

git笔记

1.git结构

2.git和dianante托管中心

代码托管中心的任务：维护远程库

2.1. 局域网环境下

- n GitLab 服务器

2.2 外网环境下

- n GitHub
- n 码云

3.本地库和远程库

3.1 团队内部协作

image-20210702104314925

3.2 跨团队协作

4.git命令行操作

4.1 本地库初始化

- 命令：git add
- 效果：

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，也不要胡乱修改。

4.2 设置签名

4.2.1 形式

- 用户名：tom
- Email 地址：

4.2.2 作用

- 区分不同开发人员的身份

4.2.3 辨析

- 这里设置的签名和登录远程库(代码托管中心)的账号、密码没有任何关系。


4.2.4 命令

1 | 项目级别/仓库级别：仅在当前本地库范围内有效

- git **config** user.name tom_pro
- git **config** user.email goodMorning_pro@atguigu.com
- 信息保存位置：./.git/config 文件

```
1 | 系统用户级别：登录当前操作系统的用户范围
```

- `git config --global user.name tom_glb`
- `git config --global`
- 信息保存位置：`~/.gitconfig` 文件

 image-20210702105636645

```
1 | 级别优先级
```

- 就近原则：项目级别优先于系统用户级别，二者都有时采用项目级别的签名
- 如果只有系统用户级别的签名，就以系统用户级别的签名为准

二者都没有不允许

5.基本操作

5.1 状态查看

```
1 | git status
2 |
3 | 查看工作区、暂存区状态
```

5.2 添加

```
1 | git add [file name]
2 |
3 | 将工作区的“新建/修改”添加到暂存区
```

5.3 提交

```
1 | git commit -m "commit message" [file name]
2 |
3 | 将暂存区的内容提交到本地库
```

5.4 查看历史记录

```
1 | git log
```

多屏显示控制方式：

- 空格向下翻页
- b 向上翻页
- q 退出

```
1 | git log --pretty=oneline
```

 image-20210702111112939

```
1 | git log --oneline
```

 image-20210702111135504

1 | `git reflog`

 image-20210702111159407

注: HEAD@{移动到当前版本需要多少步}

5.5 前进后退

本质:

1 | 基于索引值操作[推荐]

- `git reset --hard [局部索引值]`
- `git reset --hard a6ace91`

1 | 使用^符号: 只能后退

- `git reset --hard HEAD^`
- 注: 一个^表示后退一步, n 个表示后退 n 步

1 | 使用~符号: 只能后退

- `git reset --hard HEAD~n`
- 注: 表示后退 n 步

5.6 reset 命令的三个参数对比

1 | `--soft` 参数

- 仅仅在本地库移动 HEAD 指针

 image-20210702111721116

1 | `--mixed` 参数

- 在本地库移动 HEAD 指针
- 重置暂存区

 image-20210702111822021

1 | `--hard` 参数

- 在本地库移动 HEAD 指针
- 重置暂存区
- 重置工作区

5.7 删除文件并找回

5.7.1 前提

删除前，文件存在时的状态提交到了本地库。

5.7.2 操作

`git reset --hard [指针位置]`

- 删除操作已经提交到本地库：指针位置指向历史记录
- 删除操作尚未提交到本地库：指针位置使用 HEAD

5.8 比较文件找回

```
1  git diff [文件名]
2
3  * 将工作区中的文件和暂存区进行比较
4
5  git diff [本地库中历史版本] [文件名]
6
7  * 将工作区中的文件和本地库历史记录比较
8
9  不带文件名比较多个文件
```

6.分支管理

6.1 什么是分支？

在版本控制过程中，使用多条线同时推进多个任务（也就是多个功能开发，就后再合并）

6.2 分支的好处？

```
1  * 同时并行推进多个功能开发，提高开发效率
2
3  * 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。
```

6.3 分支的操作

```
1  创建分支
```

- `git branch [分支名字]`

```
1  查看分支
```

- `git branch -v`

```
1  切换分支
```

- `git checkout [分支名字]`

```
1  合并分支
```

- 第一步：切换到主分支上
`git checkout [主分支名字]`
- 第二步：执行merge命令
`git merge [要合并的分支名字]`

1 | 解决冲突（就是分支上同时修改相同的内容导致冲突，要手动合并）

- 冲突的表现



- 冲突的解决
 - 第一步：把文件修改到满意的程度，保存退出
 - 第二步：`git add [文件名]`
 - 第三步：`git commit -m "日志信息"`
 - 注意：此时 `commit` 一定不能带具体文件名

7.git基本原理

7.1 哈希



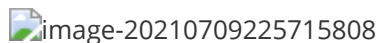
哈希是一个系列的（比如文档、图片、音频.....）加密算法，各个不同的哈希算法虽然加密强度不同，但是有以下几个共同点：

- 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

Git 底层采用的是 SHA-1 算法。

哈希算法：1.MD5 2.SHA1 3.CRC32

哈希算法可以被用来验证文件。原理如下图所示：



Git 就是靠这种机制来从根本上保证数据完整性的。

7.2 保存版本的机制

7.2.1 集中式版本控制工具的文件管理机制

以文件变更列表的方式存储信息。这类系统将它们保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。

7.2.2 git的文件管理机制

Git 把数据看作是小型文件系统的一组快照。每次提交更新时 Git 都会对当前的全部文件制作一个快照并保存这个快照的索引。为了高效，如果文件没有修改，

Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。所以 Git 的工作方式可以称之为快照流。

 image-20210709225835497

7.2.3 git文件管理机制细节

git的“提交对象”

 image-20210709225910248

 img

提交对象及其父对象形成的链条

 image-20210709225945980

8.git分支管理机制

8.1 分支的创建

 image-20210710155044190

8.2 分支的切换

 image-20210710155051489

 image-20210710155059421

 image-20210710155112426

 image-20210710155127832

9.GitHub

9.1 账号信息

GitHub 首页就是注册页面: <https://github.com/>

注册邮箱

github@ccy.com

2967911782@qq.com

element-ui

3github@ccy.com³

token值: ghp_pWv6eBbMY8IUig8B3pUmnYQGawI8UU02LqtL

 image-20210710200938765

9.2 创建远程库

9.3 创建远程库地址别名

找到新建的仓库, 有个远程库的地址, 根据这个地址, 可以找到远程库

 image-20210710184540784

git remote -v 查看当前所有远程地址别名

git remote add [别名] [远程地址]

9.4 推送

git push [别名] [分支名]

如果出现问题，不能传送到远程库的话，就把别名都删了，重新设置别名

解决办法的地址：<https://blog.csdn.net/rosecurry/article/details/93855636>

git remote add [别名] [远程地址]

或者是账号密码错误

或者是网络的问题git



image-20210711230918556

如果出现这样的问题，就是在项目中修改的内容，没有保存到本地库中的原因



正常的结果



image-20210711221533374

9.5 克隆

cd .. 退出当前的操作

git clone [远程地址]

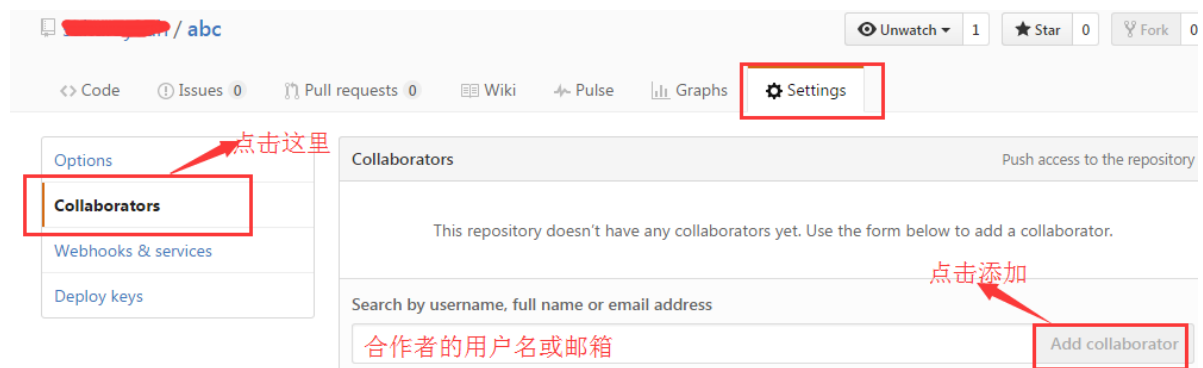


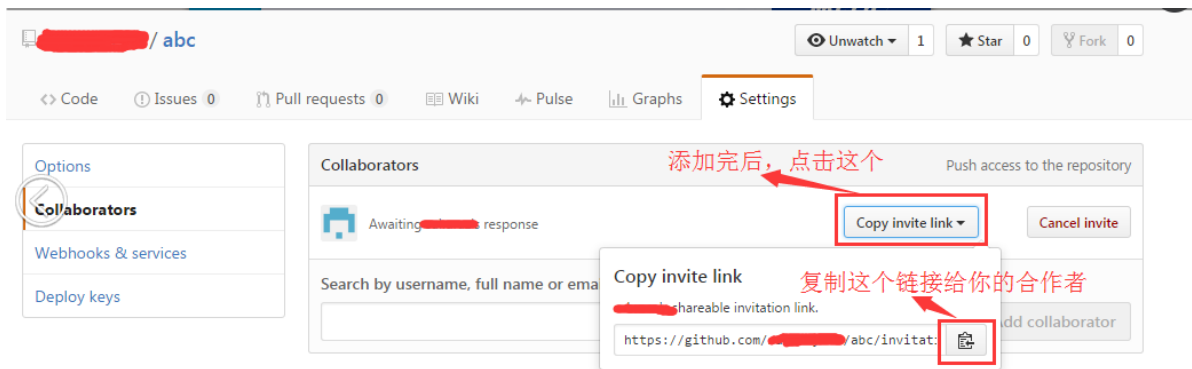
- 效果
 - 完整的把远程库下载到本地
 - 创建 origin 远程地址别名
 - 初始化本地库

9.6 团队成员邀请



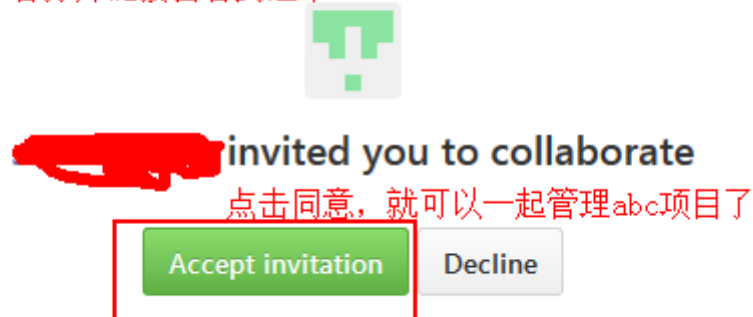
image-20210711223712183



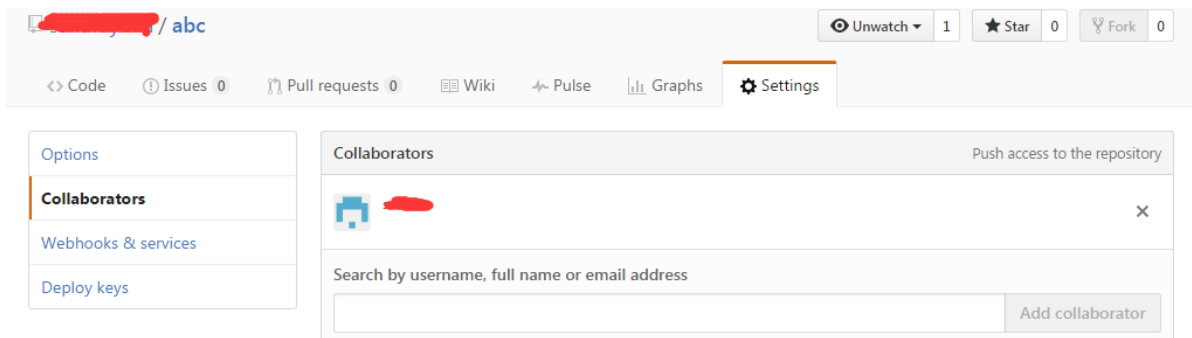


复制链接后，发给合作者，合作者在自己的账号网页中的url窗口打开收到的链接，打开后，如下图所示

合作者打开链接会看到这个



对方同意了之后，下图所示



9.7 拉取


pull=fetch+merge

- | | |
|-----------------------------|---------------------------|
| git fetch [远程库地址别名] [远程分支名] | 抓取下来，就是把合作者改过的项目拿过来看看 |
| git merge [远程库地址别名/远程分支名] | 看完合作者修改后的项目没问题，就使用merge合并 |
| git pull [远程库地址别名] [远程分支名] | 也可以使用这个，代替fetch和merge |

9.8 协同开发时冲突的解决

就是同时在文件中相同的位置修改内容

- 要点
 - 如果不是基于 GitHub 远程库的最新版所做的修改，不能推送，必须先拉取。
 - 拉取下来后如果进入冲突状态，则按照“分支冲突解决”操作解决即可。

 image-20210703102923482

冲突的解决

- 第一步：把文件修改到满意的程度，保存退出

- 第二步: git add [文件名]
- 第三步: git commit -m "日志信息"
- 注意: 此时 commit 一定不能带具体文件名
- 类比
 - 债权人: 老王
 - 债务人: 小刘

老王说: 10 天后归还。小刘接受, 双方达成一致。

老王媳妇说: 5 天后归还。小刘不能接受。老王媳妇需要找老王确认后再执行。

9.9 跨团队协作

fork (将自己的远程地址发给向他人请教的人)

image-20210713002850088

image-20210713002901463

image-20210713002915469

第一步: 收到远程库的地址后, 在请教的人中的账号的网页中打开

第二步: 复制自己的远程库地址

第三步: clone下来 git clone [复制下来的远程地址]

第四步: 修改/操作本地的内容

第五步: 修改完后, 就推送远程库

第六步: pull Request

image-20210713003822543

image-20210713003834368

image-20210713003842482

可以发消息, 请教人的操作已完成

image-20210713003904186

主远程库的主人, 接收到请教人发过来的信息

image-20210713004054534

image-20210713004104073

之间可以进行对话操作

image-20210713004130283

image-20210713004138174

第七步: 审核代码

image-20210713004156659

第八步: 审核完代码后, 进行代码合并

image-20210713004226607

第九步: 合并失败/成功的结果



第十步：前面的步骤实现后，还是看不到请教人修改后的内容，需要拉取到本地库，才能看到请教人修改后的内容



9.10 SSH登录

- 进入当前用户的家目录

```
$ cd ~
```

- 删除.ssh 目录

```
$ rm -rvf .ssh
```

- 运行命令生成.ssh 密钥目录

```
$ ssh-keygen -t rsa -C atguigu2018ybuq@aliyun.com (登录邮箱)
```

[注意：这里-C 这个参数是大写的 C]

- 进入.ssh 目录查看文件列表

```
$ cd .ssh
```

```
$ ls -lF
```

- 查看 id_rsa.pub 文件内容

```
$ cat id_rsa.pub
```



- 复制 id_rsa.pub 文件内容，登录 GitHub，点击用户头像→Settings→SSH and GPG keys
- 点击New SSH Key



- 输入复制的密钥信息



- 回到 Git bash 创建远程地址别名

```
git remote add origin_ssh git@github.com:atguigu2018ybuq/huashan.git
```

- 推送文件进行测试

10.Eclipse操作

10.1 工程初始化为本地库

10.2 Eclipse中忽略文件

10.3 推送到远程库

10.4 Oxygen Eclipse 克隆工程操作

10.5 Kepler Eclipse 克隆工程操作

10.6 解决冲突

11. 工作流

11.1 概念

在项目开发过程中使用 Git 的方式

11.2 分类

11.2.1 集中式工作流


像 SVN 一样，集中式工作流以中央仓库作为项目所有修改的单点实体。所有修改都提交到 Master 这个分支上

这种方式与SVN 的主要区别就是开发人员有本地库。Git 很多特性并没有用到

 image-20210713011112279

11.2.2 gitFlow 工作流

Gitflow 工作流通过为功能开发、发布准备和维护设立了独立的分支，让发布迭代过程更流畅。严格的分支模型也为大型项目提供了一些非常必要的结构

 image-20210713011125184

11.2.3 ForKing 工作流

Forking 工作流是在 GitFlow 基础上，充分利用了 Git 的 Fork 和 pull request 的功能以达到代码审核的目的。更适合安全可靠地管理大团队的开发者，而且能接受不信任贡献者的提交。

 image-20210713011140380

11.3 gitFlow 工作流详解

11.3.1 分支种类

- 主干分支 master

主要负责管理正在运行的生产环境代码。永远保持与正在运行的生产环境 完全一致。

- 开发分支 develop

主要负责管理正在开发过程中的代码。一般情况下应该是最新的代码。

- bug 修理分支 hotfix

主要负责管理生产环境下出现的紧急修复的代码。从主干分支分出，修理完毕并测试上线后，并回主干分支。并回后，视情况可以删除该分支。

- 准生产分支（预发布分支） release

较大的版本上线前，会从开发分支中分出准生产分支，进行最后阶段的集成测试。该版本上线后，会合并到主干分支。生产环境运行一段阶段较稳定后可以视情况删除。

- 功能分支 feature

为了不影响较短周期的开发工作，一般把中长期开发模块，会从开发分支中独立出来。开发完成后会合并到开发分支。

11.3.2 gitFlow 工作流举例

11.3.3 分支实战



11.3.4 具体操作

- 创建分支



- 切换分支审查代码



检出远程新分支



切换回master



合并分支



合并结果



合并成功后，把 master 推送到远程

12. gitlab 服务器搭建过程

12.1 官网地址

首页: <https://about.gitlab.com/>

安装说明: <https://about.gitlab.com/installation/>

12.2 安装命令摘录

```
sudo yum install -y curl policycoreutils-python openssh-server cronie
```

```
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

12.3 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 cronie
```

```
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
```

当前步骤完成后重启

12.4 gitlab 服务器操作

- 初始化配置

```
gitlab gitlab-ctl reconfigure
```

- 启动 gitlab 服务

```
gitlab-ctl start
```

- 停止 gitlab 服务

```
gitlab-ctl stop
```

12.5 浏览器访问

访问 Linux 服务器 IP 地址即可，如果想访问 EXTERNAL_URL 指定的域名还需要配置域名服务器或本地 hosts 文件。

初次登录时需要为 gitlab 的 root 用户设置密码

image-20210714231631074

root/atguigu2018good

应该会需要停止防火墙服务：

service firewalld stop

微信号：creathinFeng

13.命令符

| Git 常用命令速查表 | | master :默认开发分支 | Head :默认开发分支 |
|-----------------------------------|----------------------|---|------------------|
| | | origin :默认远程版本库 | Head^ :Head 的父提交 |
| | | | |
| 创建版本库 | | 分支与标签 | |
| \$ git clone <url> | #克隆远程版本库 | \$ git branch | #显示所有本地分支 |
| \$ git init | #初始化本地版本库 | \$ git checkout <branch/tag> | #切换到指定分支或标签 |
| 修改和提交 | | \$ git branch <new-branch> | #创建新分支 |
| \$ git status | #查看状态 | \$ git branch -d <branch> | #删除本地分支 |
| \$ git diff | #查看变更内容 | \$ git tag | #列出所有本地标签 |
| \$ git add . | #跟踪所有改动过的文件 | \$ git tag <tagname> | #基于最新提交创建标签 |
| \$ git add <file> | #跟踪指定的文件 | \$ git tag -d <tagname> | #删除标签 |
| \$ git mv <old> <new> | #文件改名 | 合并与衍合 | |
| \$ git rm <file> | #删除文件 | \$ git merge <branch> | #合并指定分支到当前分支 |
| \$ git rm --cached <file> | #停止跟踪文件但不删除 | \$ git rebase <branch> | #衍合指定分支到当前分支 |
| \$ git commit -m "commit message" | #提交所有更新过的文件 | 远程操作 | |
| \$ git commit --amend | #修改最后一次提交 | \$ git remote -v | #查看远程版本库信息 |
| 查看提交历史 | | \$ git remote show <remote> | #查看指定远程版本库信息 |
| \$ git log | #查看提交历史 | \$ git remote add <remote> <url> | #添加远程版本库 |
| \$ git log -p <file> | #查看指定文件的提交历史 | \$ git fetch <remote> | #从远程库获取代码 |
| \$ git blame <file> | #以列表方式查看指定文件的提交历史 | \$ git pull <remote> <branch> | #下载代码及快速合并 |
| 撤销 | | \$ git push <remote> <branch> | #上传代码及快速合并 |
| \$ git reset --hard HEAD | #撤销工作目录中所有未提交文件的修改内容 | \$ git push <remote> :<branch/tag-name> | #删除远程分支或标签 |
| \$ git checkout HEAD <file> | #撤销指定的未提交文件的修改内容 | \$ git push --tags | #上传所有标签 |
| \$ git revert <commit> | #撤销指定的提交 | | |

vim 查看文件的内容

: wq 退出查看文件的操作