

Git&Github

1.版本控制

1.1.版本控制介绍

记录历史状态，便于追溯，有后悔药吃

1.2.版本控制工具介绍

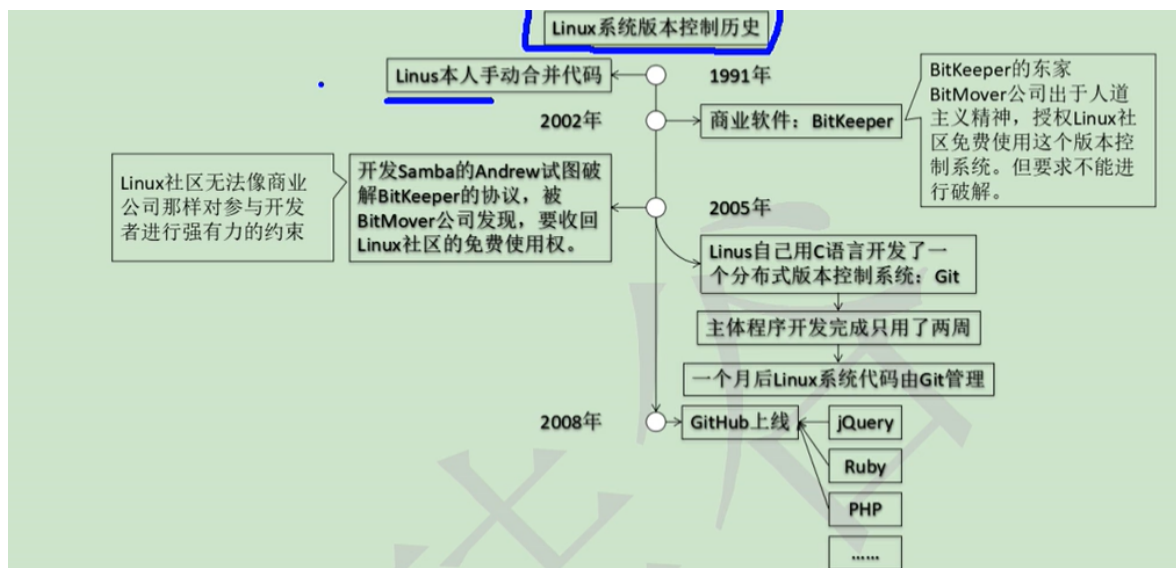
1.2.1.版本控制工具应该具备的功能

- 协同修改
多人并行不悖的修改服务器端的同一个文件
- 数据备份
不仅保存目录和文件的当前状态，还能够保存每一个提交过的历史状态
- 版本管理
在保存每一个版本的文件信息的时候要做到不保存重复数据，以节约存储空间提高运行效率。这方面SVN采用的是增量式管理的方式，而Git采取了文件系统快照的方式。
- 权限控制
对团队中参与开发的人员进行权限控制
对团队外开发者贡献的代码进行审查—Git独有
- 历史记录
查看修改人、修改时间、修改内容、日志信息将本地文件恢复到某一个历史状态
- 分支管理
允许开发团队在工作过程中多条生产线同时推进任务，进一步提高效率

1.2.2.版本控制工具介绍

- 集中式版本控制工具
CVS、SVN、VSS
- 分布式版本控制工具
Git、Mercurial、Bazaar、Darcs.....

2.Git简介



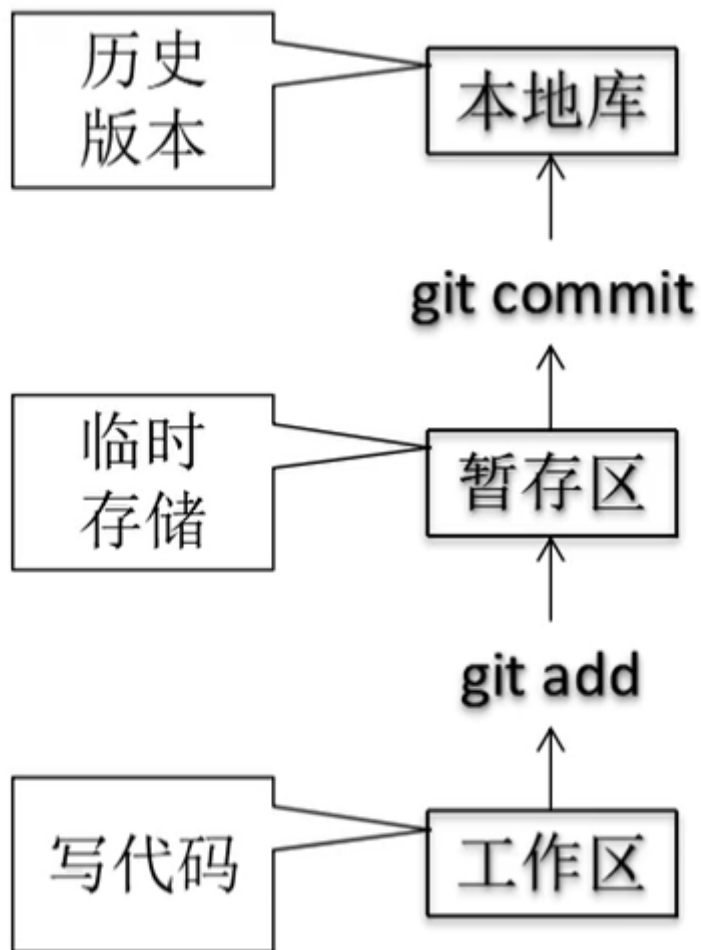
2.1 Git优势

- 大部分操作在本地完成，不需要联网
- 完整性保证
- 尽可能添加数据而不是删除或修改数据
- 分支操作非常快捷流畅
- 与 Linux命令全面兼容

2.2 安装Git

[Git官网下载地址](#)

2.3 git 结构



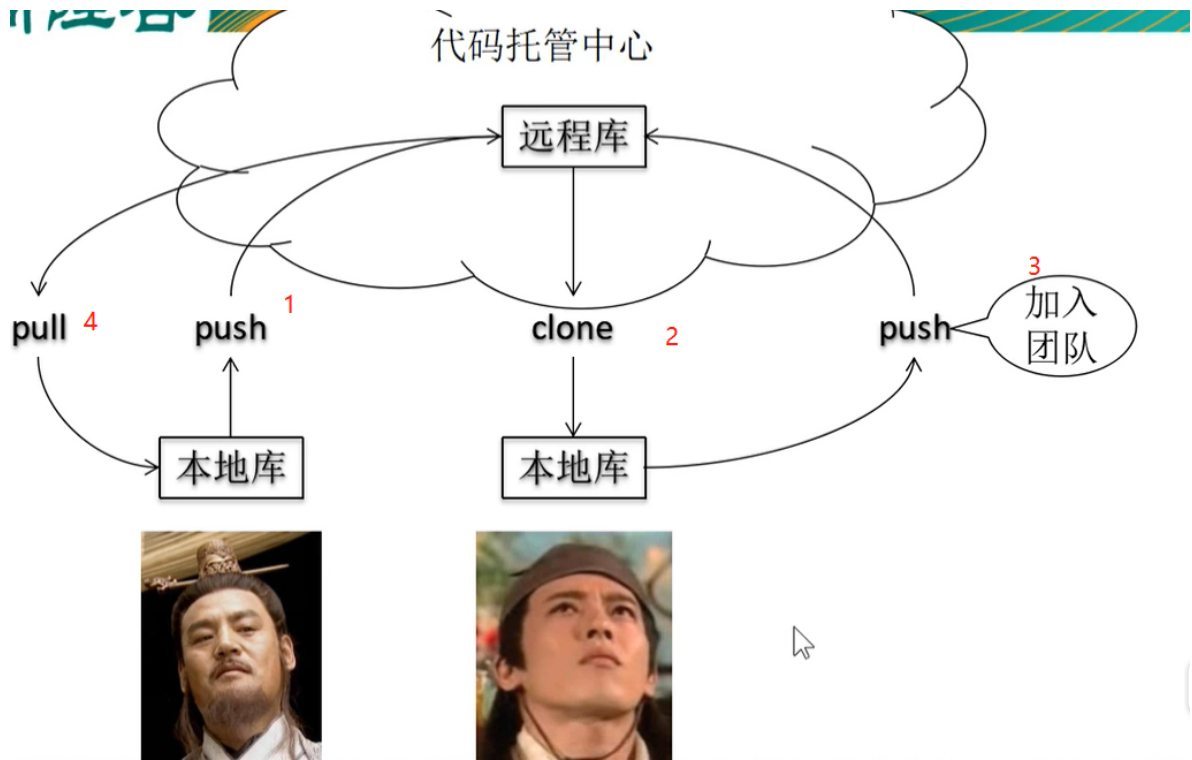
2.4 Git 和代码托管中心

代码托管中心的任务是维护远程库

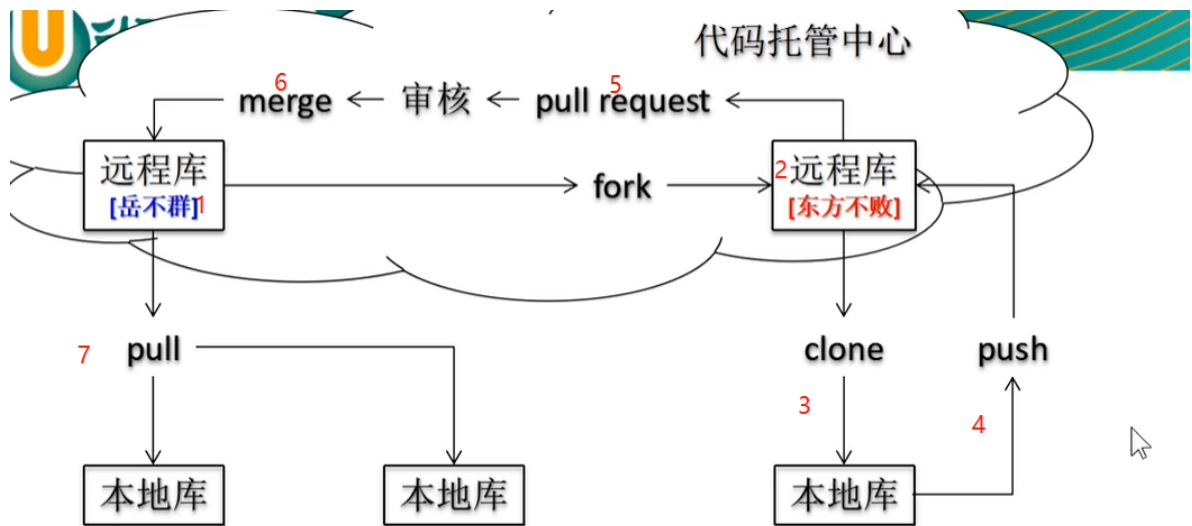
- 局域网环境下
GitLab 服务器
- 外网环境下
Github
码云

2.5 本地库和远程库

团队内开发



团队外部开发



3. Git命令操作

3.1 本地库操作

3.1.1 本地库初始化

查看隐藏文件命令，Linux以·开头的文件都是隐藏资源

```
ls -lA
```

- 进入想要创建git仓库的目录如 (F:\GitHubLocalWareHouse) ，执行

```
git init # 初始化
ls -lA   # 查看 .git目录
```

```
$ ll .git/
total 7
-rw-r--r-- 1 Lenovo 197121 130 5月 10 16:53 config
-rw-r--r-- 1 Lenovo 197121 73 5月 10 16:53 description
-rw-r--r-- 1 Lenovo 197121 23 5月 10 16:53 HEAD
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 hooks/
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 info/
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 objects/
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 refs/
```

注意： .git 目录中存放的是本地库相关的子目录和文件，不要删除，不要乱改。

- 设置签名

- 形式

- 用户名: xxx

- Email地址: xxx

- 作用:

- 区分不同开发人员的身份

- 辨析：这里设置的签名和登陆远程库（代码托管中心）的账号密码没有关系。

- 命令

- 项目级别/仓库级别：仅在当前本地仓库范围内有效

```
git config user.name [用户名]
git config user.email [Email]
```

系统用户级别：登陆当前操作用户范围

```
git config --global user.name [用户名]
git config --global user.name [Email]
```

注意： 优先级采用就近原则，项目级别优先于系统用户级别，二者都有，采用项目级别签名。二者都没有不允许。

项目级别设置的签名保存在 `.git/config` 中，该文件为隐藏文件。

系统用户级别保存在 `~/.gitconfig` 中，该文件为隐藏文件。

3.1.2 基本操作

查看工作区、暂存区状态

```
git status
```

```
MINGW64:/f/GitHubLocalWareHouse
$ cat ./git/config
cat: ./git/config: No such file or directory

ThinkPad@LAPTOP-3FCRPIUN MINGW64 /f/GitHubLocalWareHouse (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    CloudImage/Image/05df4e14736fe9537a3165723525a04.png
        deleted:    CloudImage/Image/0e6f8f71fe9d093bc423e26c7888b09.png
        deleted:    CloudImage/Image/123d7e1646231aac7f6d4b1b8f90360.png
        deleted:    CloudImage/Image/130300d8b3c4282867a4279bf124733.png
        deleted:    CloudImage/Image/13ce8e12714fe8714533fbb797b3cb2.png
        deleted:    CloudImage/Image/2b8ee33ed47e46c7b8e630c0a408807.png
        deleted:    CloudImage/Image/375bf2fa54c20c3e0a529f67560e6cd.png
        deleted:    CloudImage/Image/455b6b2490ca58fccd3ce8b6e3c00db.png
        deleted:    CloudImage/Image/5a6b81068027f213c51224a956edb73.png
        deleted:    CloudImage/Image/5f94187524791c6dc7189092e68ac6b.png
        deleted:    CloudImage/Image/60070a5f00ed7d9fc72fe4416143083.png
```

将文件的新建或修改从工作区添加到暂存区

```
git add [filename]
```

将文件从暂存区撤回回来

```
git rm --cached [filename]
```

将文件从暂存区提交到本地库

```
git commit [filename] # 进VIM编辑器写修改提示
```

```
git commit -m "提示内容" [文件名] #这种做法更常见
```

注意: ** 不写文件名将提交暂存区的所有文件

3.1.3 对版本历史记录进行查看

(1) 查看历史记录:

```
git log # 查看log日志
```

```
git log --pretty=oneline # 每行一条显示日志
git log --oneline        # 和上面相同
git reflog              # 显示需要移动的步数
```

(2) 执行版本前进后退的三种基本方式 (操作HEAD指针)

- 基于索引值操作【推荐】

查看当前版本所在的位置

```
git reflog
```

```
ThinkPad@LAPTOP-3FCRPIUN MINGW64 /f/GitHubLocalWareHouse/BigData-java/Hadoop集群
搭建 (master)
$ git rebase
e475a42 (HEAD -> master) HEAD@{0}: reset: moving to HEAD
e475a42 (HEAD -> master) HEAD@{1}: reset: moving to HEAD
e475a42 (HEAD -> master) HEAD@{2}: commit: 20200430 update20200430 update20200430
0 update20200430 update20200430
f510a33 HEAD@{3}: commit: oracle file
16c8e76 (origin/master, origin/HEAD) HEAD@{4}: commit: 20200427更新
d201bc6 HEAD@{5}: commit: mysql 双机热备
64504d0 HEAD@{6}: commit: mysql双机热备
67d7031 HEAD@{7}: commit: cloud image token
975ec9a HEAD@{8}: commit: shadowsocks client
f10ccc9 HEAD@{9}: commit: 集群搭建
aa4c64e HEAD@{10}: commit: 集群搭建
b911392 HEAD@{11}: clone: from https://github.com/yimisiyang/BigData-java.git
ThinkPad@LAPTOP-3FCRPIUN MINGW64 /f/GitHubLocalWareHouse/BigData-java/Hadoop集群
搭建 (master)
```

```
# 回退到某一个版本
git reset --hard [回退的版本号]
```

```
# 前进到某一个版本
git reset --hard [前进的版本号]
```

- 使用^符号 (只能后退, 不能往前)

```
git reset --hard HEAD^ #加一个^符号回退一步, 多个回退多步。
```

- 使用~符号(为了化解回退多步时^的尴尬) (同样只能后退)

```
git reset --hard HEAD~3 # 回退3步
```

(3) reset 命令的三个参数对比

- --soft
仅仅在本地库移动HEAD指针。
- --mixed
在本地库移动HEAD指针
重置暂存区
- --hard
在本地库移动HEAD指针, 同时重置暂存区和工作区。

(4) 删除文件的找回

回退到没有删除的那个版本即可。

删除文件找回前提: 删除前, 文件存在时的状态提交到了本地库。

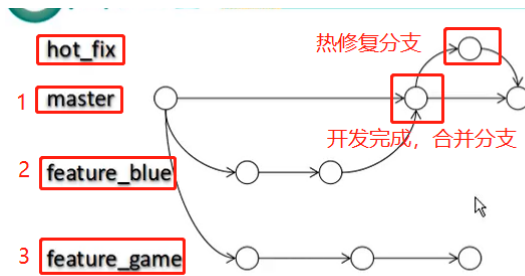
(5) 比较文件的差异

```
git diff [文件名] # diff没有其它参数, 表示和暂存区进行比较。
```

```
git diff [本地库中历史版本][文件名] # 将工作区中的文件和本地库历史记录比较
```

3.1.3 分支管理

(1) 分支介绍



分支好处

- 同时并行推进多个功能开发，提高开发效率
- 各个分支在开发过程中，如果某一个分支开发失败，不会对其他分支有任何影响。失败的分支删除重新开始即可。

(2) 分支操作

- 查看所有分支

```
git brach
```

- 创建新的分支

```
git branch USB # 创建USB分支
git checkout -b USB #创建完成后自动切换到新创建的分支
```

- 切换分支

```
git checkout [branchname] # 切换分支
```

- 合并分支(必须站在接受修改的分支上，如master)

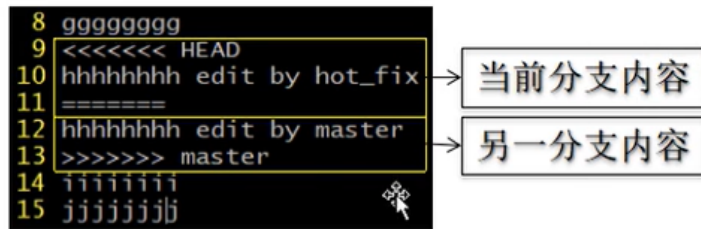
```
# 切换到主分支
git checkout master
# 查看当前所在分支
git branch -v
# 合并分支
git merge [有新内容的分支名]
```

- 解决分支冲突

分支冲突产生原因：

即在主分支和子分支上对同一个文件进行了修改，并且主分支和子分支都提交到本地库成功了。此时若将子分支合并到master主分支，这时会产生分支冲突。

冲突表现



冲突解决方法：

打开产生冲突的文件，删除特殊符号。

`vim [文件名]`

将产生的冲突删除即可。

再次使用

`git add [文件名]`

`git commit -m "提示内容"`

这里的commit命令不能加文件名，否则会报错。

3.2 远程库操作

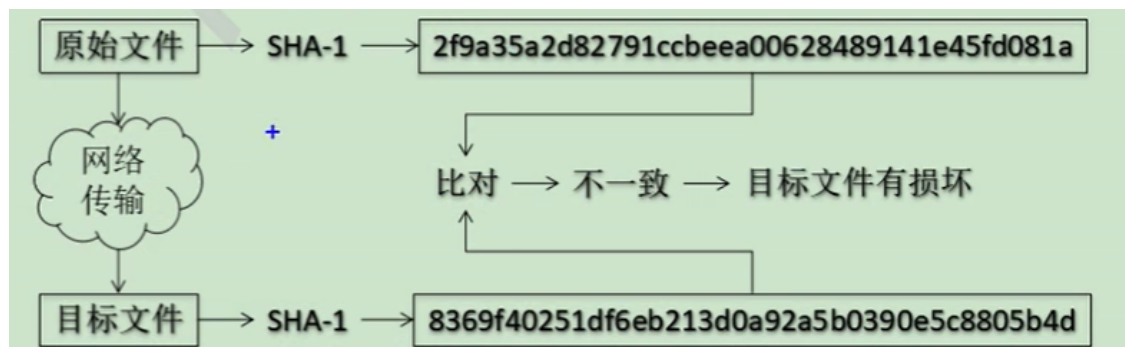
3.2.1 Git 原理

- 哈希（MD5算法属于哈希的一种）

哈希是一个系列的加密算法，各个不同的哈希算法虽然加密强度不同，但是有以下几个共同点：

- ①不管输入数据的数据量有多大，输入同一个哈希算法，得到的加密结果长度固定。
- ②哈希算法确定，输入数据确定，输出数据能够保证不变。
- ③哈希算法确定，输入数据有变化，输出数据一定有变化，而且通常变化很大。
- ④哈希算法不可逆
- ⑤Git底层采用的是SHA-1算法

哈希算法可以被用来验证文件。原理如下图所示：



3.2.2 远程库操作

在Git终端创建完本地库以后，在创建一个远程库（最好库名一样，方便管理。不做强制要求）

远程库在github官网创建。

- 在本地查看git绑定的远程库

```
git remote -v
```

```
ThinkPad@LAPTOP-3FCRPIUN MINGW64 /f/GitHubLocalWareHouse (master)
$ git remote -v
origin https://github.com/yimisiyang/CloudImage.git (fetch) 取回
origin https://github.com/yimisiyang/CloudImage.git (push) 推送
```

- 添加远程库

```
git remote add origin https://github.com/yimisiyang/BigData-java.git
```

- 将本地库推送到远程库

```
git push origin master
git push -u origin master
git push -u origin master -f #强制推送到远程
```

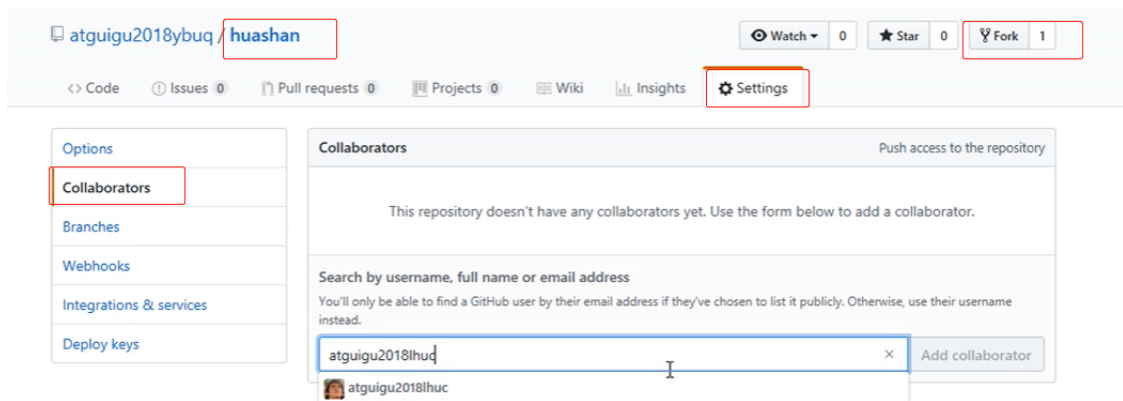
注意： pull=fetch+merge

- 把远程库克隆到本地库

```
git clone https://github.com/yimisiyang/BigData-java.git
```

- (1) 完整的把远程库下载到本地
- (2) 初始化本地库
- (3) 创建origin远程地址别名

只有邀请别人后，别人才能将克隆后的修改推送到远程库。



邀请别人成功后，别人才能将修改的内容推送到你创建的远程库。

- git fetch把远程库抓取下来(只是将远程文件下载到本地，并没有和本地进行合并)

```
git fetch origin master
```

注意： git fetch [远程库地址别名] [远程分支名]

- 查看从远程库抓取下来的内容

```
git checkout origin/master
```

- 把远程master合并到本地master

```
git merge origin/master
```

注意： git merge [远程库地址别名/远程分支名]

- git pull拉去远程库（该操作拉去远程分支后直接与本地分支进行合并）

```
git pull origin master
```

- 团队外的人提交到远程仓库
 - (1) fork 别人的项目
 - (2) 克隆该项目并做修改
 - (3) 推送到自己的远程库
 - (4) 发起pull request申请
 - (5) 别人merge 你的 pull request后就好了。

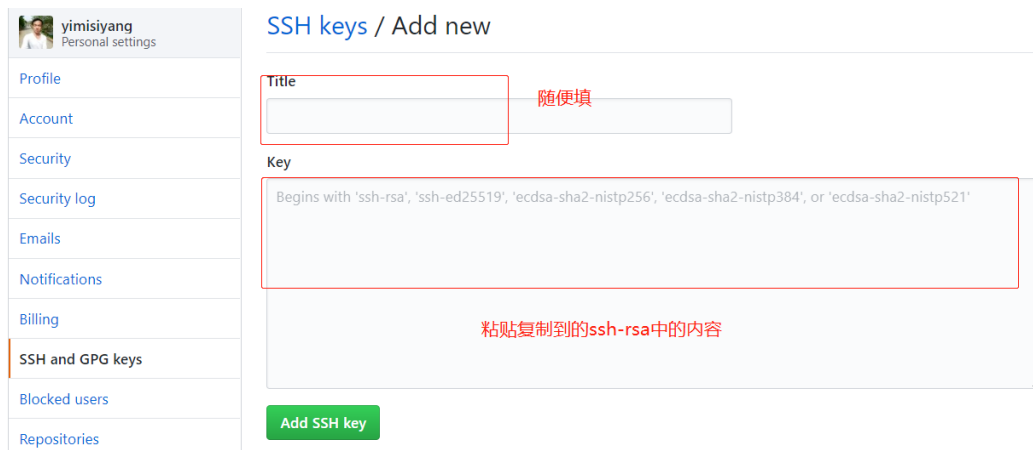
3.2.3 SSH 免密登陆

生成key

```
ssh-keygen -t rsa -C 15230034878@163.com
```

```
Lenovo@DESKTOP-SAV98C0 MINGW64 ~/.ssh
$ ll
total 5
-rw-r--r-- 1 Lenovo 197121 1675 5月 31 17:44 id_rsa
-rw-r--r-- 1 Lenovo 197121 408 5月 31 17:44 id_rsa.pub
```

读取id_rsa中的内容，将内容复制到github中settings的 ssh and GPG keys



新建一个ssh的别名，添加进去

```
git remote add origin_ssh
```

```
Lenovo@DESKTOP-SAV98C0 MINGW64 /d/workspaces/GitSpaceVideo/huashan (master)
$ git remote add origin_ssh git@github.com:atguigu2018ybuq/huashan.git
```

再次推送

```
git push origin_ssh master
```

```
Lenovo@DESKTOP-SAV98C0 MINGW64 /d/workspaces/GitSpaceVideo/huashan (master)
$ git remote -v
origin https://github.com/atguigu2018ybuq/huashan.git (fetch)
origin https://github.com/atguigu2018ybuq/huashan.git (push)
origin_ssh git@github.com:atguigu2018ybuq/huashan.git (fetch)
origin_ssh git@github.com:atguigu2018ybuq/huashan.git (push)

Lenovo@DESKTOP-SAV98C0 MINGW64 /d/workspaces/GitSpaceVideo/huashan (master)
$ git push origin_ssh master
```

4. Git图形化界面操作

所谓图形化界面即在IDE中对git进行设置。

5. GitLab服务器环境搭建

5.1 GitLab 安装（有树莓派版，惊不惊喜）

建议在CentOS7上搭建GitLab服务器。

从官网下载CentOS版本[安装包](#)

官网给的安装过程

9.2 安装命令摘录

```
sudo yum install -y curl policycoreutils-python openssh-server crontime
sudo lokkit -s http -s ssh
sudo yum install postfix
sudo service postfix start
sudo chkconfig postfix on
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.rpm.sh | sudo bash
sudo EXTERNAL_URL="http://gitlab.example.com" yum -y install gitlab-ee
```

实际问题：yum 安装 gitlab-ee(或 ce)时，需要联网下载几百 M 的安装文件，非常耗时，所以应提前把所需 RPM 包下载并安装好。

下载地址为：

```
https://packages.gitlab.com/gitlab/gitlab-ce/packages/el/7/gitlab-ce-10.8.2-ce.0.el7.x86_64.rpm
```

调整后的安装过程

9.3 调整后的安装过程

```
sudo rpm -ivh /opt/gitlab-ce-10.8.2-ce.0.el7.x86_64.rpm
sudo yum install -y curl policycoreutils-python openssh-server crontime
sudo lokkit -s http -s ssh
sudo yum install postfix
sudo service postfix start
sudo chkconfig postfix on
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.sh | sudo bash
sudo EXTERNAL_URL="http://gitlab.example.com" yum -y install gitlab-ce
```

5.2 GitLab 服务操作

装完之后重启计算机，然后进行初始化配置

```
gitlabctl reconfigure
```

注意： 该命令会执行很长时间。

启动gitlab服务

```
gitlabctl start
```

接下来就可以在浏览器访问了。

在浏览器输入IP地址访问。若打不开，检查防火墙配置。

初次登陆需要为gitlab的root用户设置密码。