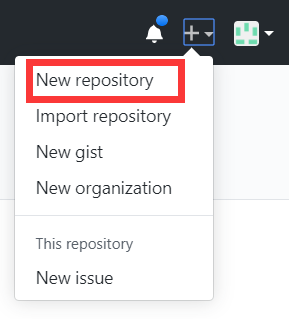
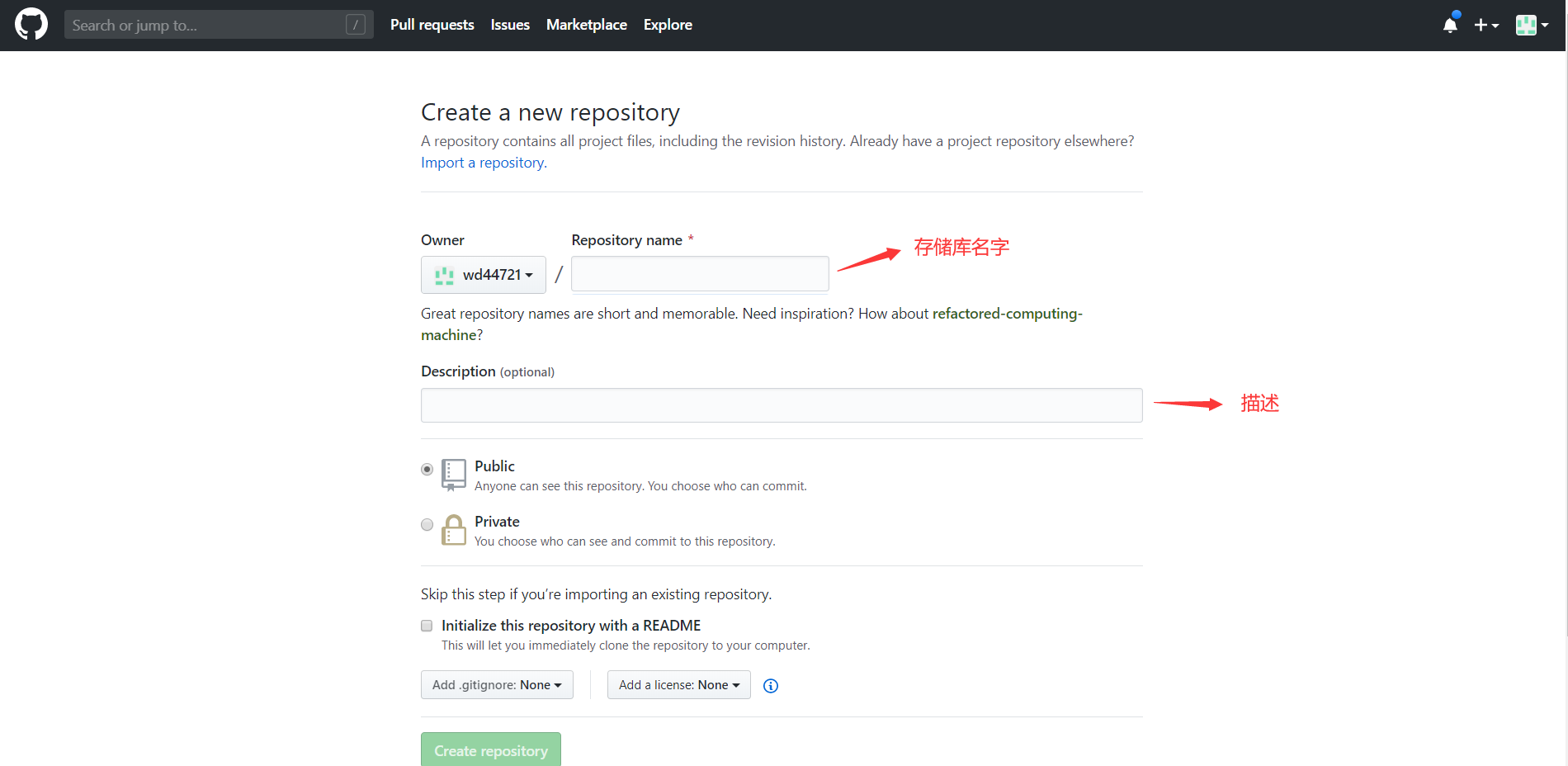
1. GitHub官方网址：<https://github.com>
2. 注册登录后首先创建一个自己的存储库，在右上上角的加号箭头。





在创建完存储库之后，需要在电脑上克隆一个自己所创建的库，方便代码同步。

1. 这里就需要安装一个软件GitBash。

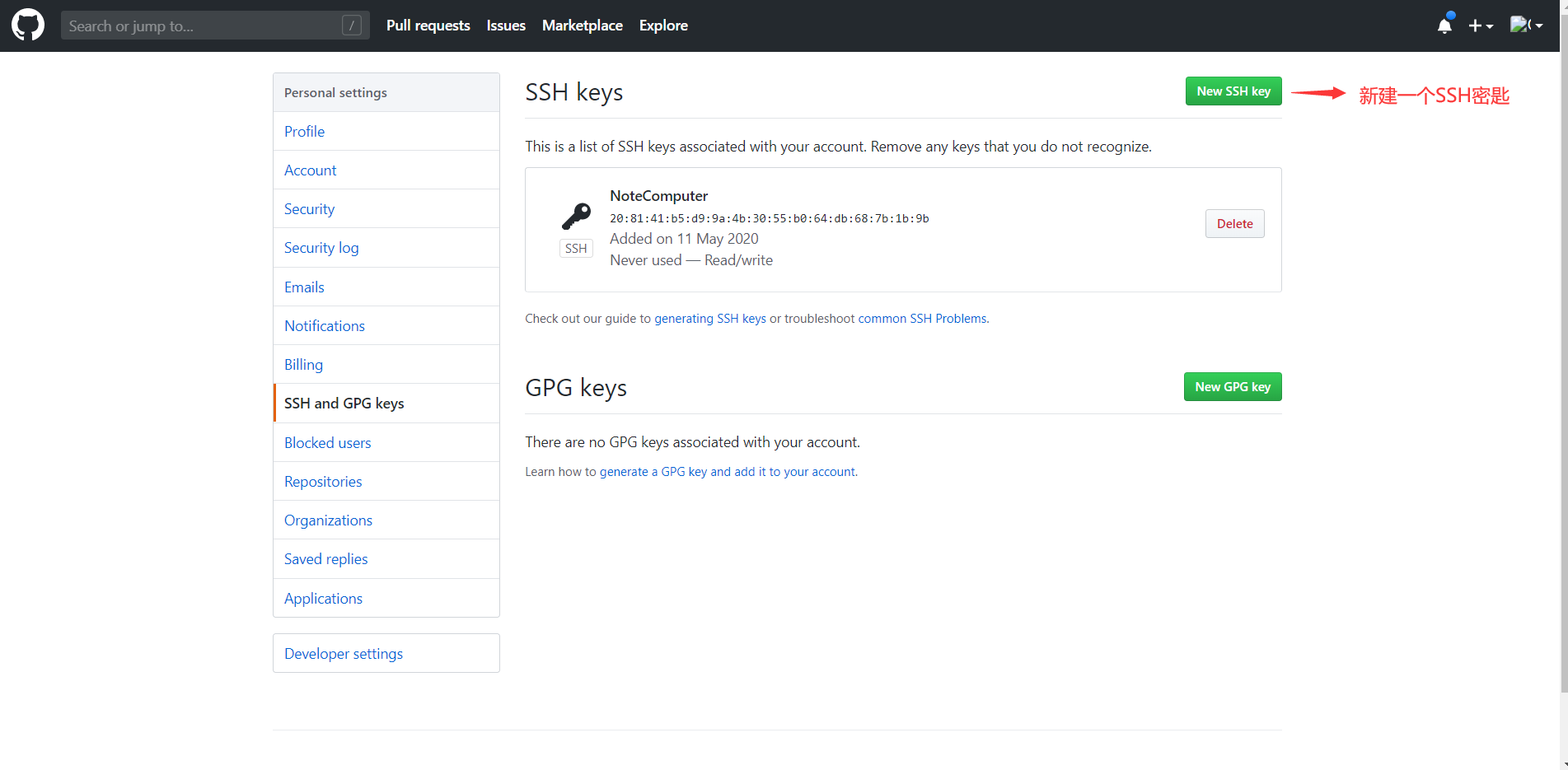
GitBash下载地址：<https://git-scm.com>

安装默认配置一直点击下一步即可。下面是配置GitBash。

3-1、首先需要在本地创建一个ssh key，目的是在你的电脑上获得一个密匙，获取之后在你的GitHub里面输入这个密匙。这样你的电脑就和GitHub绑定在一起了。

3-2、基本步骤：

1. 输入$ ssh-keygen -t rsa-C “你的邮箱地址”，然后一直按回车，最后会在C:\Users\HolmeDu（我的主机名）\.ssh\路径下生成一个id\_rsa.pub的文件，用记事本打开后复制那一串长长的密匙。回到GitHub页面里输入这一段密匙。



回到GitBash命令框，验证一下是否绑定成功

输入$ ssh -T [git@github.com](mailto:git@github.com)检查一下。第一次绑定的时候会提示是否continue，输入yes。

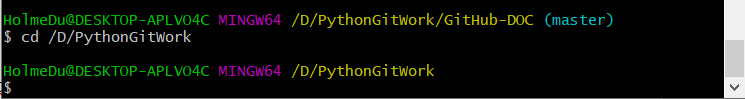
1. 配置账号密码。

$ git config -global user.name “your name”

$ git config -global user.email “your email”

1. 定位gitbash项目文件路径

输入$cd /D/PythonGitWork



输入$git clone <https://github.com/wd44721/Github_doc.git>

然后就能看到PythonGitWork文件夹里面多了一个Github-DOC的文件夹。

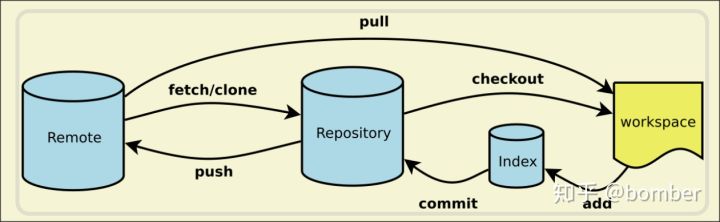
1. 同步文件到GitHub服务器上。本地新建一个test.txt

首先定位到Github-DOC文件夹里，然后输入：

$git add test.txt

$git commit -m “这是一个test描述”

$git push origin master



**一、新建代码库**

# 在当前目录新建一个Git代码库

$ git init

# 新建一个目录，将其初始化为Git代码库

$ git init [project-name]

# 下载一个项目和它的整个代码历史

$ git clone [url]

**二、配置**

Git的设置文件为.gitconfig，它可以在用户主目录下（全局配置），也可以在项目目录下（项目配置）。

# 显示当前的Git配置

$ git config --list

# 编辑Git配置文件

$ git config -e [--global]

# 设置提交代码时的用户信息

$ git config [--global] user.name "[name]"

$ git config [--global] user.email "[email address]"

**三、增加/删除文件**

# 添加指定文件到暂存区

$ git add [file1] [file2] ...

# 添加指定目录到暂存区，包括子目录

$ git add [dir]

# 添加当前目录的所有文件到暂存区

$ git add .

# 添加每个变化前，都会要求确认

# 对于同一个文件的多处变化，可以实现分次提交

$ git add -p

# 删除工作区文件，并且将这次删除放入暂存区

$ git rm [file1] [file2] ...

# 停止追踪指定文件，但该文件会保留在工作区

$ git rm --cached [file]

# 改名文件，并且将这个改名放入暂存区

$ git mv [file-original] [file-renamed]

**四、代码提交**

# 提交暂存区到仓库区

$ git commit -m [message]

# 提交暂存区的指定文件到仓库区

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

# 提交工作区自上次commit之后的变化，直接到仓库区

$ git commit -a

# 提交时显示所有diff信息

$ git commit -v

# 使用一次新的commit，替代上一次提交

# 如果代码没有任何新变化，则用来改写上一次commit的提交信息

$ git commit --amend -m [message]

# 重做上一次commit，并包括指定文件的新变化

$ git commit --amend [file1] [file2] ...

**五、分支**

# 列出所有本地分支

$ git branch

# 列出所有远程分支

$ git branch -r

# 列出所有本地分支和远程分支

$ git branch -a

# 新建一个分支，但依然停留在当前分支

$ git branch [branch-name]

# 新建一个分支，并切换到该分支

$ git checkout -b [branch]

# 新建一个分支，指向指定commit

$ git branch [branch] [commit]

# 新建一个分支，与指定的远程分支建立追踪关系

$ git branch --track [branch] [remote-branch]

# 切换到指定分支，并更新工作区

$ git checkout [branch-name]

# 切换到上一个分支

$ git checkout -

# 建立追踪关系，在现有分支与指定的远程分支之间

$ git branch --set-upstream [branch] [remote-branch]

# 合并指定分支到当前分支

$ git merge [branch]

# 选择一个commit，合并进当前分支

$ git cherry-pick [commit]

# 删除分支

$ git branch -d [branch-name]

# 删除远程分支

$ git push origin --delete [branch-name]

$ git branch -dr [remote/branch]

**六、标签**

# 列出所有tag

$ git tag

# 新建一个tag在当前commit

$ git tag [tag]

# 新建一个tag在指定commit

$ git tag [tag] [commit]

# 删除本地tag

$ git tag -d [tag]

# 删除远程tag

$ git push origin :refs/tags/[tagName]

# 查看tag信息

$ git show [tag]

# 提交指定tag

$ git push [remote] [tag]

# 提交所有tag

$ git push [remote] --tags

# 新建一个分支，指向某个tag

$ git checkout -b [branch] [tag]

**七、查看信息**

# 显示有变更的文件

$ git status

# 显示当前分支的版本历史

$ git log

# 显示commit历史，以及每次commit发生变更的文件

$ git log --stat

# 搜索提交历史，根据关键词

$ git log -S [keyword]

# 显示某个commit之后的所有变动，每个commit占据一行

$ git log [tag] HEAD --pretty=format:%s

# 显示某个commit之后的所有变动，其"提交说明"必须符合搜索条件

$ git log [tag] HEAD --grep feature

# 显示某个文件的版本历史，包括文件改名

$ git log --follow [file]

$ git whatchanged [file]

# 显示指定文件相关的每一次diff

$ git log -p [file]

# 显示过去5次提交

$ git log -5 --pretty --oneline

# 显示所有提交过的用户，按提交次数排序

$ git shortlog -sn

# 显示指定文件是什么人在什么时间修改过

$ git blame [file]

# 显示暂存区和工作区的差异

$ git diff

# 显示暂存区和上一个commit的差异

$ git diff --cached [file]

# 显示工作区与当前分支最新commit之间的差异

$ git diff HEAD

# 显示两次提交之间的差异

$ git diff [first-branch]...[second-branch]

# 显示今天你写了多少行代码

$ git diff --shortstat "@{0 day ago}"

# 显示某次提交的元数据和内容变化

$ git show [commit]

# 显示某次提交发生变化的文件

$ git show --name-only [commit]

# 显示某次提交时，某个文件的内容

$ git show [commit]:[filename]

# 显示当前分支的最近几次提交

$ git reflog

**八、远程同步**

# 下载远程仓库的所有变动

$ git fetch [remote]

# 显示所有远程仓库

$ git remote -v

# 显示某个远程仓库的信息

$ git remote show [remote]

# 增加一个新的远程仓库，并命名

$ git remote add [shortname] [url]

# 取回远程仓库的变化，并与本地分支合并

$ git pull [remote] [branch]

# 上传本地指定分支到远程仓库

$ git push [remote] [branch]

# 强行推送当前分支到远程仓库，即使有冲突

$ git push [remote] --force

# 推送所有分支到远程仓库

$ git push [remote] --all

**九、撤销**

# 恢复暂存区的指定文件到工作区

$ git checkout [file]

# 恢复某个commit的指定文件到暂存区和工作区

$ git checkout [commit] [file]

# 恢复暂存区的所有文件到工作区

$ git checkout .

# 重置暂存区的指定文件，与上一次commit保持一致，但工作区不变

$ git reset [file]

# 重置暂存区与工作区，与上一次commit保持一致

$ git reset --hard

# 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变

$ git reset [commit]

# 重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

$ git reset --hard [commit]

# 重置当前HEAD为指定commit，但保持暂存区和工作区不变

$ git reset --keep [commit]

# 新建一个commit，用来撤销指定commit

# 后者的所有变化都将被前者抵消，并且应用到当前分支

$ git revert [commit]

# 暂时将未提交的变化移除，稍后再移入

$ git stash

$ git stash pop

**十、其他**

# 生成一个可供发布的压缩包

$ git archive