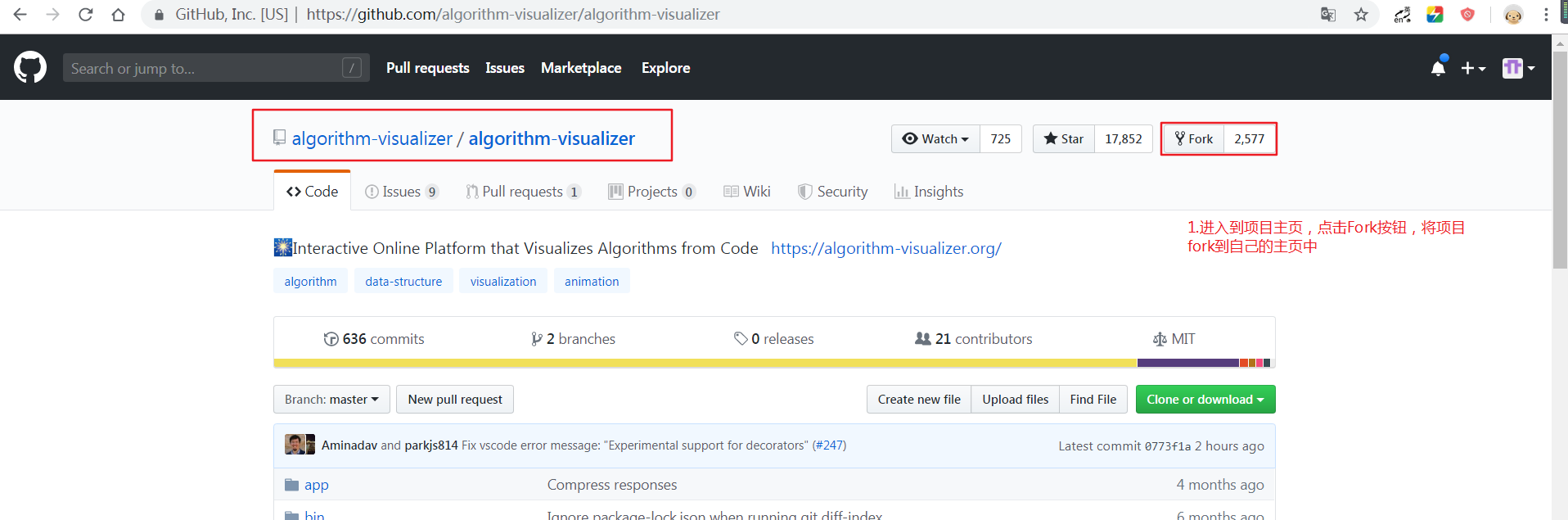
### 小结

要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

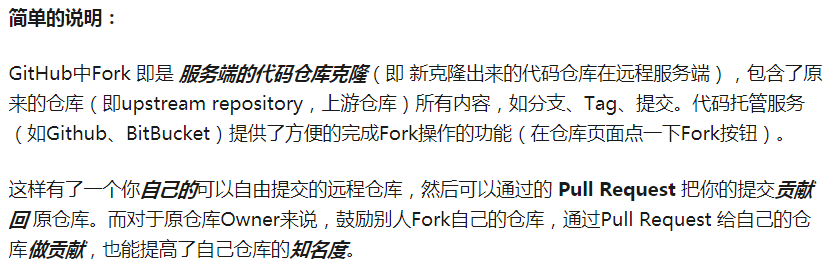
Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

### 远程仓库克隆的先决条件

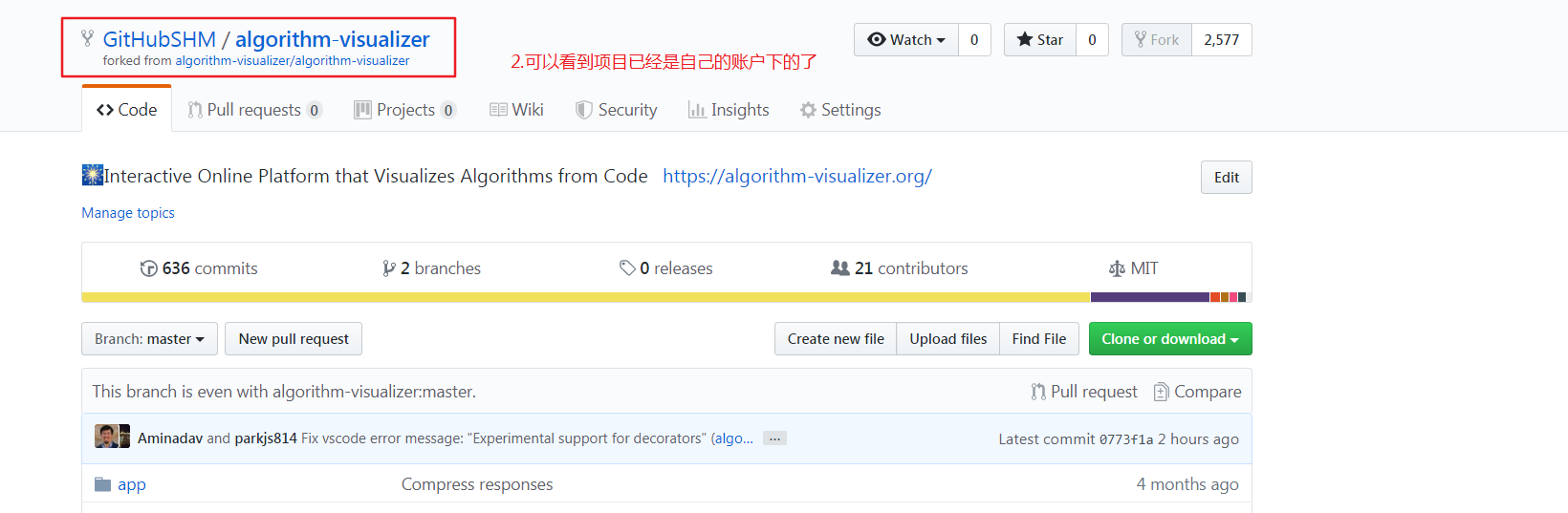
1. 如果是自己的git（github、码云）账号，那么需要在本地生成pubkey添加到账户的setting中，然后使用克隆命令进行克隆
2. 如果是他人的项目，需要在项目主页点击fork，然后在重复步骤1



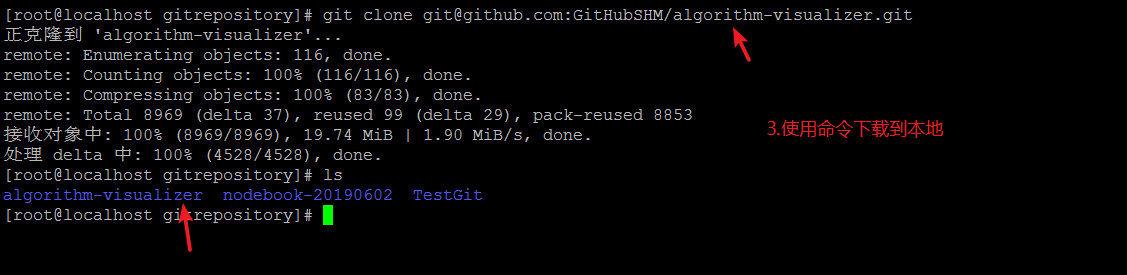
进入项目主页点击“fork”



Fork名词解释



成功后看到自己主页下已经存在此项目了



Clone项目

## 版本回退

不断对文件进行修改，然后不断提交修改到版本库里，就好比玩RPG游戏时，每通过一关就会自动把游戏状态存盘，如果某一关没过去，你还可以选择读取前一关的状态。有些时候，在打Boss之前，你会手动存盘，以便万一打Boss失败了，可以从最近的地方重新开始。Git也是一样，每当你觉得文件修改到一定程度的时候，就可以“保存一个快照”，这个快照在Git中被称为commit。一旦你把文件改乱了，或者误删了文件，还可以从最近的一个commit恢复，然后继续工作，而不是把几个月的工作成果全部丢失。

版本控制系统肯定有某个命令可以告诉我们**\*历史记录**，在Git中，我们用git log命令查看：

$ git log

commit 1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master)

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 21:06:15 2018 +0800

append GPL

commit e475afc93c209a690c39c13a46716e8fa000c366

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 21:03:36 2018 +0800

add distributed

commit eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 20:59:18 2018 +0800

wrote a readme file

git log命令显示从最近到最远的提交日志，我们可以看到3次提交，最近的一次是append GPL，上一次是add distributed，最早的一次是wrote a readme file。

**\*简化的日志信息**

如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline（\*\*oneline而不是online哦）：

$ git log --pretty=oneline

1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL

e475afc93c209a690c39c13a46716e8fa000c366 add distributed

eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0 wrote a readme file

$ git log --pretty=oneline

1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL

e475afc93c209a690c39c13a46716e8fa000c366 add distributed

eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0 wrote a readme file

看到的一大串类似1094adb...的是commit id（版本号）, 和SVN不一样，Git的commit id不是1，2，3……递增的数字，而是一个SHA1计算出来的一个非常大的数字，用十六进制表示.

**\*版本回退**

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

现在，我们要把当前版本append GPL回退到上一个版本add distributed，就可以使用git reset命令：

$ git reset --hard HEAD^

HEAD is now at e475afc add distributed

最新的那个版本append GPL已经看不到了！好比你从21世纪坐时光穿梭机来到了19世纪，想再回去已经回不去了，肿么办？

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个append GPL的commit id是1094adb...，于是就可以指定回到未来的某个版本：

$ git reset --hard 1094a

HEAD is now at 83b0afe append GPL

版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

**\*无法找到回退之前版本的commitid，怎样回退版本**



### 小结

现在总结一下：

* HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。
* 穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。
* 要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

## 工作区、暂存区、版本库概念解释

**工作区：**

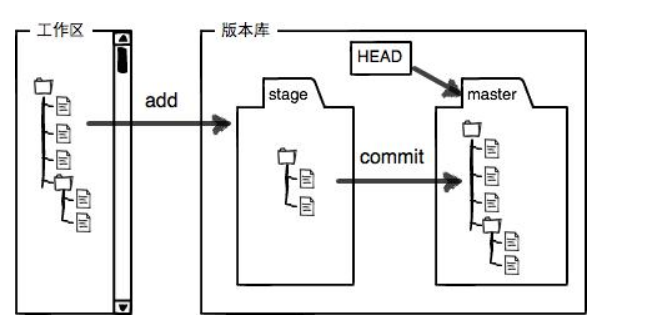
在磁盘中所能看到的文件，就是使用#git init时所在的目录内即为工作区（不包括.git文件）

**版本库：**

在工作区中隐藏的目录.git,是git用来管理版本信息的仓库，即为版本库

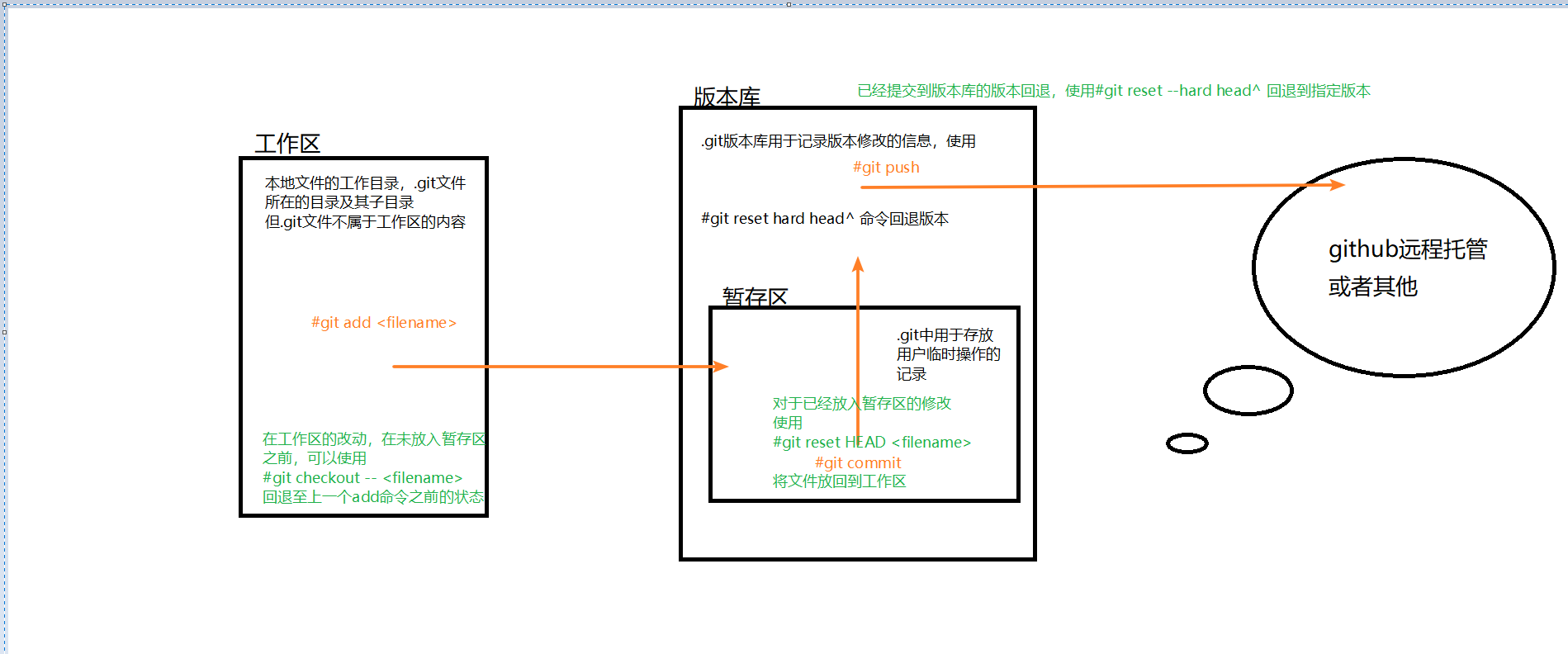
**暂存区：**

位于版本库内，在使用add/rm操作文件但尚未提交时，此时文件的信息位于暂存区



类似于购物车，添加或者删除商品的操作信息位于购物车内，但是实际买不买还要通过付款最终确定。

## 本地撤销修改



### 丢弃工作区修改

命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

### 丢弃暂存区修改

使用add命令后，工作区内容被提交到暂存区，已被提交的内容使用checkout将不会在撤回了，前提是需要将暂存区的内容撤销到工作区再使用checkout撤销

**Git同样告诉我们，用命令git reset HEAD <file>可以把暂存区的修改撤销掉（unstage），重新放回工作区（这里的撤销只是位置变动，而非内容撤销，内容撤销还是要再使用checkout的）**

### 丢弃版本库的修改（也就是[版本回退](#_版本回退)）

现在，假设你不但改错了东西，还从暂存区提交到了版本库，怎么办呢？还记得[版本回退](#_版本回退)一节吗？可以回退到上一个版本。不过，这是有条件的，就是你还没有把自己的本地版本库推送到远程。

### 小结

又到了小结时间。

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/896043488029600/897013573512192)一节，不过前提是没有推送到远程库。

## 同步远程到本地

**更新远程文件到本地方式一**

查看远程仓库

#git remote -v

从远程获取最新版本到本地

#git fetch origin aaa

比较远程分支和本地分支

# git log -p aaa origin/aaa

合并远程分支到本地

#git merge origin/aaa

**√远程文件到本地方式二，在本地建临时分支，合并后删除**

查看远程仓库

#git remote -v

从远程获取最新版本到本地

#git fetch origin master:temp

比较远程分支和本地分支的区别

#git diff temp

合并远程分支到本地

#git merge temp

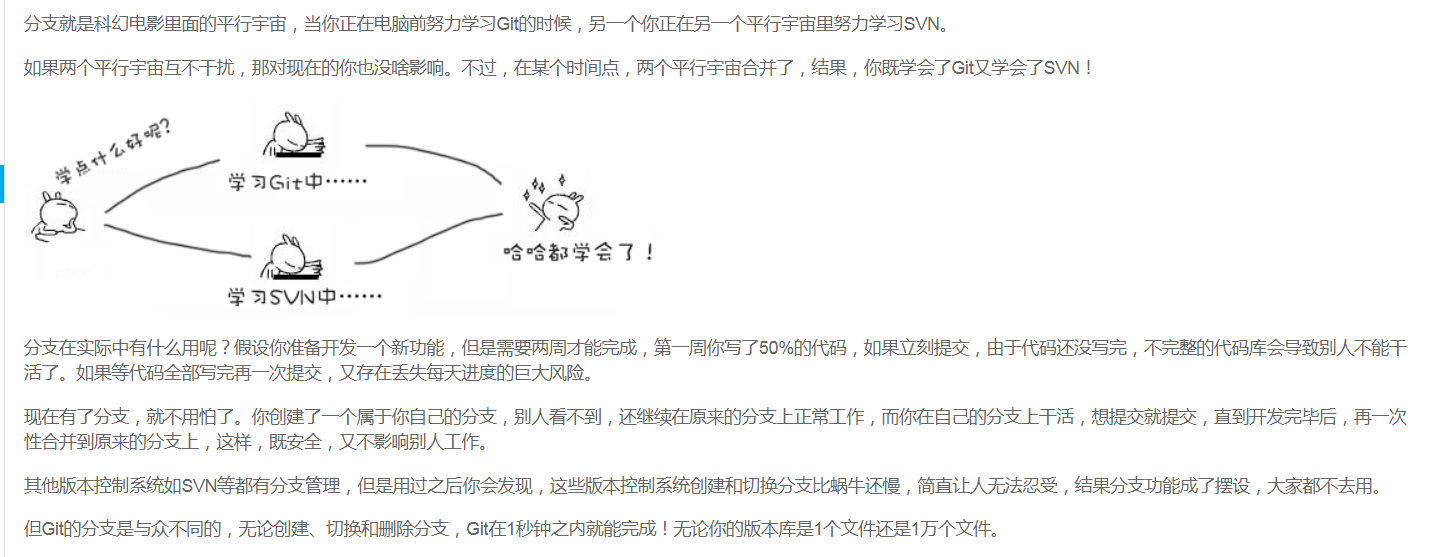
**使用pull更新**

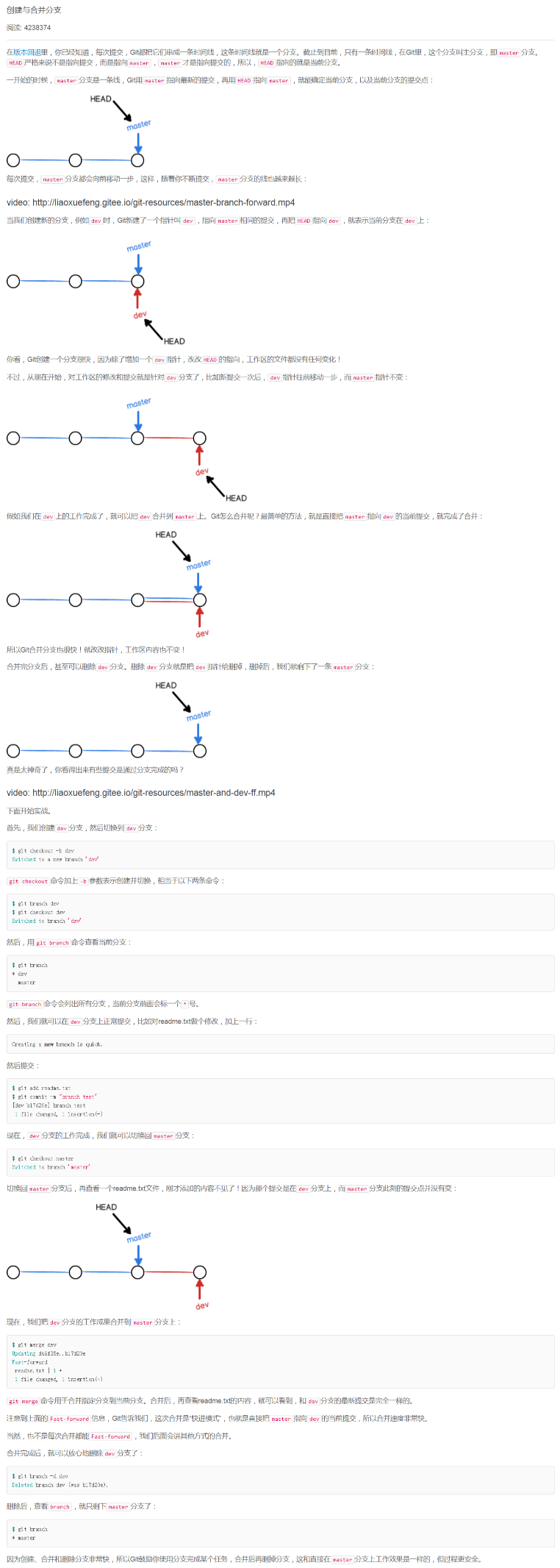
#git pull origin aaa

## 分支管理

### 创建、切换、合并、删除分支

项目的主分支为master,在项目开发过程中，如果一次任务需要较长时间来完成，中间又不能将完成一半的功能提交上去，而又怕丢失进度，可以创建一个项目分支，在分支可以自由的进行版本控制，而在功能完成之后可以合并到主干分支。





**小结**

Git鼓励大量使用分支：

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> （当前分支将被<name>分支覆盖）

删除分支：git branch -d <name>

合并远程到本地 #git pull

### 删除远程分支



## 解决合并文件冲突

当两个分支同时修改了相同的文件并且提交时，进行merge合并时会出现冲突的提示

这时需要将冲突文件解决冲突后再次提交（文本文件直接打开解决，二进制文件例如word等文件只能取舍使用其中一个覆盖另一个）

#git reset –hard 分支

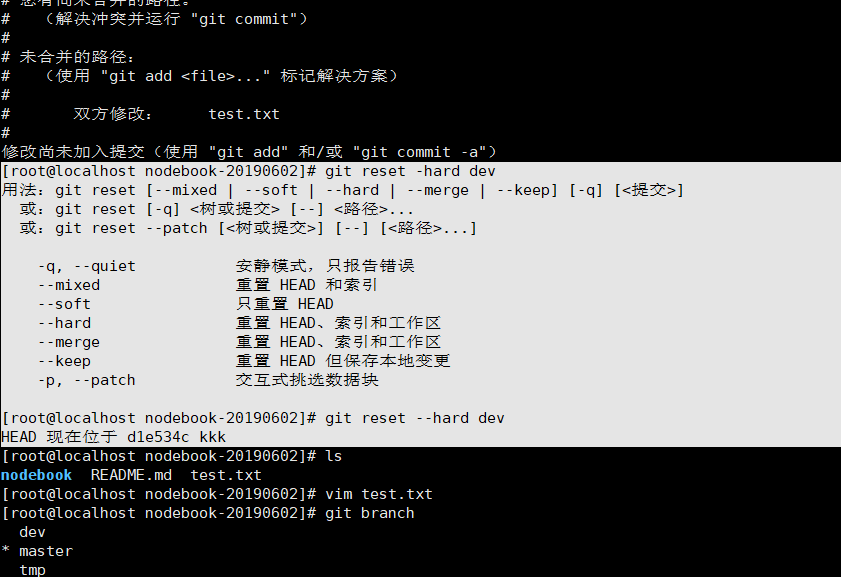
使用分支覆盖当前分支，重置了HEAD、索引和工作区

#git push origin/master –force

推送远程（强制）

#$ git log --graph --pretty=oneline --abbrev-commit

查看分支合并情况



### 小结

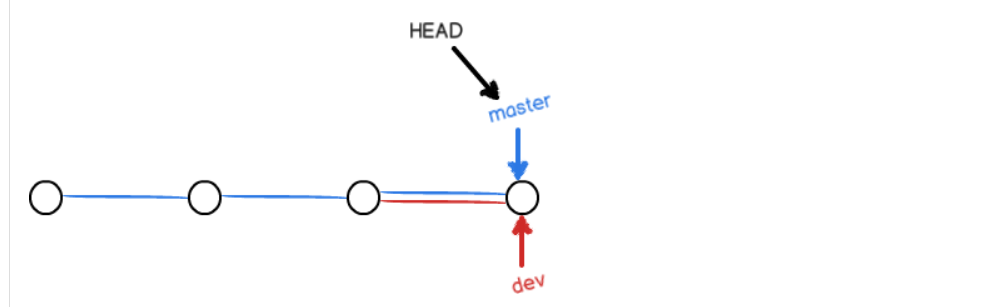
当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。

用git log --graph命令可以看到分支合并图。

## 分支管理策略

通常情况下，在分支之间进行合并时，默认使用Fast forword快速合并，



合并之后删除分支会丢失分支的提交信息，在merge时加入—no-ff参数可以生成一个新的提交并填写新的提交信息

#git merge - -no-ff -m “合并dev分支” dev

将dev分支合并到当前分支，并且新的提交信息为“合并dev分支”

合并后，我们用git log看看分支历史：

$ git log --graph --pretty=oneline --abbrev-commit

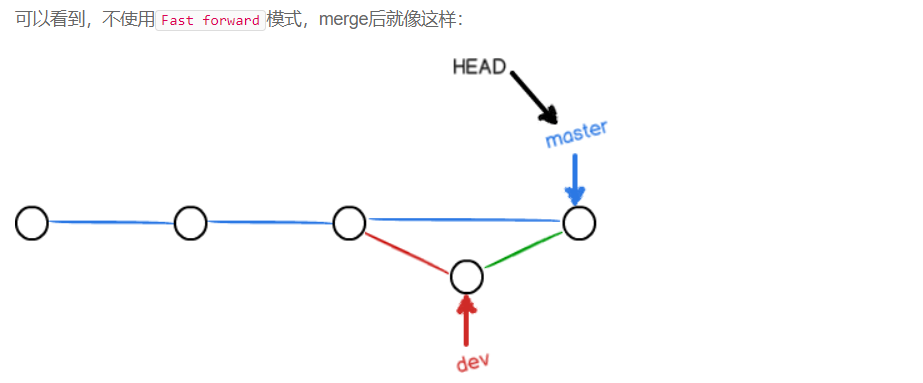
\* e1e9c68 (HEAD -> master) merge with no-ff

|\

| \* f52c633 (dev) add merge

|/

\* cf810e4 conflict fixed





分支策略：master仅用来发布新版本，dev分支用来充当发布前的合并测试版本。

## 保存工作现场，排查紧急bug

当任务进行一半此时又要切换到其他分支上完成紧急的bug修复，可以使用stash保存工作状态,等之后可以恢复现场继续工作

#git stash

使用 stash list查看工作现场列表

#git stash list

回复工作现场有两种方法

1. 先使用stash apply回复，stash信息仍然保留，可以再用stash drop删除（只有一个现场的情况下）
2. 直接使用stash pop，恢复的同时将stash信息删除

你可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令：

$ git stash apply stash@{0}

### 小结

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场。

## 强制删除分支

开发一个新feature，最好新建一个分支；

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

## 多人协作

当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。

要查看远程库的信息，用git remote：

$ git remote

origin

或者，用git remote -v显示更详细的信息：

$ git remote -v

origin git@github.com:michaelliao/learngit.git (fetch)

origin git@github.com:michaelliao/learngit.git (push)

上面显示了可以抓取和推送的origin的地址。如果没有推送权限，就看不到push的地址。

#### 推送分支

推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上：

$ git push origin master

如果要推送其他分支，比如dev，就改成：

$ git push origin dev

但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，就是在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！

未完待续<https://www.liaoxuefeng.com/wiki/896043488029600/900375748016320>

## 忽略特殊文件

项目中一些文件不需要与远程进行同步，可以通过配置的方式忽略这些文件

配置方法：在git工作区的根目录下创建一个特殊的.gitignore文件，然后把需要忽略的文件名填进去

不需要从头写.gitignore文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：[https://github.com/github/gitignore](https://github.com/github/gitignore" \t "_blank)

忽略文件的原则是：

1. 忽略操作系统自动生成的文件，比如缩略图等；
2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的.class文件；
3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

## 配置命令别名

将git status 简化为 git st

# git config –global alias.st status

--global参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。

配置一个git last，让其显示最后一次提交信息：

$ git config --global alias.last 'log -1'

这样，用git last就能显示最近一次的提交：

$ git last

commit adca45d317e6d8a4b23f9811c3d7b7f0f180bfe2

Merge: bd6ae48 291bea8

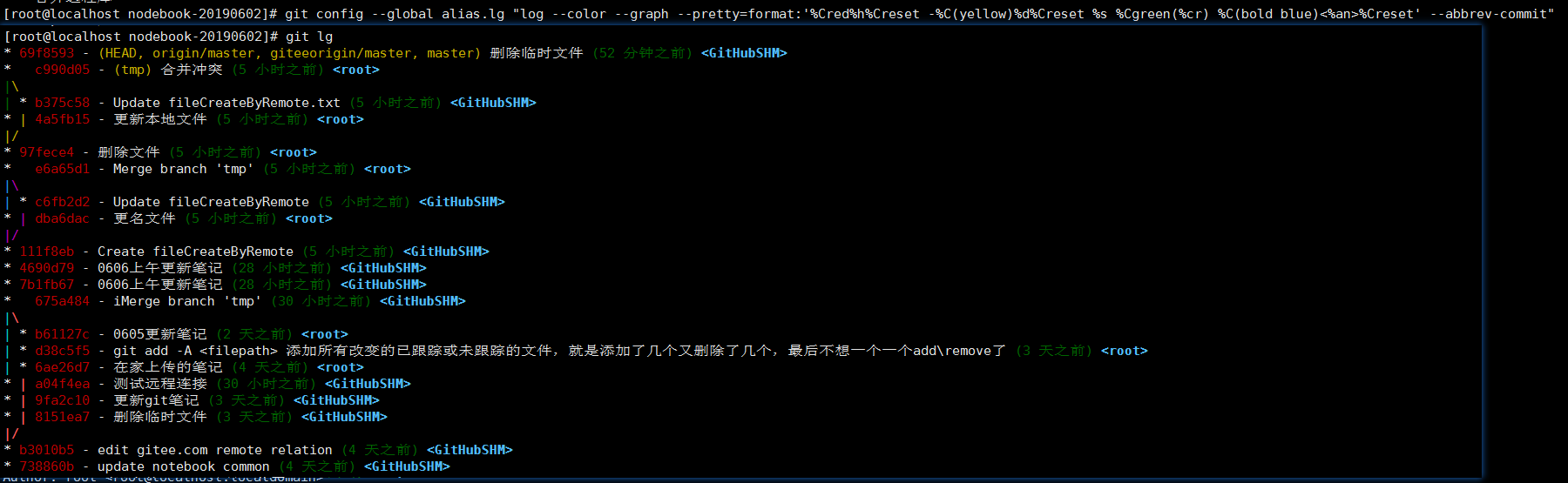
Author: Michael Liao <askxuefeng@gmail.com>

Date: Thu Aug 22 22:49:22 2013 +0800

merge & fix hello.py

配置不同日志信息显示不同颜色

# 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"



配置Git的时候,加上—global是针对当前用户起作用的，如果不加，那就只针对当前的仓库起作用

每个仓库的单独配置文件放在 .git/config

$ cat .git/config

[core]

repositoryformatversion = 0

filemode = true

bare = false

logallrefupdates = true

ignorecase = true

precomposeunicode = true

[remote "origin"]

url = git@github.com:michaelliao/learngit.git

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

[branch "master"]

remote = origin

merge = refs/heads/master

[alias]

last = log -1

别名在[alias]下，要删除别名只需要删除对应行即可

全局配置文件放在用户主目录文件下 .gitconfig 隐藏文件

配置别名也可以直接修改这个文件，如果改错了，可以删掉文件重新通过命令配置。

1.查看用户的配置信息： git config --global --list 要查看当前仓库的，把global改成local就好

2.仓库的 .git/config 只存当前仓库的local配置 而用户的global配置在用户主目录下的 .gitconfig中

## 版本标签及说明管理

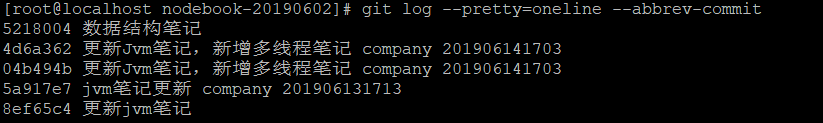
在版本更迭中，每commit一次就是一个版本，如果想要为当前提交的版本加一个标签方便查阅，则使用

#git tag <tagname>

设置当前提交的版本的标签，使用#git tag 查看标签列表

如果想为之前版本加入标签，首先查看版本记录

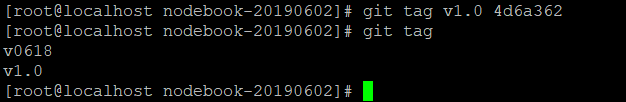
#git log –pretty=oneline –abbrev-commit



显示commitid列表

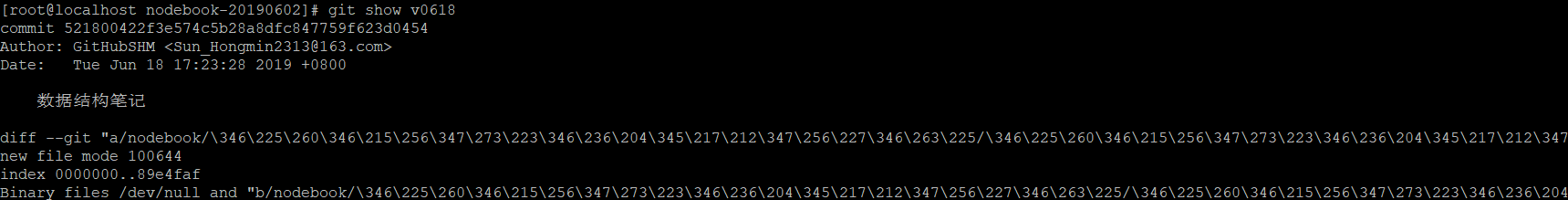
再使用

#git tag <tagname> commitid



来为历史版本添加版本标签,注意标签不是按时间排序的而是按字母顺序排序，可以用

# git show <tagname> 查看标签信息,使用q退出查看。



可以为标签附加说明文字

# git tag -a <tagname> -m “说明文字” [commitid]

