https://github.com/yhzh/MyGitRepository.git

https://github.com/yhzh/MyGitRepository..gitgit

# -初始化 git

首先在你选择的硬盘新建一个文件作为库，然后切换到该目录下，开始写命令操作。

------------------------------初始化 git 创建 .git 目录------------------

$ git init

# 配置邮件地址跟自己的用户名才可以提交版本

git config user.email "这个是邮箱"

git config user.name "你的名字"

# 添加到暂存区，才可以commit

----------如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪，然后提交 。git add 命令添加到暂存区中

$ git add fileNmae fileName2 … #可以同时添加个文件

$ git add README

$ git commit -m 'initial project version'

# 查看缓存区状态

$ git status

# 提交

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

git commit只负责把暂存区的修改提交

$ git commit –m “注释语句”

# 显示日志

$ git log --pretty=oneline #单行显示日志

0229947904e8218ba29dc0863110db83bcfeb9f2 3 modify

ded077aa59db8c4218a56a74c870350af5da131b 2 modify

f7dff8c2cc09bc37c42b3c0133f78eff0596e632 b file

682f4356d9d1c1e9e7170e62a5ffc3aea40627be first commt

# 回退版本

用git log可以查看提交历史，以便确定要回退到哪个版本。

用git reflog查看命令历史，以便确定要回到未来的哪个版本。

$ git re**set** HEAD readme.txt #回退单个文件“readme.txt”的版本

上一个版本就是HEAD^，上上一个版本就是HEAD^^

git reset HEAD~10 //回退至10个版本之前, 并重置缓存区

git re**set** --hard HEAD^

再往后倒退

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针

git reset --hard 022994 #版本号没必要写全，git会自动去找。

现在，你回退到了某个版本，关掉了电脑，第二天早上就后悔了，想恢复到新版本怎么办？找不到新版本的commit id怎么办？

在Git中，总是有后悔药可以吃的。当你用$ git reset --hard HEAD^回退到add distributed版本时，再想恢复到append GPL，就必须找到append GPL的commit id。Git提供了一个命令git reflog用来记录你的每一次命令：

$ git reflog

ea34578 HEAD@{0}: reset: moving to HEAD^

3628164 HEAD@{1}: commit: append GPL

ea34578 HEAD@{2}: commit: add distributed

cb926e7 HEAD@{3}: commit (initial): wrote a readme file

# 查看工作区和版本库里面最新版本的区别

$ git diff HEAD –- 文件名

# 撤销工作区的修改

git checkout -- file

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。

git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时

# 暂存区修改撤销掉重新回放回工作区

git reset HEAD file

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

# 删除文件

$ git rm test.txt

恢复到工作区

git reset head fileName

git checkout -- fileName

# 远程仓库

## 生成私钥和公钥

$ ssh-keygen -t rsa -C "310263760@qq.com" #一路按 enter键

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/Administrator/.ssh/id\_rsa):

Created directory '/c/Users/Administrator/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/Administrator/.ssh/id\_rsa.

Your public key has been saved in /c/Users/Administrator/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:Z5s1iuECD1AdH5MuDkE6g2s5AZij7NuQoUKkNL5nIFE 310263760@qq.com

The key's randomart image is:

+---[RSA 2048]----+

|ooE.o...o. |

|\*= o. ...o |

|O+\* . .. |

|+Bo+. . . |

|\*+= oo .S o o |

|+=.o +.. = = . |

|. \* o o + |

| . . . |

| |

+----[SHA256]-----+

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

## 关联本地库跟远程库

一、先有本地库，后有远程库的时候，如何关联远程库

$ git remote add origin <https://github.com/yhzh/gitRepository.git>

## 推送到远程库

第一次推送,先关联远程库

# mygit随意取名字

git remote add origin <https://github.com/yhzh/gitRepository.git>

git remote add mygit <https://github.com/yhzh/gitRepository.git>

git push -u mygit master

第二次推送，不用写参数

git push mygit master

git push -u gitRemoteReposiry master

## 取消本地目录下关联的远程库：

$ git remote remove origin

## 查看有没有关联的远程库

$ git remote -v

## 从远程库克隆到本地

git clone https://github.com/yhzh/gitRepository.git

## 把最新代码更新到本地库

$ git pull

## git 协议

Git 支持许多数据传输协议。之前的例子使用的是 git:// 协议，不过你也可以用 http(s):// 或者 user@server:/path.git 表示的 SSH 传输协议。

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。

# 创建管理分支

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>

删除分支：git branch -d <name>

## 创建分支

$ git checkout –b test #创建并切换分支到 test

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch test

$ git checkout test

## 合并分支

把test分支的工作成果合并到master分支上

先切换回主分支master,然后执行合并。

$ git checkout master

$ git merge test

删除分支

$ git branch -d test #合并完后就可以放心的删除分支了。

用git log --graph命令可以看到分支合并图。

合并冲突，会把各自分支的内容合并然后得手动解决，在提交。

## 分支管理策略

本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去

合并 a 分支 --no-ff参数，表示禁用Fast forward

git merge --no-ff -m "with no-ff" a

**查看分支历史**

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

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 branch-name origin/branch-name。

# 标签管理

注意，标签不是按时间顺序列出，而是按字母排序的

## 打标签

$ git tag v1.0

## 查看标签

$ git tag

## 删除标签

创建的标签都只存储在本地，不会自动推送到远程

$ git tag -d v1.0

---------如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪，然后提交：

--------如果想对某个开源项目出一份力，可以先把该项目的 Git 仓库复制一份出来，这就需要用到 git clone 命令。如果你熟悉其他的

--------VCS 比如 Subversion，你可能已经注意到这里使用的是 clone 而不是 checkout。

--------克隆库，语法 命令格式为 git clone [url]--

$ git clone git://github.com/schacon/grit.git

git@github.com/yhzh/gitRepository.git

------------------------

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。

-------

之后操作提交就一句话

$ git push -u origin master ------第一次推送加上了 -u 参数

$ git push origin master ------之后推送就不用加了

------------------添加项目到远程仓库总结

-----------提交异常的情况下，估计是本地跟远程版本冲突，解决办法

$ git pull origin master ---把远程跟本地合并

$ git push origin master ---然后在推送到远程

--------跳过使用暂存区域

git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交

----------查看提交历史

$ git log

$ git log -p -2 ------查看最近两次的更新

-----------提交操作

有时候我们提交完了才发现漏掉了几个文件没有加，或者提交信息写错了。想要撤消刚才的提交操作，可以使用 --amend 选项重新提交

$ git commit --amend ----撤销操作