git操作手册

<https://github.com/guogenjuan/home-project>

guogenjuan@126.com/juan@@871378

## 环境配置

### 1、安装（windows）

<https://git-scm.com/downloads> 官网上下载安装，成功之后在开始菜单找到Git->Git Bash,弹出一个类似的窗口就说明安装成功了。



安装完成后，进一步设置：

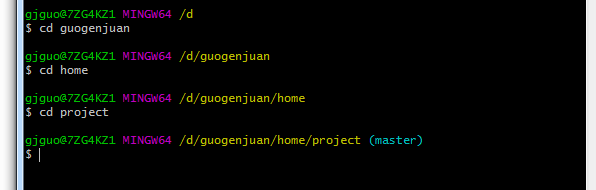
$ git config --global user.name "Your Name"$ git config --global user.email [email@example.com](mailto:email@example.com)

因为git是分布式版本控制系统，所以每个机器必须自报家门，你的名字和你的邮箱。

注意：git config命令的global参数，这个参数表示这太机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和email地址。

### 2、创建版本库（repository）

第一步：在本机上新建目录D:\guogenjuan\home\project（最好不含有中文），然后在git bash窗口，通过cd命令进入这个目录：



第二步：通过git init命令把这个目录变成Git可以管理的仓库

仓库这样就建好了，在目录下面多了一个.git目录。是git用来跟踪管理版本库的。

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。

### 3、文件添加到版本库

第一步：在仓库目录project下新建一个文件，readme.txt(windows下不要用记事本编辑任何文件，用notepad++、UE等工具，选择UTF-8格式)

第二步：告诉git，把文件添加到仓库

$ git add readme.txt

第三步：告诉git把文件提交到仓库

$ git commit -m "wrote a readme file"

### 4、小结

现在总结一下今天学的两点内容：

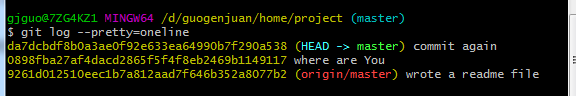
* 初始化一个Git仓库，使用git init命令。
* 添加文件到Git仓库，分两步：
* 使用命令git add <file>，注意，可反复多次使用，添加多个文件；
* 使用命令git commit -m <message>，完成。

## 本地版本库操作

### 版本管理

#### 版本回退

第一步：git log命令查看查看提交历史记录，如果嫌信息输出太多，可以加上--pretty=oneline



需要友情提示的是，你看到的一大串类似da7dcb…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，为什么commit id需要用这么一大串数字表示呢？因为Git是分布式的版本控制系统，后面我们还要研究多人在同一个版本库里工作，如果大家都用1，2，3……作为版本号，那肯定就冲突了。

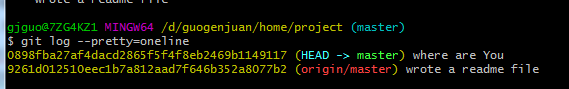
第二步：首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^,往上100个版本时，写成HEAD~100



现在查看readme的内容变成了上个版本！

#### 上个版本

通过1，版本还原到了上个版本，我们再git log下，发现“commit again”版本没有了



如果要还原上个版本，除非我们知道它的commit\_ id的前几位（git会自己去找，当然位数越多越好）

$ git reset --hard da7dcb

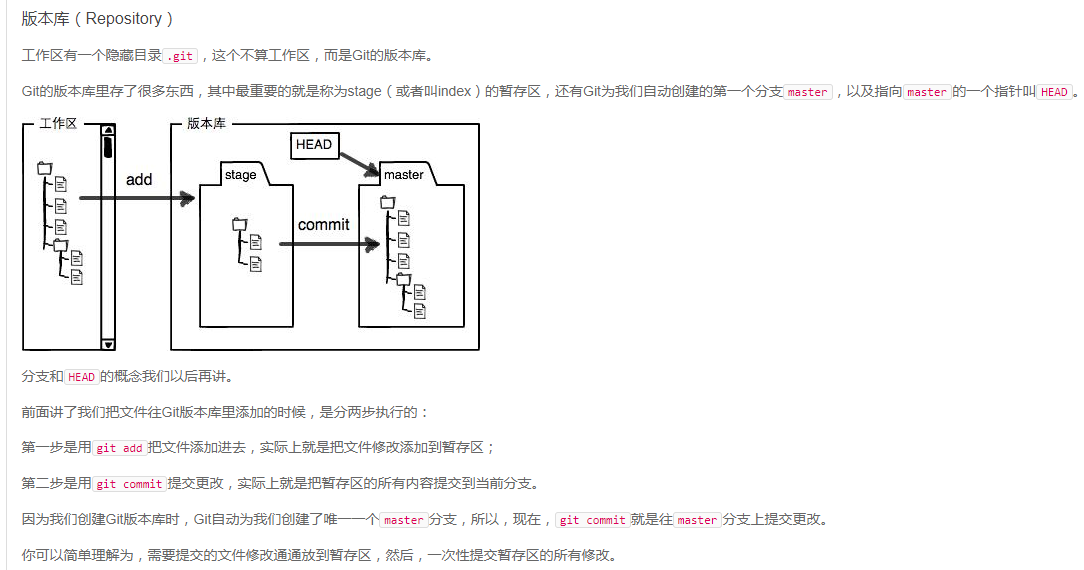
再查看内容，发现又回到了“commit again”版本

#### 小结

* 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



### 工作区和暂存区

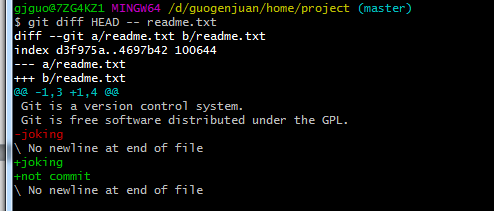


### 管理修改

#### 小结

如果不用git add到暂存区，那就不会加入到commit中。

用git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别



### 撤销修改

#### 小结

* 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>，把暂存区的修改撤销掉（unstage），重新放回工作区，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/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节git reset –hard HEAD^或者commit\_id，不过前提是没有推送到远程库。

### 删除文件

本地库里删除文件后：

一是确实要从版本库中删除该文件，那就用命令git rm filename删掉，并且git commit：文件就从版本库中被删除了。

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本

git checkout -- test.txt

\*\*\*\*\*\*git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

## 远程仓库

### 添加远程仓库

第一步：在github上添加一个仓库

第二步：已有的本地仓库与之关联

在本地的project仓库下运行命令：

$ git remote add origin git@github.com:guogenjuan/home-project.git

第三步：把本地库的内容推送到远程，用git push命令

$ 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。

#### 小结

* 要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；
* 关联后，使用命令git push -u origin master第一次推送master分支的所有内容；
* 此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；
* 分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

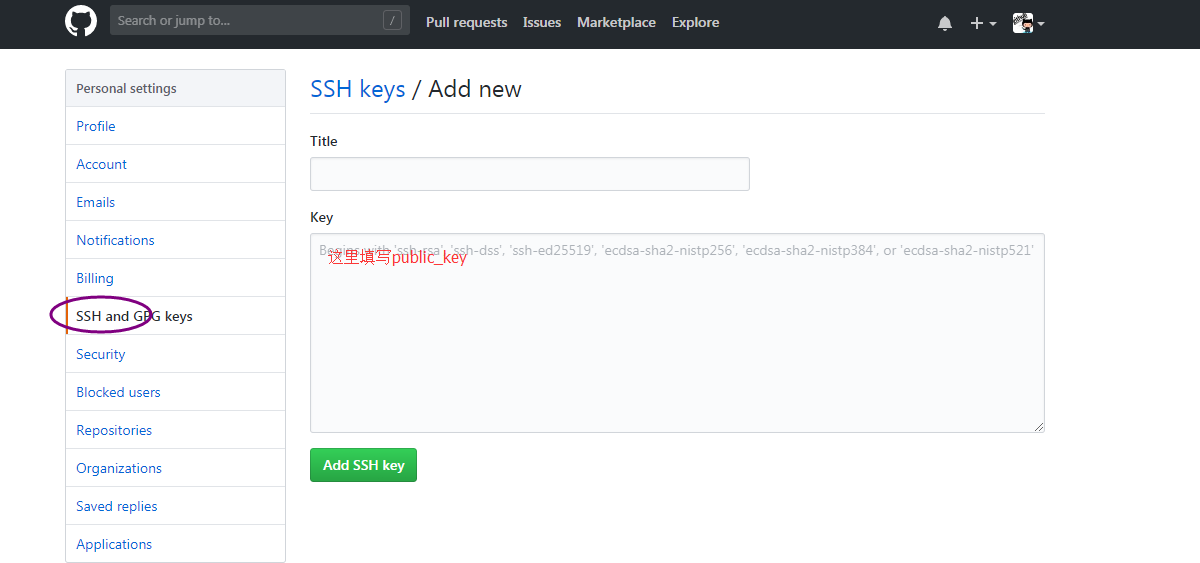
### 配置秘钥公钥

用git clone远程仓库的时候，报错Permission denied (publickey).

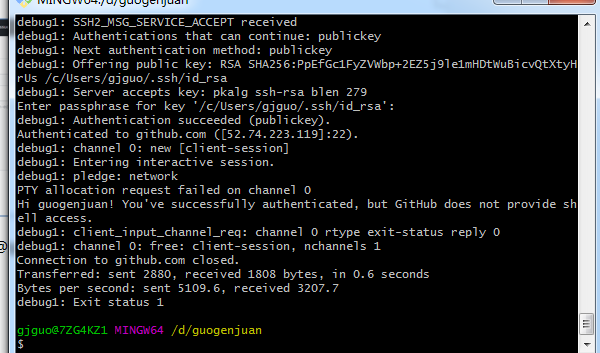
步骤一：生成秘钥公钥

 ssh-keygen -t rsa

步骤二：在github上配置公钥，复制 public key (id\_rsa.pub) 到你账户



步骤三：使用命令 ssh -v git@github.com测试



### 从远程仓库克隆

#### 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。

## 分支管理

### 创建与合并分支

#### 小结

Git鼓励大家使用分支：

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：切换到master下合并，git merge <name>

删除分支：git branch -d <name>

git branch -D <name>强行删除

### 解决冲突

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

Git分支十分强大，在团队开发中应该充分应用。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

### Bug分支

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。

### 多人写作

要在dev分支上开发，就必须创建远程origin的dev分支到本地，于是他用这个命令创建本地dev分支：

$ git checkout -b dev origin/dev

$ git push origin dev ----提交到远程

因此，多人协作的工作模式通常是这样：

1. 首先，可以试图用git push origin <branch-name>推送自己的修改；
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>。

这就是多人协作的工作模式，一旦熟悉了，就非常简单。

#### 小结

* 查看远程库信息，使用git remote -v；
* 本地新建的分支如果不推送到远程，对其他人就是不可见的；
* 从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；
* 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；
* 建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；
* 从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

## 标签

### 创建标签

* 命令git tag <tagname>用于新建一个标签，默认为HEAD，也可以指定一个commit id
* 命令git tag -a <tagname> -m "blablabla..."可以指定标签信息
* 命令git tag可以查看所有标签

### 操作标签

* 命令git push origin <tagname>可以推送一个本地标签
* 命令git push origin --tags可以推送全部未推送过的本地标签
* 命令git tag -d <tagname>可以删除一个本地标签
* 命令git push origin :refs/tags/<tagname>可以删除一个远程标签