

***Package na***

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

### github提速访问

#### 加速方法一

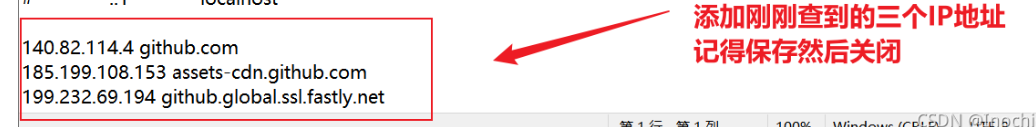
<https://blog.csdn.net/Inochigohan/article/details/120190887>

* 随便打开一个浏览器，在搜索栏上输入域名解析网址：The Best IP Address, Email and Networking Tools - IPAddress.com，<https://www.ipaddress.com/>

分别查询① github.com、② assets-cdn.github.com 、③ github.global.ssl.fastly.net三个链接的DNS解析地址，找到解析出来的IP地址。

* 步骤二：修改电脑hosts文件

把刚刚查询到的三个网站解析出来的IP地址添加到**C:\Windows\System32\drivers\etc**下的hosts文件（如果hosts文件没有更改权限的话，可以先用记事本以管理员身份运行，再打开该文件）。



192.168.9.1 windows10.microdone.cn

140.82.114.3 github.com

185.199.108.153 assets-cdn.github.com

199.232.69.194 github.global.ssl.fastly.net

以上的可以参考

* 步骤三：刷新本机的DNS缓存

输入命令：ipconfig /flushdns

#### 加速方法二

（1）查询github.com的最优DNS解析地址：在浏览器中打开DNS查询网站；我们输入github.com，点击检测，在下方的结果列表中，选择TTL值小的IP地址52.192.72.89，如下图所示：

<https://tool.chinaz.com/dns>



(2)接着打开本地电脑的**C:\Windows\System32\drivers\etc**目录，找到hosts文件，利用记事本打开，如下图所示：

(3) 在hosts文件的末尾一行添加上述步骤中查到的TTL值小的IP地址52.192.72.89，添加内容为:52.192.72.89 github.com

在这里插入图片描述

在这里插入图片描述

(4)点击文件-》保存，win10若遇到文件不能保存需要权限问题，使用下图的方法解决。



* 步骤三：刷新本机的DNS缓存

输入命令：ipconfig /flushdns

#### 加速方法三

<https://blog.csdn.net/m0_62197295/article/details/123598265>

GitHub 开源项目

FastGithub

<https://github.com/dotnetcore/FastGithub>

值得注意的是

### Git的安装

解决官网下载很慢问题，下载传送门 <https://npm.taobao.org/mirrors/git-for-windows/>

### 基础

工作区

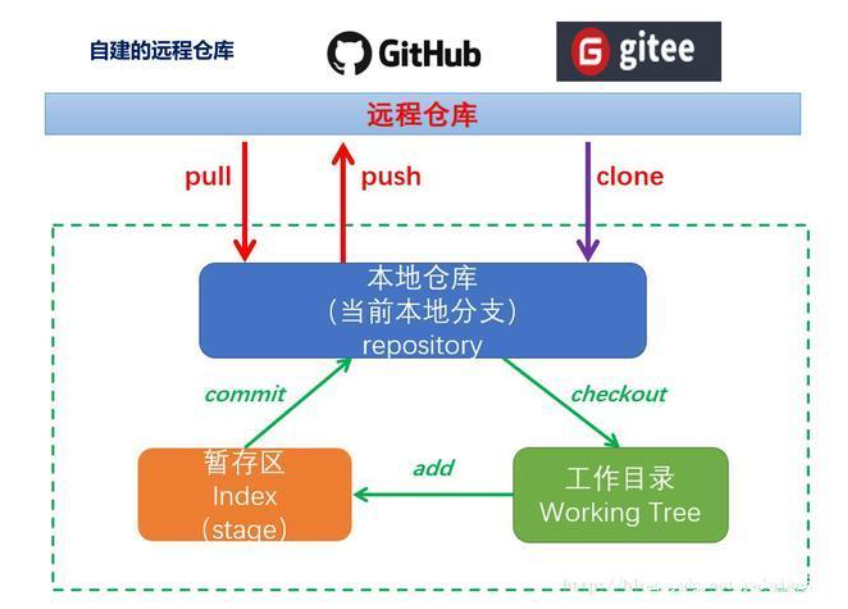
文件夹中除.git文件夹之外的都属于工作区，我们在工作区内进行增删改等操作。

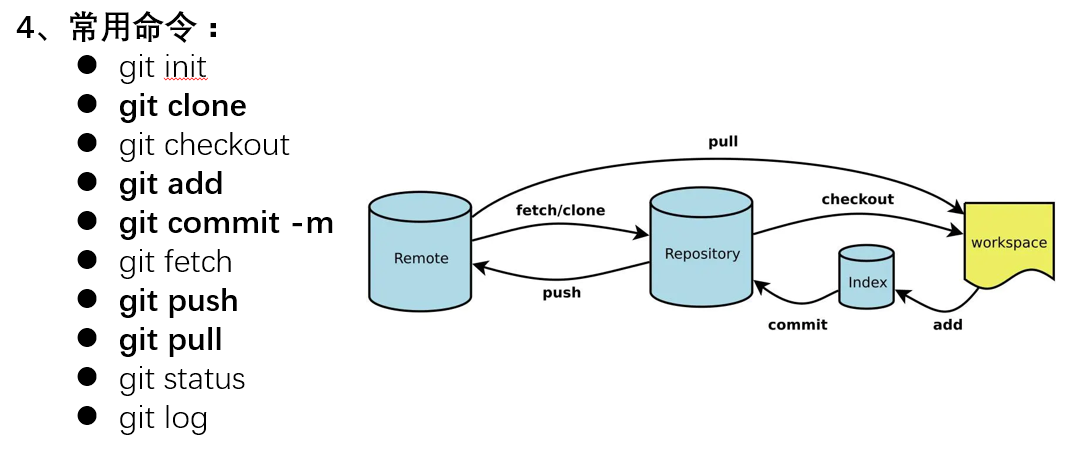
暂存区

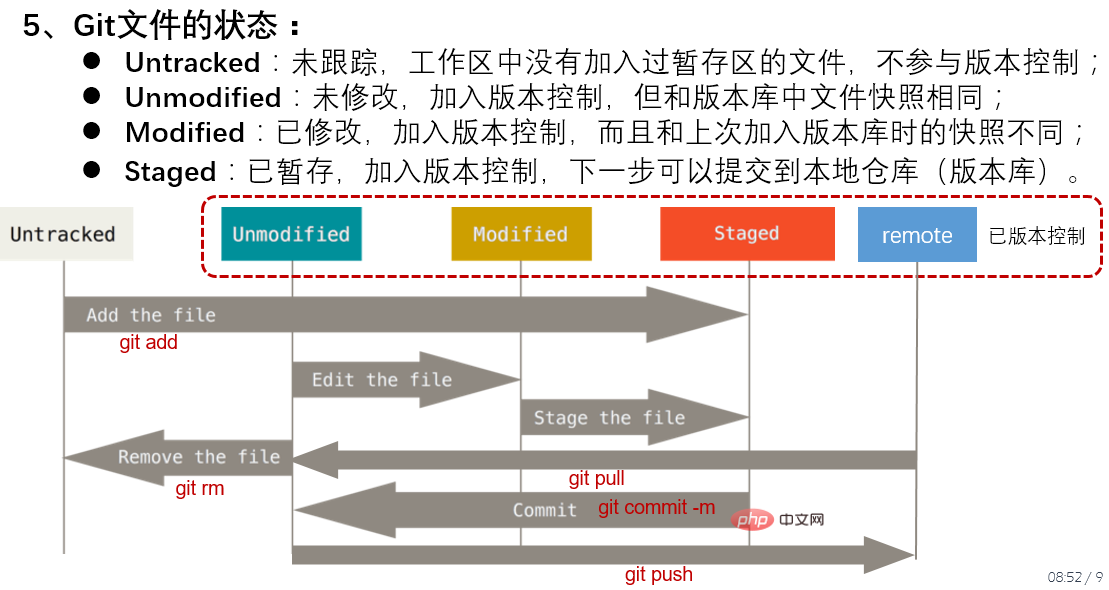
.git文件夹中的一部分空间，暂存区用来暂时存放被更改了的需要提交为新版本的文件。

版本库

.git 文件夹中除暂存区以外的空间。各个版本在版本库中以commita的时间为线索构成链表，每个版本的哈希值为该版本的版本号，提交时通过-m写入的备注就是该版本的版本说明。





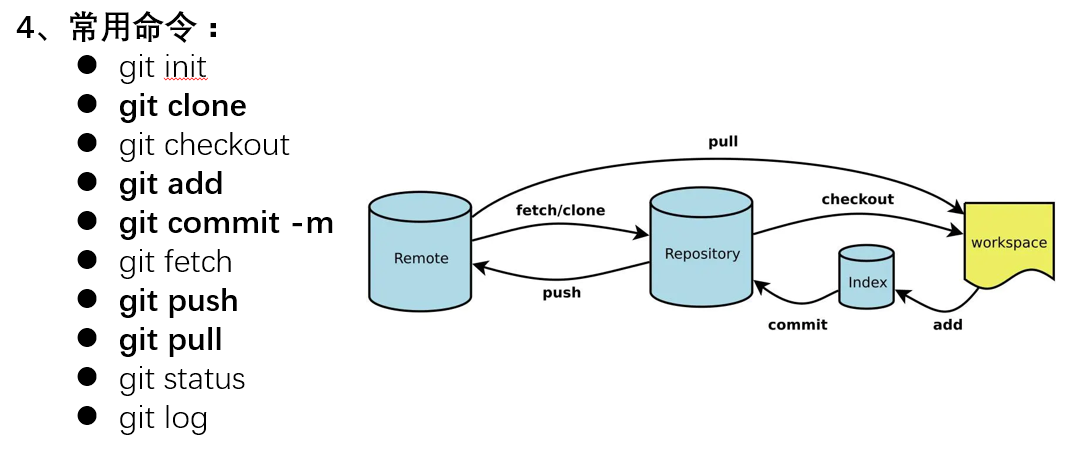
****

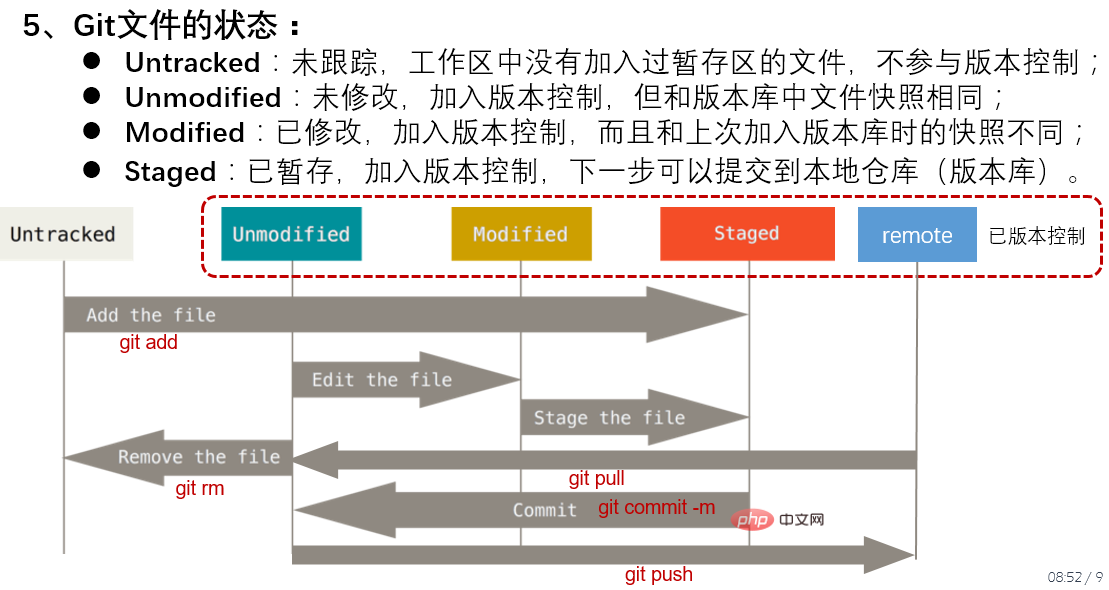


测试demo

<http://192.168.10.249:8888/mingth/demo.git>

### 常见命令



****

#### git 分支管理

##### 查看分支列表

git branch 分支前面的 \* 表示当前所处的分支。

##### 创建新分支

git branch 分支名称

##### 切换分支

git checkout 分支名称

##### 分支的快速创建与切换

git checkout -b 分支名称

##### 合并分支

功能分支的代码开发测试完毕之后，可以使用如下命令，将完成后的代码合并到master主分支上：

git checkout master

git merge 分支名称

##### 删除分支

当把功能分支的代码合并到master主分支上以后，就可以使用如下命令，删除对应的功能分支：

git branch -d 分支名称

##### 将本地分支推送到远程仓库

如果是第一次将本地分支推送到远程仓库，需要执行如下的命令：

|  |
| --- |
| git push -u 远程仓库的别名 本地分支名称:远程分支名称  -u表示把本地分支和远程分支进行关联，远程仓库的别名一般是origin。  如果不是第一次将本地分支推送到远程仓库，需要执行如下命令：  git push |

##### 查看远程仓库的所有分支列表

|  |
| --- |
| git remote show 远程仓库别名 |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

#### git clone 从云上克隆项目到本地

|  |
| --- |
| http://192.168.10.249:8888/aicam/aicam.git // 公开，开发人员：徐志远  http://192.168.10.249:8888/aicam/core.git // 私有，仅管理员可操作  http://192.168.10.249:8888/aicam/static.git // 公开，开发人员：梁钢城  http://192.168.10.249:8888/aicam/algorithm.git // 公开，开发人员：徐志远、卢斯、冯雷、汪浩、白美林  http://192.168.10.249:8888/aicam/model-so.git // 私有，开发人员：徐志远、卢斯、冯雷 |

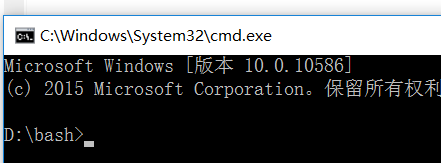
通常情况下部分git链接中是带有权限的，就是特定的链接才能够进行访问。

同时同步如上链接时可以使用cmd命令行，



使用git自带的bash容易出现错误，主要是字符和window之间的不兼容。



可以使用如上进行打开cmd命令，

|  |
| --- |
| D:\bash>git clone http://192.168.10.249:8888/aicam/algorithm.git  Cloning into 'algorithm'...  Username for 'http://192.168.10.249:8888': wanghao  Password for 'http://wanghao@192.168.10.249:8888':  remote: Counting objects: 184, done.  remote: Compressing objects: 100% (104/104), done.  Receiving objects: 100% (184/184), 85.14 MiB | 11.28 MiB/s, done. Total 184 (delta 78), reused 184 (delta 78)  Resolving deltas: 100% (78/78), done. |



进入到demo文件夹内有git文件夹，这个就是git工程的项目文件。

#### git init 初始化仓库

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 /d/svn共享文件/工业机械臂/ROS工业机械臂/AI+智能分拣实训平台\_probot\_hera/实验代码/01-平台代码  **$ git init**  Initialized empty Git repository in D:/svn共享文件/工业机械臂/ROS工业机械臂/AI+智能分拣实训平台\_probot\_hera/实验代码**/01-平台代码/.git/**  汪浩@DESKTOP-KQMGM9O MINGW64 /d/svn共享文件/工业机械臂/ROS工业机械臂/AI+智能分拣实训平台\_probot\_hera/实验代码/01-平台代码 (master) |

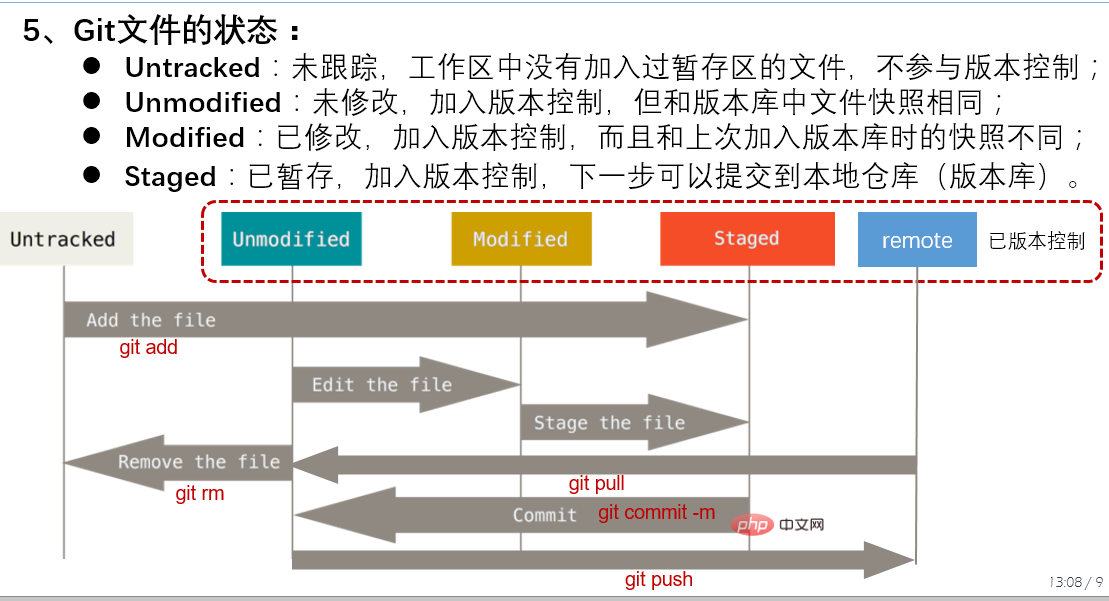
在01-平台代码文件夹内创建.git文件夹

**$ ls -a**

./ ../ .git/ src/

可以查看到创建的git仓库

#### git status -s 对状态的跟踪(简短的输出结果)



git中有两个状态：内容状态和文件状态，

内容状态标示内容文件的改变，有三个区域：工作目录，暂存区和提交区

文件状态有两个状态：已跟踪和未跟踪

git status 命令用于查看项目的当前状态。

|  |
| --- |
| $ **git status**  On branch master  Your branch is up to date with 'origin/master'.  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git restore <file>..." to discard changes in working directory)  modified: src/base64tool.py |

当我们修改过src/base64tool.py文件时这里就会提示修改过

|  |
| --- |
| $ **git status -s**  M src/base64tool.py |

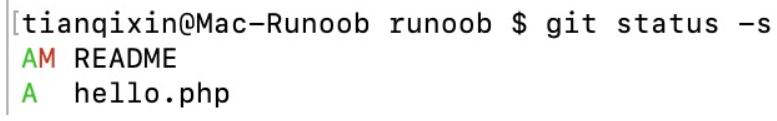
显示具体的修改信息，M就表示已修改。

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ **git add src/base64tool.py**  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ **git status -s**  M src/base64tool.py |

这里的M就变成绿色

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  **$ notepad++ src/base64tool.py**  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git status -s  MM src/base64tool.py |

AM 状态的意思是这个文件在我们将它添加到缓存之后又有改动。有的版本

版本之间提示不一样，有的版本用A表示已经添加到暂存。

|  |
| --- |
|  |

|  |
| --- |
|  |

#### git add 将该文件添加到暂存区。

|  |
| --- |
| git add [file1] [file2] ... //添加一个或多个文件到暂存区：  git add [dir] //添加指定目录到暂存区，包括子目录：  git add . //添加所有文件 |

删除暂存区文件：**git rm -- cached 文件名**

**删除之后，只是暂存器文件没了，即git没法追踪到这个文件了，原本文件还在**。

一、查看文件

git status -s

二、删除文件

git rm --cached （文件路径）只删除暂存区文件

git rm --f（文件路径）删除暂存和物理文件

|  |
| --- |
|  |

#### git ls-files 查看暂存区中文件信息

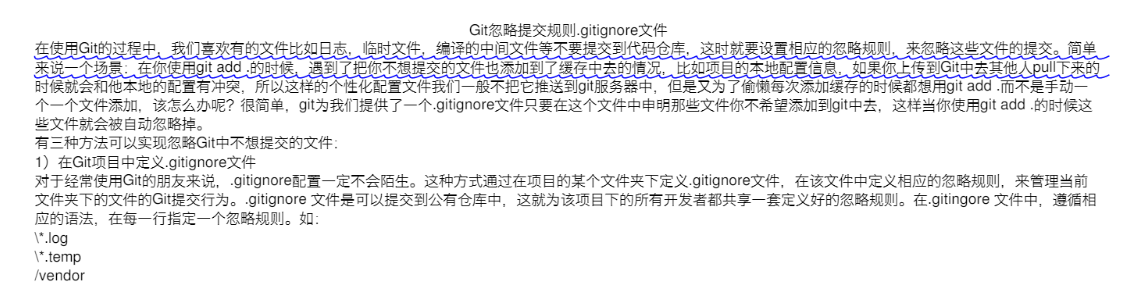
|  |
| --- |
|  |

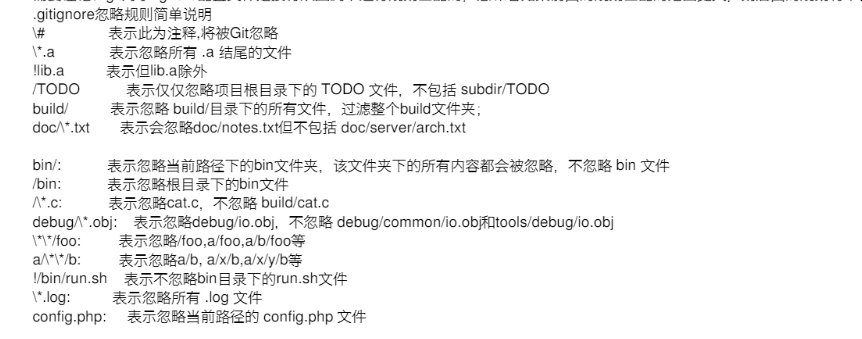
#### git rm -r --cached \* 删除暂存区所有文件

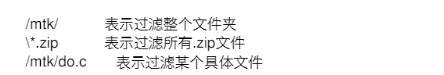
git rm -r --cached \*

git rm -r --cached ./

#### git 提交文件规则







新建

3.注意：若是你不慎在建立.gitignore文件以前就push了项目，那么即便你在.gitignore文件中写入新的过滤规则，这些规则也不会起做用，Git仍然会对全部文件进行版本管理。

简单来讲出现这种问题的缘由就是Git已经开始管理这些文件了，因此你没法再经过过滤规则过滤它们。

因此你们必定要养成在项目开始就建立.gitignore文件的习惯，不然一单push，处理起来会很是麻烦。

**可以先删除所有的暂存区文件，然后再来按照规则进行添加即可。**

使用

**git rm -r --cached \***

**git rm -r --cached ./**

规则示例：

\*/zonesion/common/

不提交所有工程中的common目录、

#### git commit -m "备注" 从add的暂存区提交到本地仓库，并备注信息

commit操作的对象是暂存区内的文件，通过add把文件加入暂存区后，利用commit操作暂存区内的更改作为新版本加入版本库中，版本指针指向该新版本

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git commit -m "v1.0"  Author identity unknown  \*\*\* Please tell me who you are.  Run  git config --global user.email "you@example.com"  git config --global user.name "Your Name" |

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git commit -m "v1.0"  [master da0b171] v1.0  1 file changed, 3 insertions(+) |

前面章节我们使用 git add 命令将内容写入暂存区。git commit 命令将暂存区内容添加到本地仓库中。

**提交暂存区到本地仓库中:**

**$ git commit -m [message]**

[message] 可以是一些备注信息。

**提交暂存区的指定文件到仓库区**：

**$ git commit [file1] [file2] ... -m [message]**

git commit -s -m "descriptions about the code"

只要加入-s参数即可自动加上Signed-off-by信息

#### git commit -am "备注" 从工作区提交到本地仓库，并备注信息, -s参数即可自动加上Signed-off-by信息

-a 参数设置修改文件后不需要执行 git add 命令，直接来提交

|  |
| --- |
| 再次之前我们在src文件夹中添加了一个test。txt文件，  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git commit -am "v1.2"  [master 6d69f38] v1.2  1 file changed, 2 insertions(+), 1 deletion(-)  可见此命令并没有吧所有的文件都提交到本地仓库中  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git ls-files  src/base64tool.py  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git add .  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git ls-files  src/base64tool.py  src/test.txt  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ git commit -am "v1.2"  [master 76ca4a2] v1.2  1 file changed, 1 insertion(+)  create mode 100644 src/test.txt  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ |

#### git reflog 查看本地仓库精简版本信息

#以下两个命令都可以

git reflog #查看精简版本信息

git log #查看详细版本信息

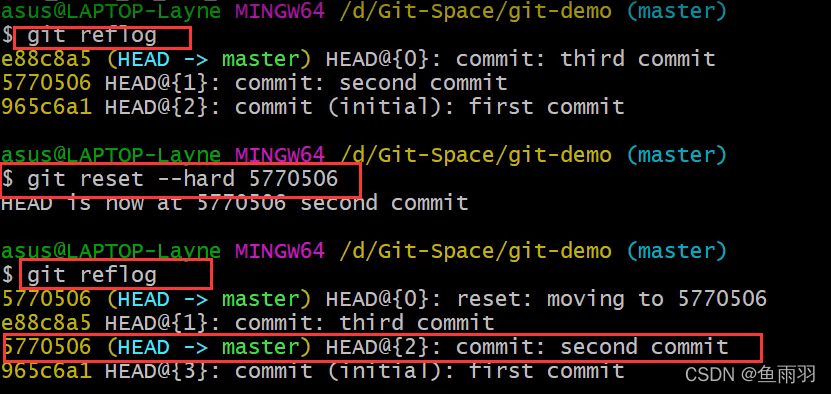
|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ ***git reflog***  100db4c (HEAD -> master) HEAD@{0}: commit: v1.1  da0b171 HEAD@{1}: commit: v1.0  35ec0cb (origin/master, origin/HEAD) HEAD@{2}: pull: Fast-forward  c395cbb HEAD@{3}: clone: from http://192.168.10.249:8888/mingth/demo.git |

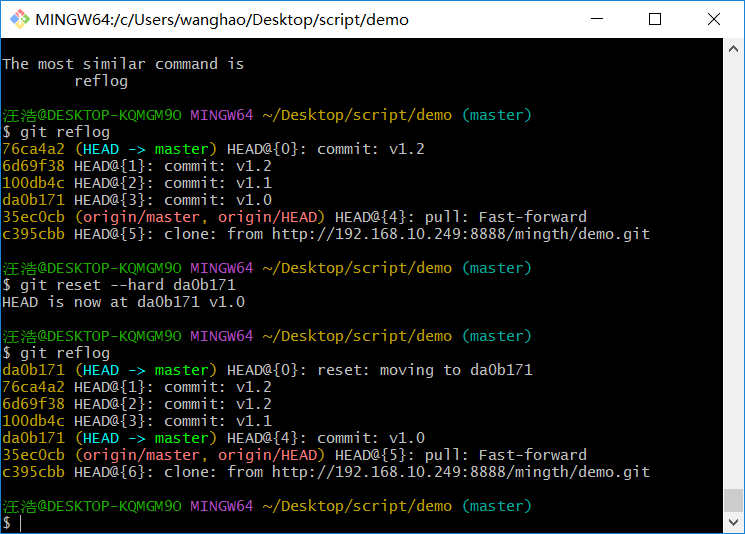
以上就是我之前提交的两个版本，

#### git reset --hard 版本号版本穿梭，回到以前版本

下面蛋黄色就是版本号

git控制切换版本，实质是通过调用不同的指针指向不同的版本。

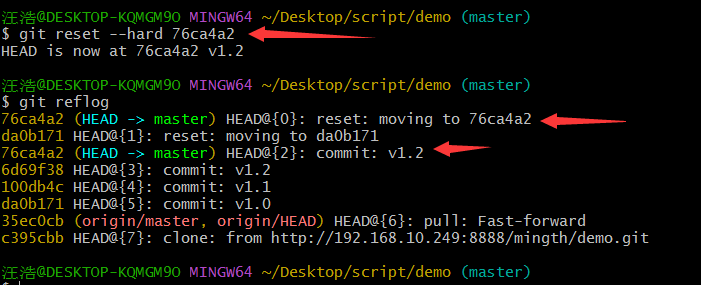




如上，当前的版本从版本号76ca4a2 回到 da0b171

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ cd src/  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo/src (master)  $ ls  base64tool.py |

此时可以看见工作名录中没有之前添加进去的test.txt文件。



当我们重新回到之前v1.2的版本时，这里的HEAD{0}(当前指针)重新指向76ca4a2这个版本。并且每一次操作都会生成相应你的日志信息，并且更新当前HEAD指针指向的版本号。带有(HEAD -> master)信息的其实就是一个版本

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  $ cd src/  汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo/src (master)  $ ls  base64tool.py test.txt |

可见test.txt文件回来了

#### git 删除提交的某一个版本

<https://blog.csdn.net/weixin_45965246/article/details/120898566>

|  |
| --- |
| 1. git rebase -i <想要删除的commit id>  2. git rebase -i HEAD~3 #数字代表需要删除的个数，顺序从后往前排列，那么根据假设我们应该输入HEAD~1才能到达commit,输入HEAD~2只会显示commit3和commit2。 |

保存退出之后就直接删除了，现在只剩下了commit3.txt的提交版本，并输入以下命令更新远程仓库

git push --force

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

#### git fetch 和 git merge

假设你配置好了一个远程仓库，并且你想要提取更新的数据，你可以首先执行:

|  |
| --- |
| **$ git fetch origin**  remote: Counting objects: 3, done.  remote: Compressing objects: 100% (2/2), done.  remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  Unpacking objects: 100% (3/3), done.  From github.com:tianqixin/runoob-git-test  0205aab..febd8ed master -> origin/master |

以上信息"0205aab..febd8ed master -> origin/master" 说明 master 分支已被更新，我们可以使用以下命令将更新同步到本地：

|  |
| --- |
| **$ git merge origin/master**  Updating 0205aab..febd8ed  Fast-forward  README.md | 1 +  1 file changed, 1 insertion(+) |

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  **$ git merge origin/master**  Auto-merging src/base64tool.py  Merge made by the 'ort' strategy.  src/base64tool.py | 4 +++-  src/fenglei\_test.txt | 1 +  2 files changed, 4 insertions(+), 1 deletion(-)  create mode 100644 src/fenglei\_test.txt |

|  |
| --- |
| 汪浩@DESKTOP-KQMGM9O MINGW64 ~/Desktop/script/demo (master)  **$ git merge FETCH\_HEAD**  Already up to date. |

|  |
| --- |
|  |

##### FETCH\_HEAD解释

在.git目录中会有一个名为FETCH\_HEAD的文件；

每一个执行过fetch操作的项目都会存在一个FETCH\_HEAD列表，其中每一行对应于远程服务器的一个分支。

也就是说这个文件保存了远程仓库中每个分支的FETCH\_HEAD。这个FETCH\_HEAD指向了该分支最近的一次commit；

其实跟HEAD是一回事，只是这个HEAD是指向的远程仓库某个分支的最新的一个commit；

这样我们就知道了执行git fetch的本质就是更新远程仓库分支的commit记录，并且更新FETCH\_HEAD，保证其指向该分支最近的一次commit；

git fetch origin branch1 //如果指定了某个远程分支，那就更新这个分支的commit记录及FETCH\_HEAD；

//如果不加分支名的话，更新该仓库的所有分支的commit记录以及FETCH\_HEAD；

顺便说一下HEAD的概念：

HEAD实际上就是一个指向某个commit的引用；表示的是你当前指向的记录；

默认情况下HEAD指向的是这个分支名，分支名永远指向该分支的最近一次commit；

也就是说默认情况下HEAD指向分支的最近一次提交；

HEAD -> master -> C1

当你进行一次新的commit时，HEAD就会从此前指向的commit往后产生该新的commit，并指向该新的commit；

这里引入一个新的概念DETACHED\_HEAD，即分离的HEAD：

分离的HEAD指向的是一个commit，而不是指向分支名；

Git学习系列15 HEAD ORIG\_HEAD FETCH\_HEAD detached HEAD：https://blog.csdn.net/luoshabugui/article/details/85256509 //这几个head讲解得很好

HEAD、master 与 branch: //对于HEAD和分支的解释很详细：https://www.jianshu.com/p/4219b6f62ce3

git fetch、git pull和FETCH\_HEAD:https://www.cnblogs.com/Venom/p/5477367.html

#### git pull 命令

git pull 命令用于从远程获取代码并合并本地的版本。

git pull 其实就是 git fetch 和 git merge FETCH\_HEAD 的简写。

更新操作：

|  |
| --- |
| $ git pull  $ git pull origin |

将远程主机 origin 的 master 分支拉取过来，与本地的 brantest 分支合并。

|  |
| --- |
| git pull origin master:brantest |

如果远程分支是与当前分支合并，则冒号后面的部分可以省略。

|  |
| --- |
| git pull origin master |

#### git remote 命令

##### git remote add

git remote add [shortname] [url]

shortname 为本地的版本库，例如：

|  |
| --- |
| # 提交到 Github  $ git remote add origin git@github.com:tianqixin/runoob-git-test.git  $ git push -u origin master |

##### git remote –v

2.git remote -v | --verbose 列出详细信息，在每一个名字后面列出其远程url，此时， -v 选项(译注:此为 –verbose 的简写,取首字母),显示对应的克隆地址。

|  |
| --- |
| wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  $ git remote -v  origin https://github.com/w809291634/aiarm.git (fetch)  origin https://github.com/w809291634/aiarm.git (push) |

其他相关命令：

##### git remote rm name # 删除远程仓库

##### git remote rename old\_name new\_name # 修改仓库名

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

#### git push 命令

git push 命用于从将本地的分支版本上传到远程并合并。

命令格式如下：

git push <远程主机名> <本地分支名>:<远程分支名>

如果本地分支名与远程分支名相同，则可以省略冒号：

**git push <远程主机名> <本地分支名>**

这里的**origin就相当于http://192.168.10.249:8888/mingth/demo.git**

|  |
| --- |
| $ **git push origin master**  To http://192.168.10.249:8888/mingth/demo.git  ! [rejected] master -> master (fetch first)  error: failed to push some refs to 'http://192.168.10.249:8888/mingth/demo.git'  hint: Updates were rejected because the remote contains work that you do  hint: not have locally. This is usually caused by another repository pushing  hint: to the same ref. You may want to first integrate the remote changes  hint: (e.g., 'git pull ...') before pushing again.  hint: See the 'Note about fast-forwards' in 'git push --help' for details. |

|  |
| --- |
|  |

其实，执行添加了-u 参数的命令 git push -u origin master就相当于是执行了

git push origin master 和

git branch --set-upstream master origin/master。

所以，在进行推送代码到远端分支，且之后希望持续向该远程分支推送，则可以在推送命令中添加 -u 参数，简化之后的推送命令输入。

————————————————

版权声明：本文为CSDN博主「Lakers2015」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/Lakers2015/article/details/111318801

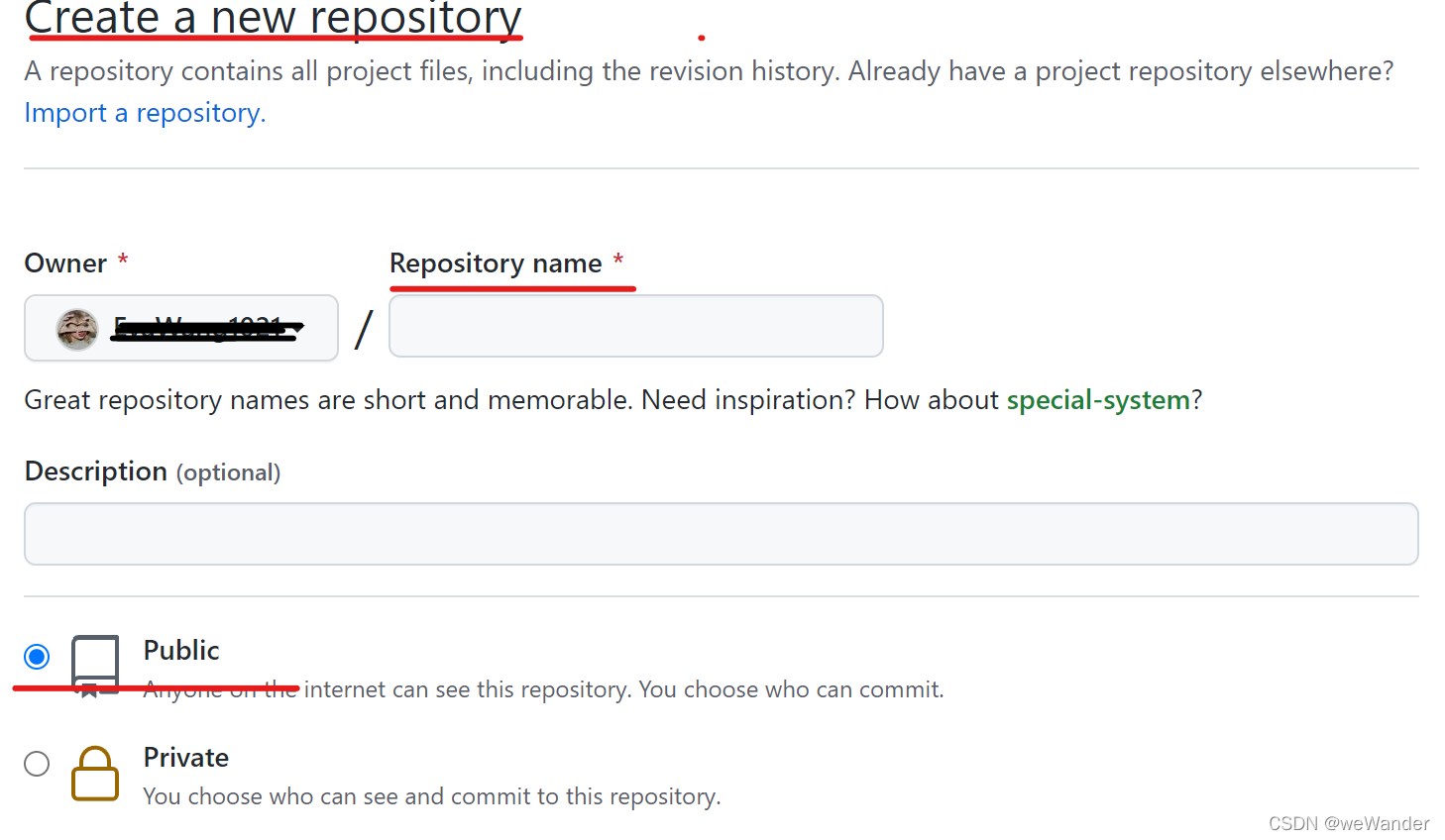
#### git 上传到github

<https://blog.csdn.net/weWander/article/details/122072325> 某一

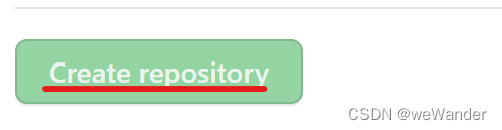
<https://www.runoob.com/git/git-remote-repo.html> 菜鸟联盟

##### 将自己本地的一个完成项目推送到github

###### 二、Github注册和创建仓库（PS：认识repository是仓库的意思就可啦）



完成上述后，点击创建即可



###### 三、接下来就是本地代码提交到github的操作了，根据我自己的理解画了个图，建议你学完也试试，记住得快一点

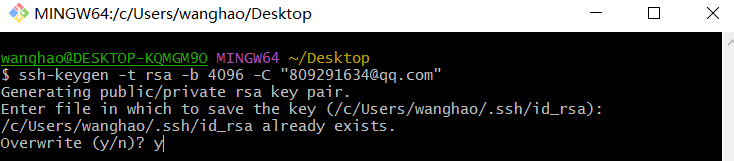


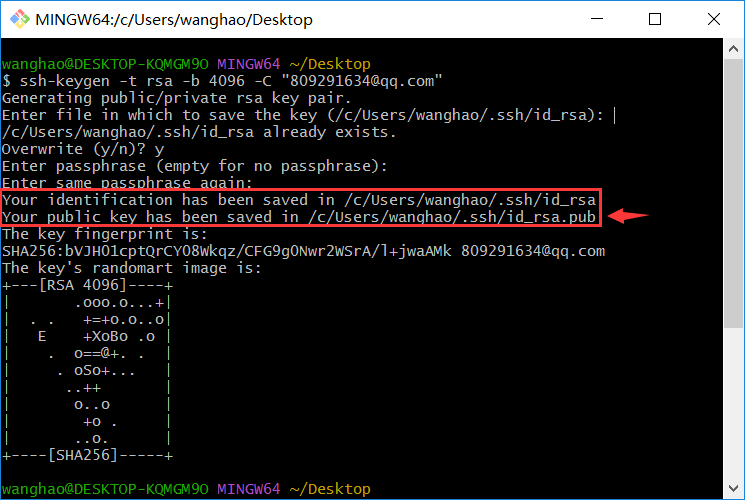
1、github和本地连接

1. 执行如下命令

$ ssh-keygen -t rsa -b 4096 -C "你的邮箱"

随便找个位置输入如上命令

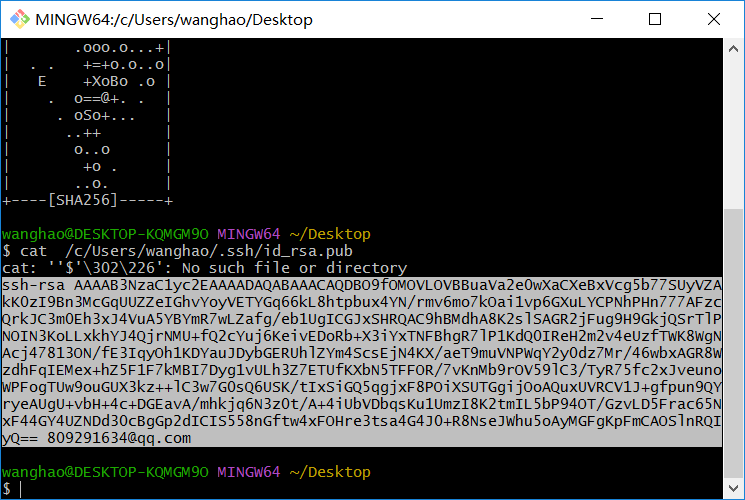




如果显示overwrite提示，表示覆盖，输入y其他的都是enter

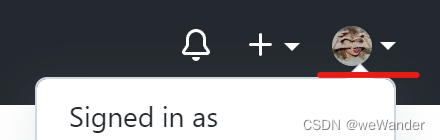
于是生成的key文件存放在/c/Users/wanghao/.ssh/id\_rsa.pub路径

这里我们读取这个文件

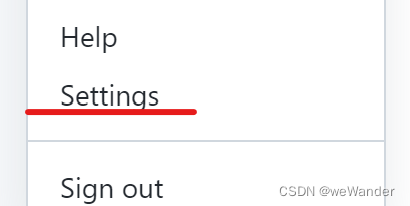
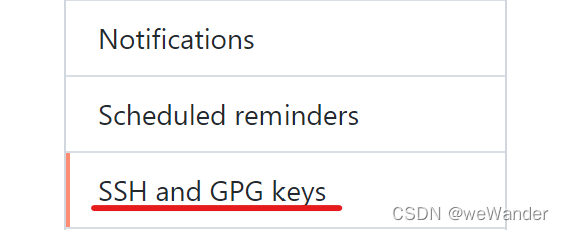


高亮部分就是下面用到的

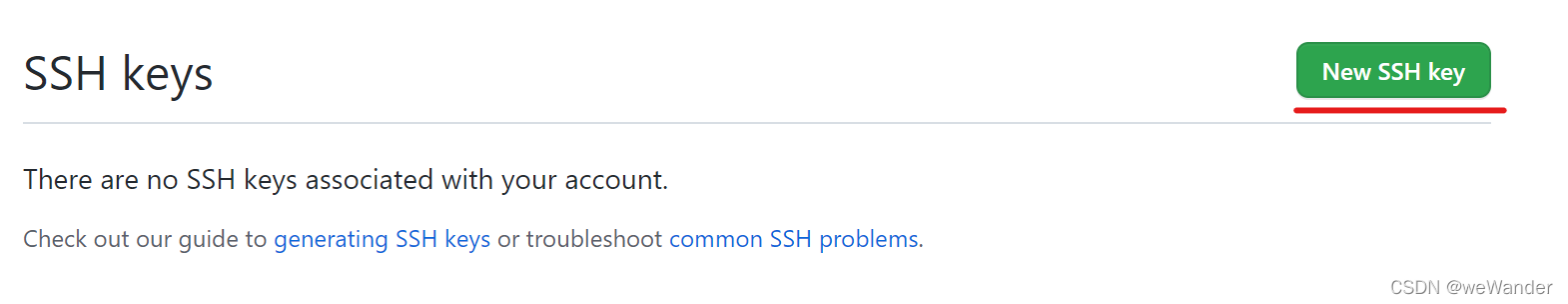
2、转到github上，点击右上角你的头像部分；

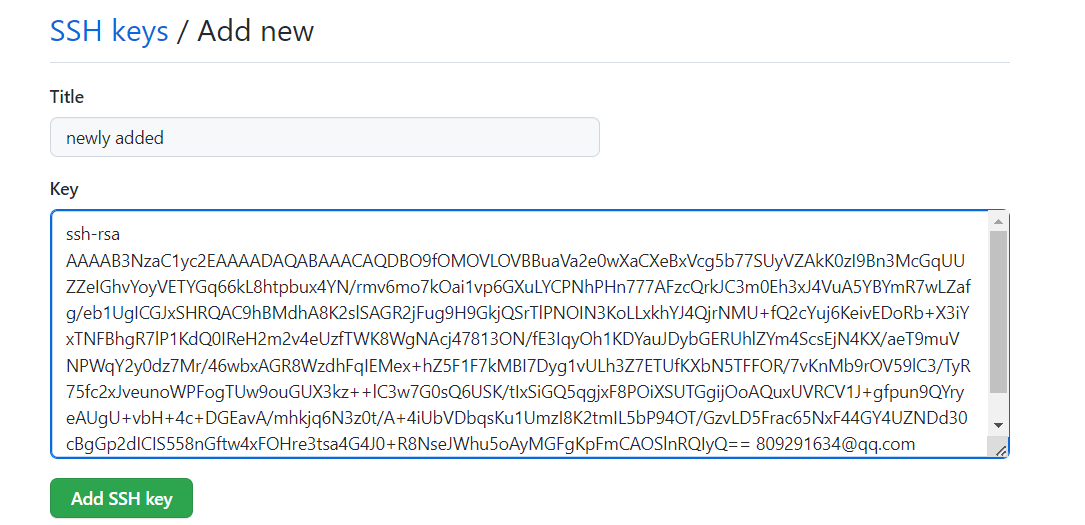


出现下拉框，选择setting，接着继续选择SSH and GPG keys；

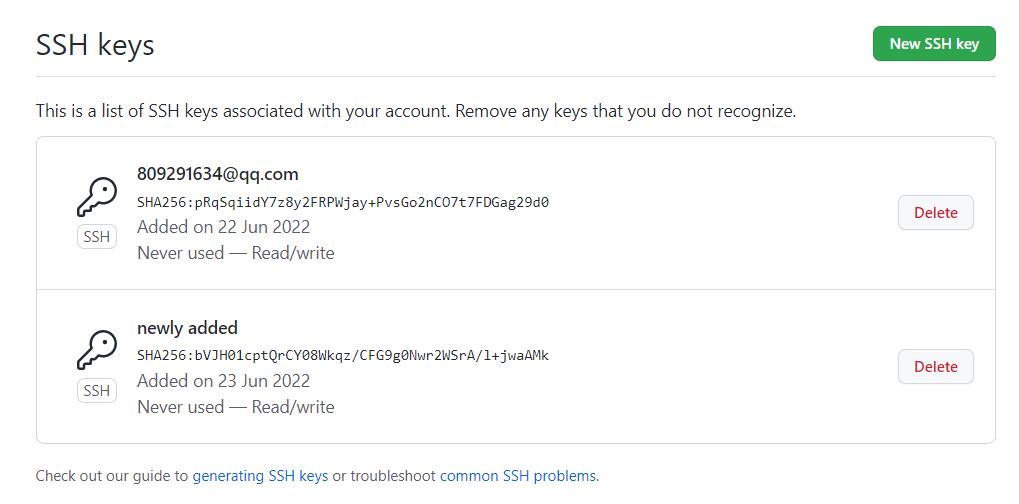
 

点击Add new keys

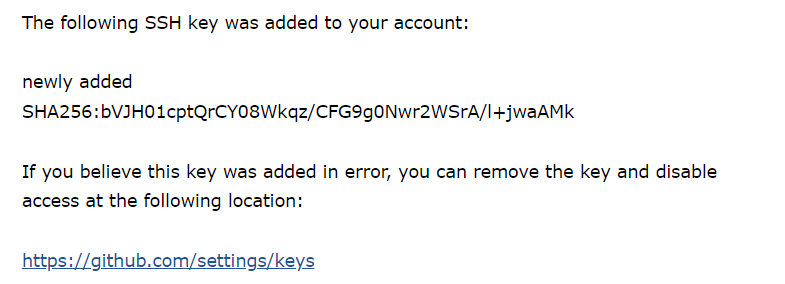




title可以不需要填，这里随意



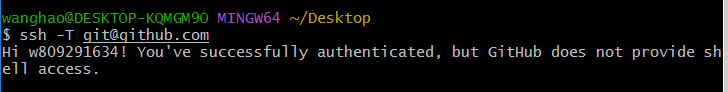
添加完成后，



接着GitHub会像你的邮箱发送一封邮件，

3、接着连接github和本地仓库，在本地终端命令行输入命令

ssh -T git@github.com



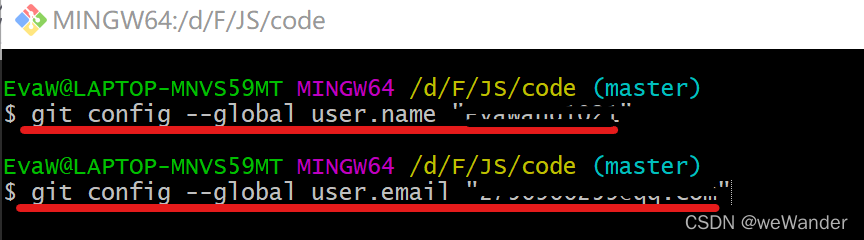
根据提示完成连接，显示successfully则表示连接到github成功

###### 四、推送到远程仓库

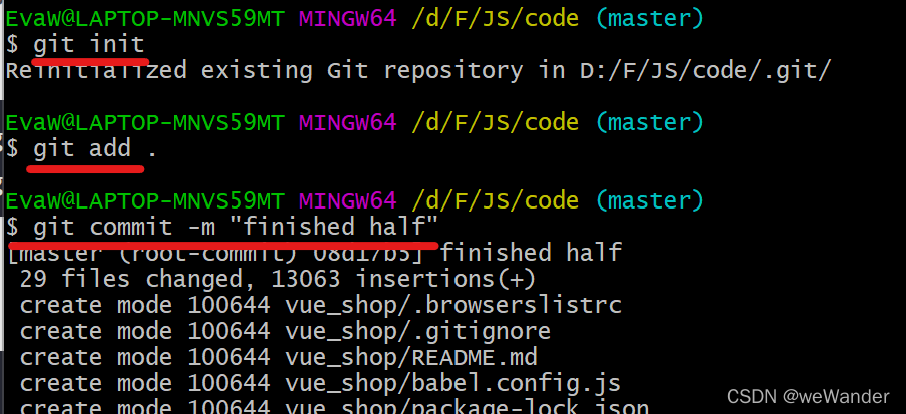
1. 推送本地文件到github，比如你想把文件夹A推送上去，那就在文件夹A所在路径，右击，”git bash”



然后敲以下两行代码，设置用户，作为git的基础配置,作用是告诉git你是谁,你输入的信息将出现在你创建的提交中.（这个只要下载过git，配置一次就可以永久使用啦）



接着初始化本地仓库，将数据添加到本地仓库中，”finished half”（我自己 起的名字）为本次提交的内容，

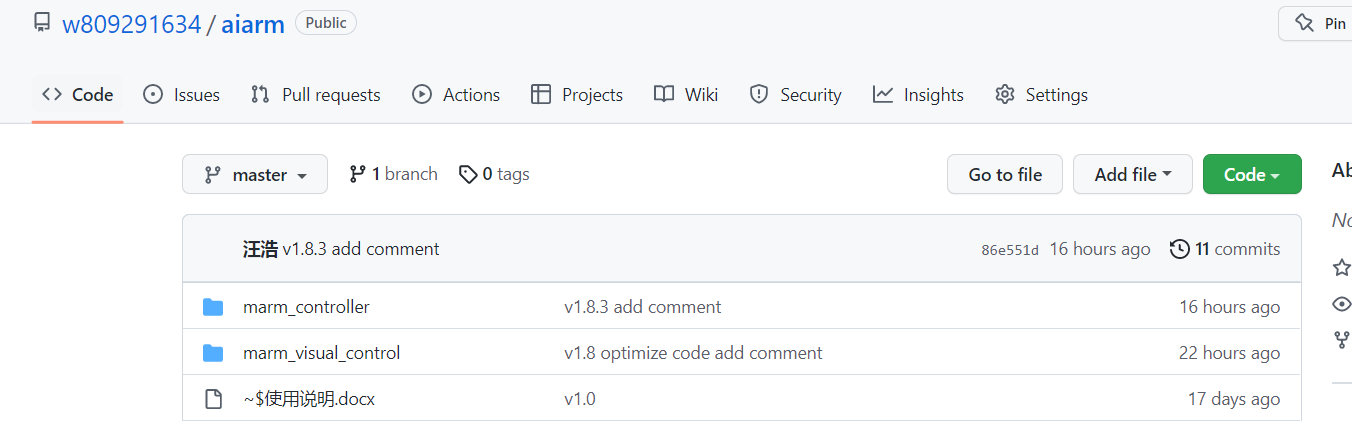


2. 更改本地分支名（可以不做）

**因为2020年10月后，github将所有新建的reposity的默认分支名master改为main**。所以我们需要新建分支并将当前分支切换到main上，因为是刚上手，查阅后才知道master分支更名，所以本地仓库默认的分支是master，远程用的是main。这里建议大家本地和远程主分支名同步。

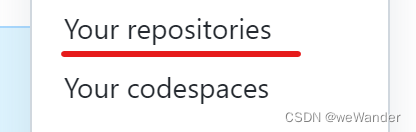
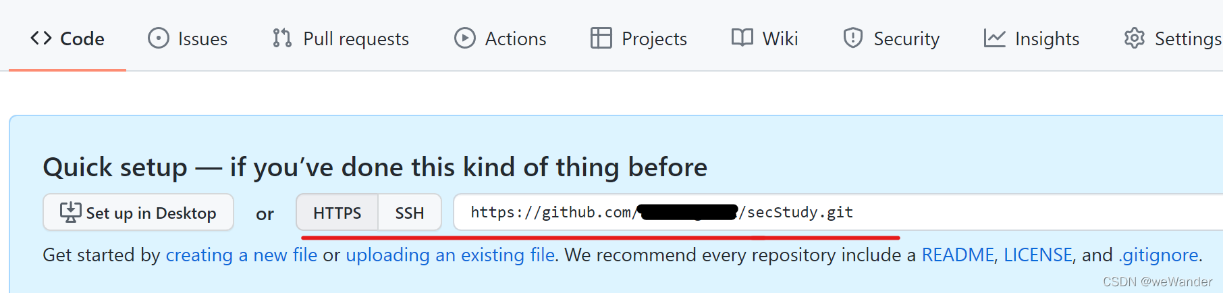
https://img-blog.csdnimg.cn/03dfee8d3f9a4289ba54d1fda98a5cce.png?x-oss-process=image/watermark,type_d3F5LXplbmhlaQ,shadow_50,text_Q1NETiBAd2VXYW5kZXI=,size_20,color_FFFFFF,t_70,g_se,x_16

接着找到你新建的仓库，获取仓库地址（github上完成哦！）远程仓库的访问方式分为两种HTTP和SSH。这里选用HTTP（需要输入用户名和密码，密码不是登录github的密码，是生成的token，后面再介绍）

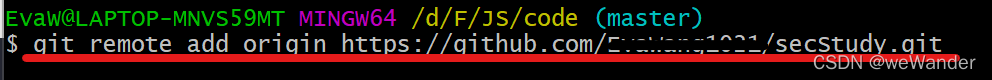


但是我们如果不修改分支的话，我们这里直接使用本地工程中的master分支推送上去也是叫master的

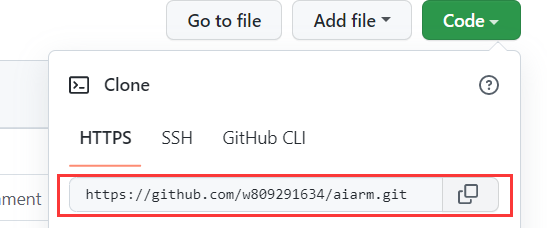
接着找到你新建的仓库，***获取仓库地址（github上完成哦！）远程仓库的访问方式分为两种HTTP和SSH***。这里选用HTTP（需要输入用户名和密码，密码不是登录github的密码，是生成的token，后面再介绍）

然后继续git，下面http的地址就是上面图片红线的地址



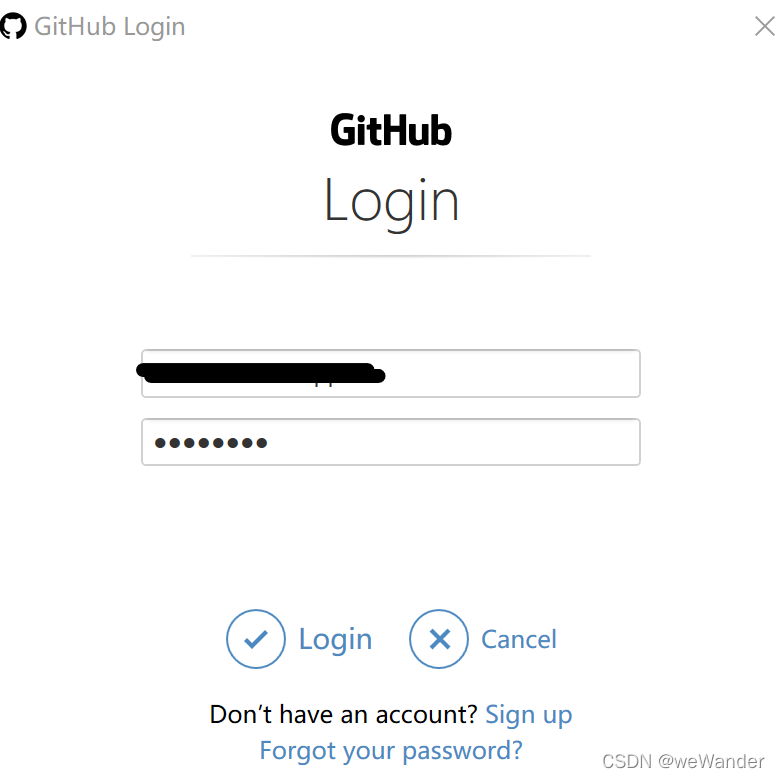
这里将后面的地址改成我们的



3. 请求推送云端仓库，

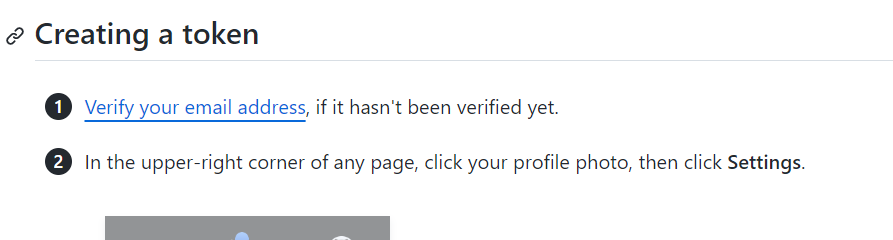
https://img-blog.csdnimg.cn/0fc37acd6999488d85af1956fc1f6ccd.png?x-oss-process=image/watermark,type_d3F5LXplbmhlaQ,shadow_50,text_Q1NETiBAd2VXYW5kZXI=,size_20,color_FFFFFF,t_70,g_se,x_16

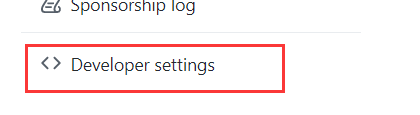
接着就会需要你输入github的账号密码啦



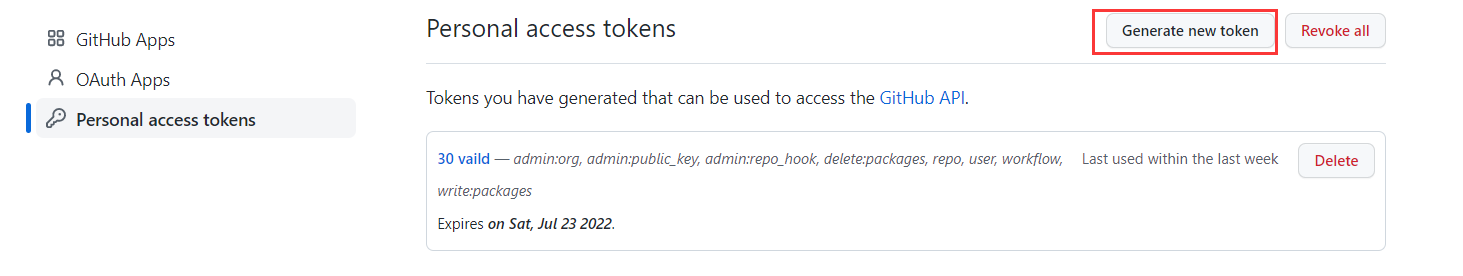
输玩用户名密码后，命令行提示错误。**我查了一下，输入的密码不是github登录密码，换成了所谓的token值**，去这个网址看怎样在github上怎样设置并获取token，

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>

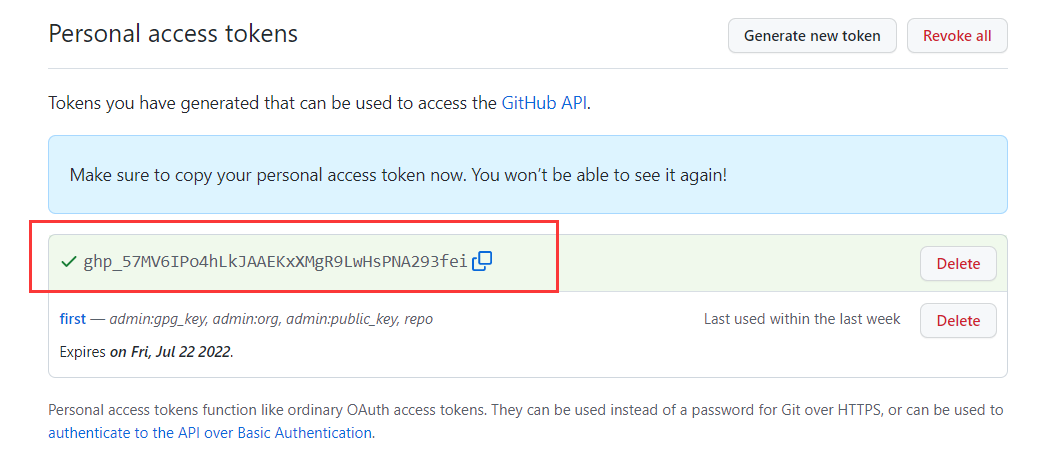




在个人设置的***最底部找到开发者设置***，



创***建完成之后记得保存***。只显示一次。



参考的如下

**809291634@qq.com**

**ghp\_57MV6IPo4hLkJAAEKxXMgR9LwHsPNA293fei**

|  |
| --- |
|  |

|  |
| --- |
| wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  $ git push https://github.com/w809291634/aiarm.git master  Enumerating objects: 139, done.  Counting objects: 100% (139/139), done.  Delta compression using up to 4 threads  Compressing objects: 100% (132/132), done.  Writing objects: 100% (139/139), 3.28 MiB | 2.51 MiB/s, done.  Total 139 (delta 78), reused 0 (delta 0), pack-reused 0  remote: Resolving deltas: 100% (78/78), done.  To https://github.com/w809291634/aiarm.git  \* [new branch] master -> master |

用户名为自己的邮箱，[这里是809291634@qq.com](mailto:这里是809291634@qq.com)

密码为token

The following SSH key was added to your account:

注意下面的这个tokens自己保存，这个页面之后无法获取

##### 问题解决

###### ERROR: You're using an RSA key with SHA-1, which is no longer allowed. Please use a newer client or a different key type.

 原来是 GitHub 在 2022 年 3 月 15 日之后将不再支持 RSA 算法生成的密钥，原因是 RSA 不够安全，而笔者之前一直是使用如下命令生成密钥对的：

ssh-keygen -t rsa -C "邮箱"

  可以看出，在上述命令中，实际上使用的是 RSA 算法来生成的密钥，现在只要更改加密算法即可，可以选择 ed25519。

————————————————

版权声明：本文为CSDN博主「暗诺星刻」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/wangpaiblog/article/details/124416005

|  |
| --- |
| zonesion@zonesion:~/catkin\_ws/src/gazebo\_smartcity$ ssh-keygen -t ed25519 -C "809291634@qq.com"  Generating public/private ed25519 key pair.  Enter file in which to save the key (/home/zonesion/.ssh/id\_ed25519):  /home/zonesion/.ssh/id\_ed25519 already exists.  Overwrite (y/n)? y  Enter passphrase (empty for no passphrase):  Enter same passphrase again:  Your identification has been saved in /home/zonesion/.ssh/id\_ed25519.  Your public key has been saved in /home/zonesion/.ssh/id\_ed25519.pub.  The key fingerprint is:  SHA256:jE98SWgUHgWnbjeCDhSDQJvuKoQOmBKjyJ9Vuve8Ook 809291634@qq.com  The key's randomart image is:  +--[ED25519 256]--+  |oo .o =+o |  | + o o = |  | o . = . |  |+ . O . . |  |\*= . = S = |  |Xo = + + . |  |=.. o + o |  |o. o E +. |  |o ..++. |  +----[SHA256]-----+  zonesion@zonesion:~/catkin\_ws/src/gazebo\_smartcity$ cat /home/zonesion/.ssh/id\_ed25519.pub  ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIGEgnieEOXc2wOPg33mGjgolotdhWRRtRxZDt6s980Kg 809291634@qq.com  zonesion@zonesion:~/catkin\_ws/src/gazebo\_smartcity$ git push -u origin main  对象计数中: 153, 完成.  Delta compression using up to 4 threads.  压缩对象中: 100% (140/140), 完成.  写入对象中: 100% (153/153), 12.05 MiB | 1.01 MiB/s, 完成.  Total 153 (delta 37), reused 0 (delta 0)  remote: Resolving deltas: 100% (37/37), done.  To git@github.com:w809291634/gazebo\_smartcity.git  \* [new branch] main -> main  分支 main 设置为跟踪来自 origin 的远程分支 main。  zonesion@zonesion:~/catkin\_ws/src/gazebo\_smartcity$ |

|  |
| --- |
|  |

###### 不断提示需要password

git@github.com's password:

Permission denied, please try again.

git@github.com's password:

Permission denied, please try again.

git@github.com's password:

<https://blog.csdn.net/li4692625/article/details/120307465>

|  |
| --- |
| 输入密码无效；  解决办法：  vim ~/.ssh/config  #添加  Host github.com  identityFile ~/.ssh/id\_rsa#自己的私钥文件名 这里要改  Host github.com  Hostname ssh.github.com  Port 443  User git |

|  |
| --- |
| zonesion@zonesion:~/catkin\_ws/src/laser\_uwb\_loc$ ssh-keygen -t ed25519 -C "809291634@qq.com"  Generating public/private ed25519 key pair.  Enter file in which to save the key (/home/zonesion/.ssh/id\_ed25519):  Enter passphrase (empty for no passphrase):  Enter same passphrase again:  Your identification has been saved in /home/zonesion/.ssh/id\_ed25519.  Your public key has been saved in /home/zonesion/.ssh/id\_ed25519.pub.  The key fingerprint is:  SHA256:Siw9KQs1Q4RAsnxC5DHm4kn5S9/aBBOaNKNoMzW9r8U 809291634@qq.com  The key's randomart image is:  +--[ED25519 256]--+  |=O oo |  |B.=.. |  |oB B+o |  |= O.\*=o. |  |.B.=o+\* S |  |. +.o=\*o |  | ....E |  | \* |  | o . |  +----[SHA256]-----+ |

vim ~/.ssh/config

|  |
| --- |
| Host github.com  identityFile ~/.ssh/id\_ed25519  Host github.com  Hostname ssh.github.com  Port 443  User git |

主要修改这里的ID文件。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

#### 解决git push要输入密码，git的配置文件

git config --global user.name "xxx" git config --global user.email "xxx"

通常修改的是

cat ~/.gitconfig

这个全局配置文件

但是也有一个git包的单独配置文件

##### 方法一 修改全局的配置文件

###### 一、在客户端把用户名和密码配入配置文件

#保存身份信息

**git config --global credential.helper store**

接下来再提交的时候，输入一次帐号和密码之后就不需要再次输入

|  |
| --- |
| wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  $ **git push origin master**  Everything up-to-date |

输入一次后再次输入提交就不需要密码了。

wanghao@DESKTOP-KQMGM9O MINGW64 ~

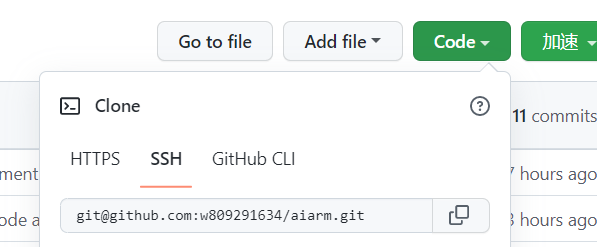
$ ls .gitconfig

.gitconfig

|  |
| --- |
| wanghao@DESKTOP-KQMGM9O MINGW64 ~  $ cat ~/.gitconfig  [user]  email = 809291634@qq.com  name = w809291634  [core]  autocrlf = input  [safe]  directory = D:/Program\_Files/RT-ThreadStudio/workspace/ZXBee\_Plus\_test  directory = D:/Program\_Files/RT-ThreadStudio/workspace/ZXBee\_Plus\_VET6  directory = D:/Program\_Files/RT-ThreadStudio/workspace  [credential]  helper = store |

|  |
| --- |
| wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  $ git push origin master  Everything up-to-date |

###### 二、使用ssh方式



[git@github.com:w809291634/aiarm.git](mailto:git@github.com:w809291634/aiarm.git)

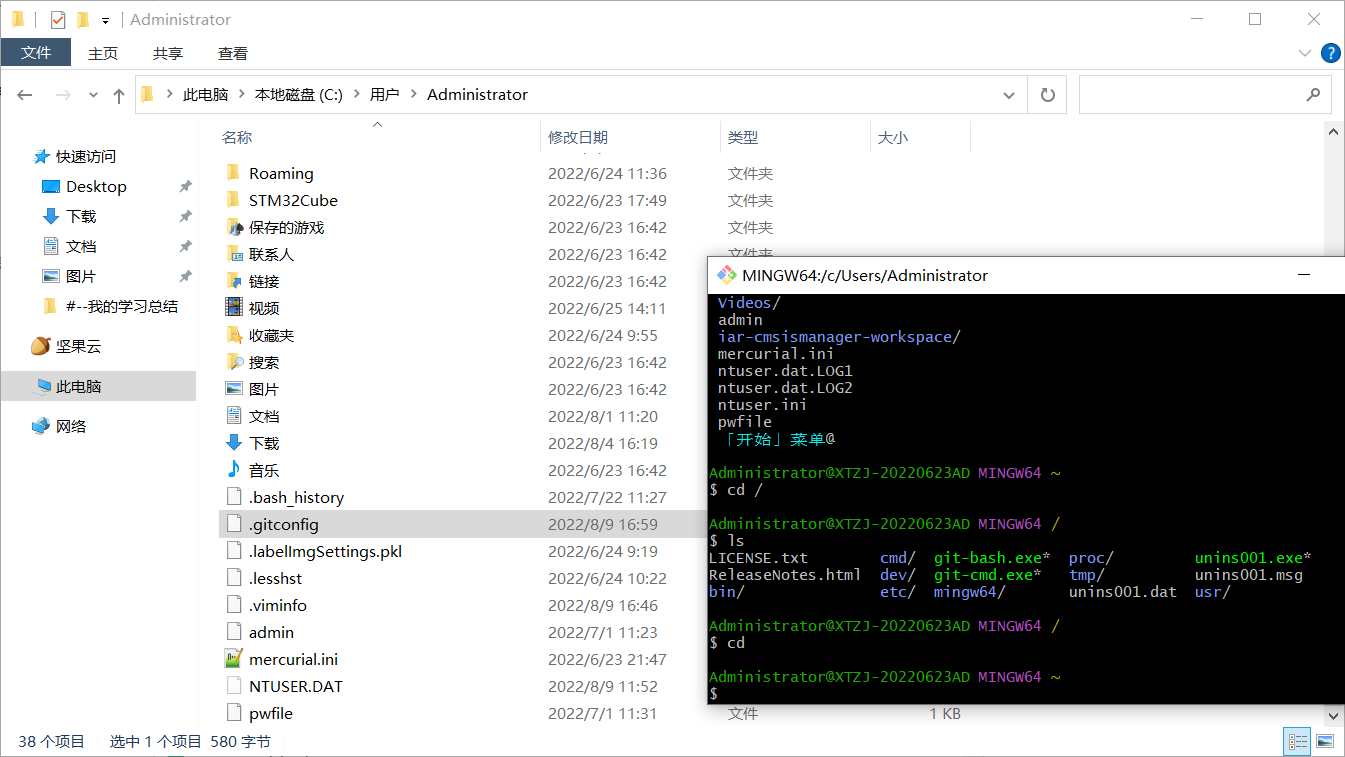
替换origin到ssh方式

|  |
| --- |
| wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  $ git remote add origin git@github.com:w809291634/aiarm.git  error: remote origin already exists.  wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  $ git remote rm origin  wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  **$ git remote add origin git@github.com:w809291634/aiarm.git**  wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)  $ git remote -v  origin git@github.com:w809291634/aiarm.git (fetch)  origin git@github.com:w809291634/aiarm.git (push) |

wanghao@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/#--项目经验代码/鱼眼相机抓取及颜色识别/机械臂视觉(最新) (master)

$ git push origin master

Everything up-to-date



全局配置文件通常存放在如上目录中。

##### 方法二 修改本地的配置文件

|  |
| --- |
| Administrator@XTZJ-20220623AD MINGW64 /e/current\_task/gazebo智慧城市小车 (master)  $ **cat .git/config**  [core]  repositoryformatversion = 0  filemode = false  bare = false  logallrefupdates = true  symlinks = false  ignorecase = true |

正确的步骤：

步骤一： 更改git commit的时候用到的用户信息

|  |
| --- |
| git config --local user.name "wanghao"  git config --local user.email "809291634@qq.com"  git config --global user.name "xxx" //非必须执行  git config --global user.email "xxx" //非必须执行 |

输入上面两条信息后得到如下

|  |
| --- |
| Administrator@XTZJ-20220623AD MINGW64 /e/current\_task/gazebo智慧城市小车 (master) cat .git/config  [core]  repositoryformatversion = 0  filemode = false  bare = false  logallrefupdates = true  symlinks = false  ignorecase = true  [user]  name = wanghao  email = 809291634@qq.com |

修改你本地git仓库里面的config文件。目录位于 .git -》config 文件 。在url前面手动输入用户名和密码 格式为 http://或者https://username:userpassword@具体的仓库地址(这里不要写协议名称，就是http，htthps). 示例 https://test:http://testpasswd@gitlab.test.com/test.git

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

### git gui

#### Git中的Sign Off功能是什么？

签名是提交消息结尾的一行，用于证明提交作者是谁。它的主要目的是提高跟踪谁做了什么，特别是补丁。

|  |
| --- |
|  |

|  |
| --- |
|  |

### Sourcetree使用

如果使用时无法上传到GitHub，这里就要注意使用openssh



<https://blog.csdn.net/qq_37910492/article/details/85097215> 公私钥的使用注意事项

#### 1.使用 git 客户的生成公私钥：id\_rsa、id\_rsa.pub

1.1设置Git的user name和email：

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

$ git config --global user.email "xxx.mail@xxx.com"

1.2.生成SSH密钥过程：

1.2.1.检查是不是已经存在密钥（能进去说明已经存在，就删掉文件夹，重新创建）：

cd ~/.ssh

1.3.生成 SSH 密钥：

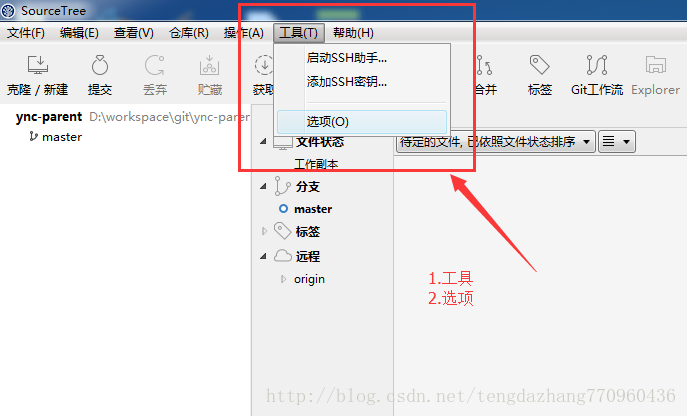
$ ssh-keygen -t rsa -C “xxx.mail@xxx.com”

按3个回车，密码为空。

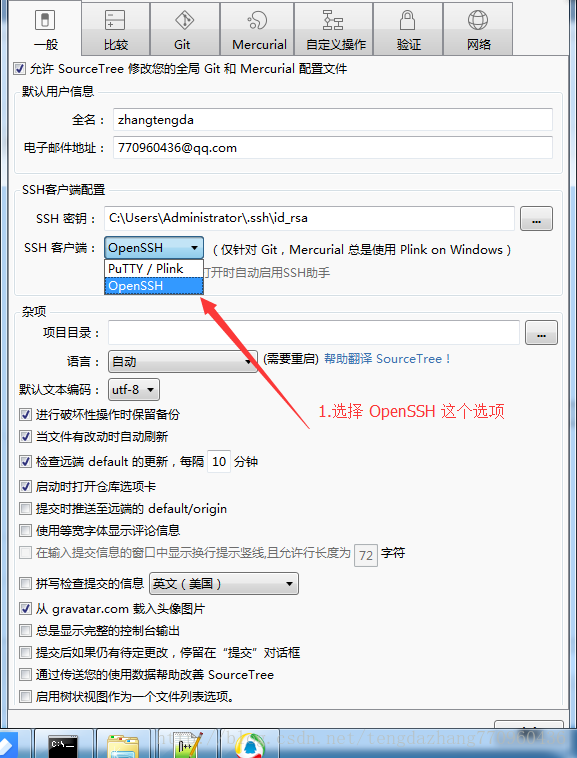
1.4.文件存放位置 ~/.ssh，如果是window的话就在：C:\Users\Administrator.ssh 下面，当然如果你不是 Administrator 用户的话，需要换成对应的用户。

#### 2.设置 SourceTree 的 SSH客户端

1.配置SourceTree 的 SSH 客户的为：OpenSSH



1.2.设置 OpenSSH,这时候，SSH 密钥这一栏自然会去选择当前用户下的 .ssh 目录下的 id\_rsa 这个私钥：



#### 4.解释：

1.ssh-keygen 是公钥私钥的非对称加密方式：

1.1.公钥：用于向外发布，任何人都能获取。

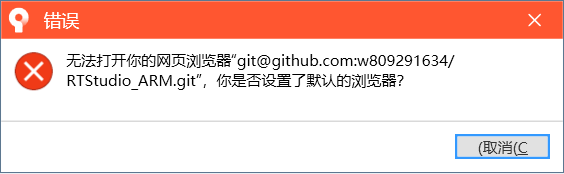
1.2.私钥：要自己保存，切勿给别人

2.公钥私钥加解密的原理

2.1.客户端把自己的公钥存放到要链接的远程主机上（相当于我们把自己的 id\_rsa.pub 存放到 git 服务器上）

2.2.客户端要链接远程主机的时候，远程主机会向客户的发送一条随机的字符串，客户的收到字符串之后使用自己的私钥对字符串加密然后发送到远程主机，远程主机根据自己存放的公钥对这个字符串进行解密，如果解密成功证明客户端是可信的，直接允许登录，不在要求登录。

#### 无法打开网络浏览器







设置用户名即可。保存后点击就正常，重启之后就没有红色感叹号

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

### 常见问题

#### 1、使用bash here功能，使用bash终端使用ctrl+v复制文本出现如下错误



汪浩@DESKTOP-KQMGM9O MINGW64 /e/3-嵌入式开发/bash

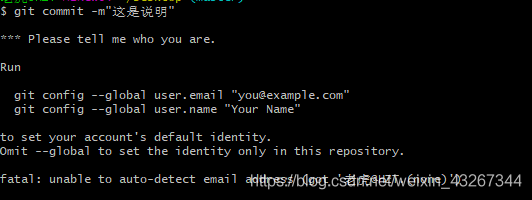
$ git clone http://192.168.10.249:8888/aicam/algorithm.git

Cloning into 'algorithm'...

fatal: protocol 'http' is not supported

原因是这里的QQ复制的文本在bash中粘贴时出现clone http，删除空格重新填入后正常克隆

#### 2、git config --global user.email "you@example.com" git config --global user.name "Your name"



这个错误的原因就是没有设置账户的默认身份，那么很简单了，没有设置默认身份那就去设置一下就OK了。

打开创建的git文件夹，找到里面的config文件，打开之后在最后一句加上：

[user]

email=Your email

name=Your Name

#### 3、执行Git命令时报错 bash: $'\302\226git': command not found

bash: $'\302\226git': command not found

解决方式

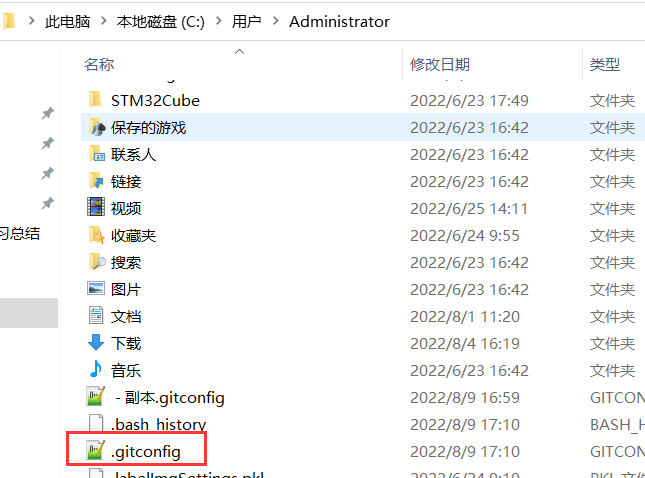
此种情况常出现在复制、粘贴命令时，命令开头往往会有多余的空格。

检查命令中是否有多余的空格，去除多余的空格即可。

#### 4、error: '提交代码/ARMEmbed-rttstudio/' does not have a commit checked out】

一定要在有git的目录进行相应的提交。

#### 5、unsafe repository





增加git所在目录即可

git config --global --add safe.directory /目录

或者上面这个命令，路径要对

#### 6、error: src refspec main does not match any

提交代码是，有可能缺少readme.txt文件

#### 7、git的换行符问题

1，不同平台对换行符的制定不同

windows <回车换行> （carriage return AND line feed） “ ” CRLF

Unix系统 <换行> （carriage return） “ ” CR

window unix mac 等平台，换行符的不同：

3，不同平台换行符的转化

windows -> unix :

1，用unix 平台的命令工具：

1. cat a.txt | tr -d "^M" > b.txt

2. cat a.txt | sed 's/^M//g'

3. vim 中执行sed 命令替换 ( :^,$ s/^m//g ;or :s/

//g; )

4. dos2unix 文件 （适用与对文件夹所有文件进行转化）

unix ->windows :

在window 平台适用 UltraEdit 和 EditPlus 等工具的 可以设置换行模式进行自动转化

或者在unix 平台中使用命令：

1. cat a.txt | sed 's/

/

/g'

2. unix2dos 文件 （适用与对文件夹所有文件进行转化）

2，gitbash 中设置默认检入检出换行符为linux（ ）风格：（为了团队更好的协作）

1，开启自动换行

[a@b]$ git config --global core.autocrlf true

#签出时将换行符转换成CRLF，签入时转换回 LF。 git config --global core.autocrlf true

#签出时不转换换行符，签入时转换回 LF git config --global core.autocrlf input

#签出签入均不转换 git config --global core.autocrlf false

2，开启安全换行

[a@b]$ git config --global core.safecrlf true

解释：

如果你把换行符搞乱了，在一个文件中既包含windows风格的换行符也包含unix风格换行符，那么 safecrlf 就可以发挥作用了：

#拒绝提交包含混合换行符的文件 git config --global core.safecrlf true

#允许提交包含混合换行符的文件 git config --global core.safecrlf false

#提交包含混合换行符的文件时候给出警示 git config --global core.safecrlf warn

1.Git设置

git config --global core.autocrlf false

git config --global core.safecrlf true

含义：

AutoCRLF

#提交时转换为LF，检出时转换为CRLF

git config --global core.autocrlf true

#提交时转换为LF，检出时不转换

git config --global core.autocrlf input

#提交检出均不转换

git config --global core.autocrlf false

SafeCRLF

#拒绝提交包含混合换行符的文件

git config --global core.safecrlf true

#允许提交包含混合换行符的文件

git config --global core.safecrlf false

#提交包含混合换行符的文件时给出警告

git config --global core.safecrlf warn

2.IDE设置使用UNIX换行符

IDEA的设置File -> Settings

Editor -> Code Style

Line separator (for new lines) ，选择：Unix and OS X (\n)

对已使用Windows换行符的文件，可以使用Sublime Text打开，

View->Line Endings，选Unix，保存；

vscode的设置 文件->首选项->设置

搜索->Eol->设置默认行尾字符为\n

————————————————

版权声明：本文为CSDN博主「普通网友」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/JackieDYH/article/details/123371921