

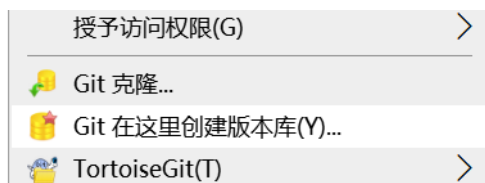
6.Git 的基本使用01-TortoiseGit 操作本地仓库

6.1 初始化仓库

方法一：

新建一个文件夹,进入文件夹内部操作

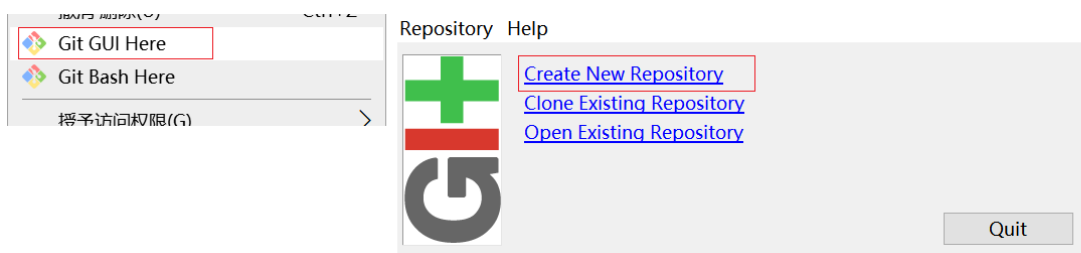
1)右键--> 在这里创建Git 版本库



注意: 不要直接在桌面上操作,否则桌面就是一个仓库

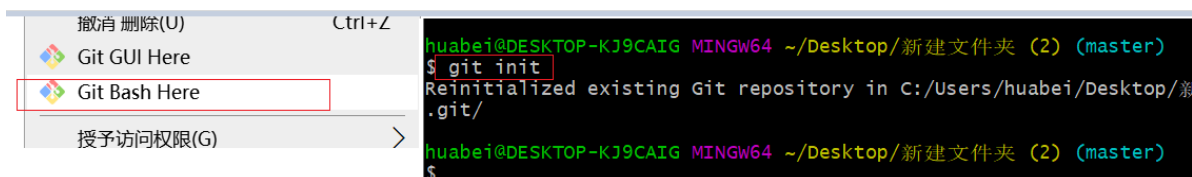
方法二：

2) 右键-->Git GUI here



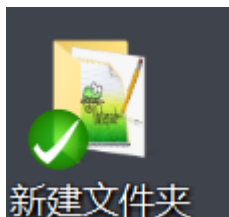
方法三：命令行模式

2) git init



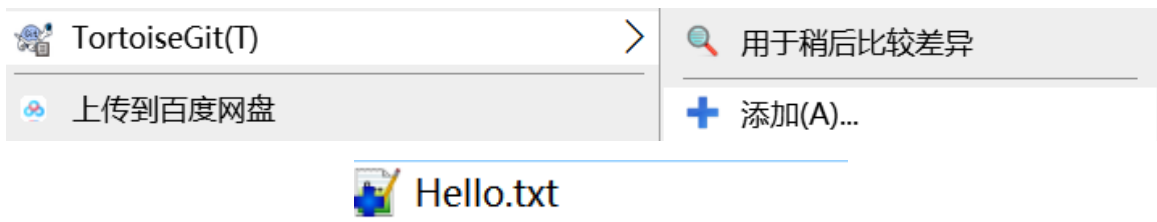
创建完毕仓库,我们发现,此时我们创建的文件夹下有一个.git 文件已经生成了

并且仓库文件夹上多了一个 绿色图标



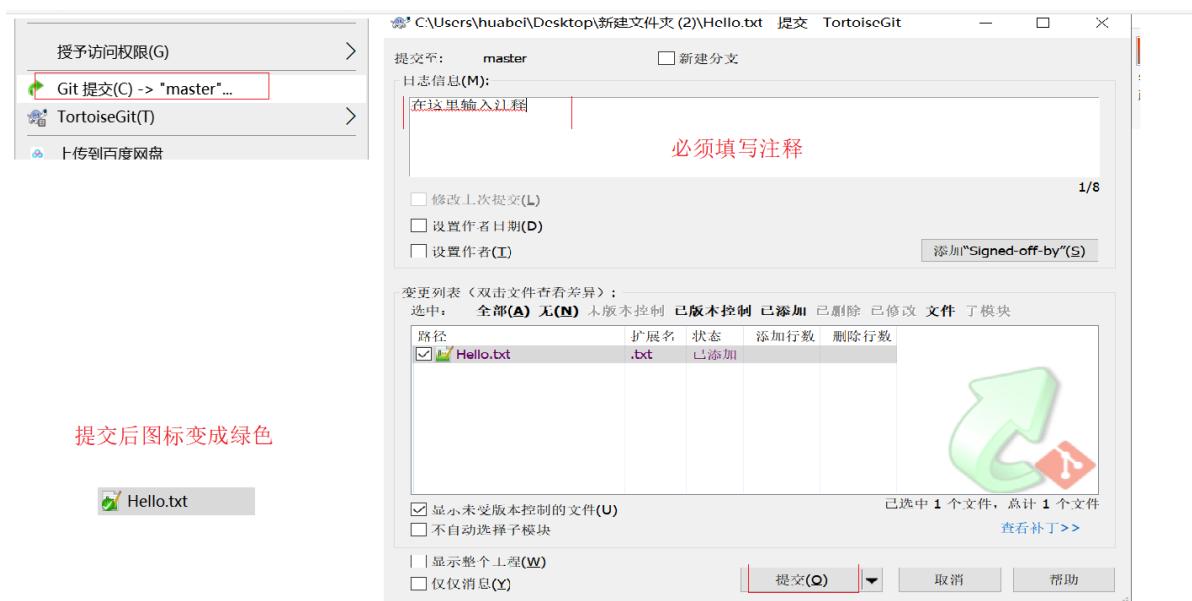
6.2 添加文件

- 1) 在仓库中新建一个文件
- 2) 选中新建的文件-->右键--> TortoiseGit--> 添加
- 3) 此时我们看到文件夹上多了一个 "加号"



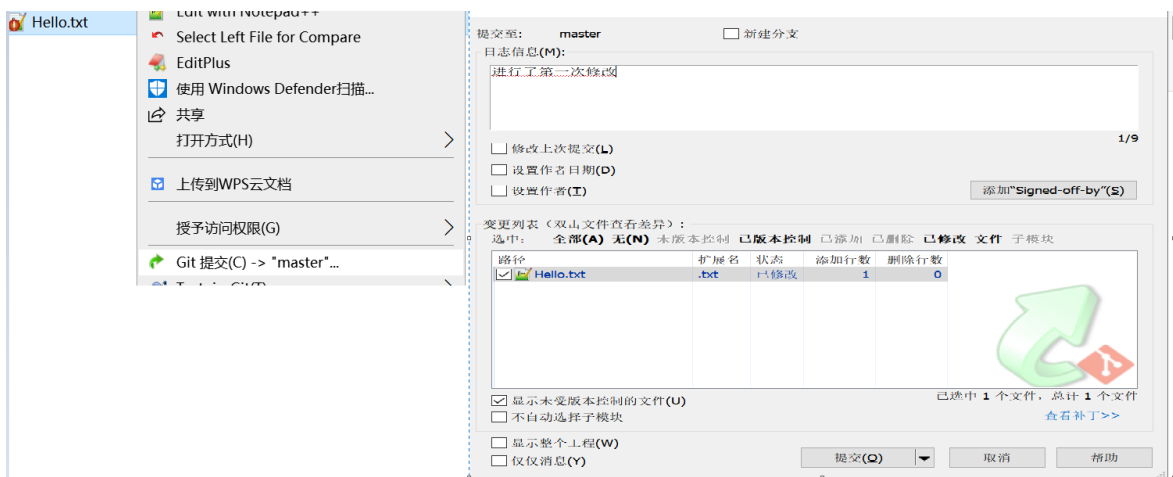
6.3 提交文件至本地仓库

- 1) 选中文件
- 2) 右键--git提交



6.4 修改文件,与再次提交文件

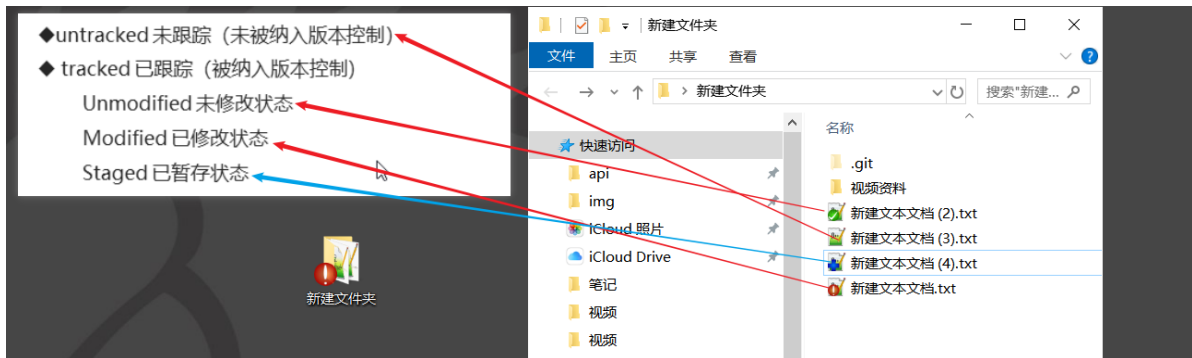
当我们修改文件以后,文件上多了一个红色感叹号,表示我们上次提交后该文件被修改过
提交后文件图标又变成绿色



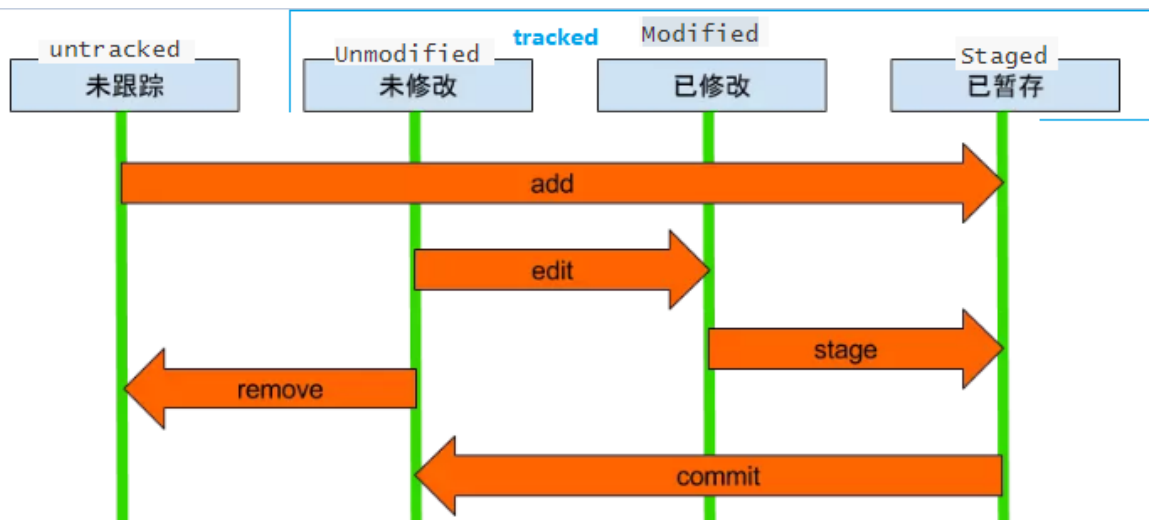
6.5 文件状态讲解

Git工作目录下的文件存在两种状态:

- 1 **untracked** 未跟踪 (未被纳入版本控制) : 比如新建的文件(此时文件夹上没有图标或者有一个"问号")
- 2 **tracked** 已跟踪 (被纳入版本控制)
 - 2.1 **Staged** 已暂存状态 : 添加 但未提交状态(此时文件夹上有一个"加号")
 - 2.2 **Unmodified** 未修改状态 : 已提交(此时文件夹上有一个"对号")
 - 2.3 **Modified** 已修改状态 : 修改了,但是还没有提交 (此时文件夹上有一个"红色感叹号")



这些文件的状态会随着我们执行Git的命令发生变化



6.6 修改文件,不提交和上一个版本比较差异(diff)

修改文件,此时不要提交

选中文件-->右键--> TortoiseGit--> 比较差异



6.7 查看提交历史记录

选中文件

右键--> TortoiseGit--> 显示日志

此时我们可以看到所有的历史提交记录

版本树	操作	信息	作者	日期
		工作树变更	itcast itcast	2019-10-27 16:4...
		master 进行了第一次修改 在这里输入注释		2019-10-27 16:34:29

SHA-1: f841e5945b0d5ecffb8722fdf70e2a7afdad6185

* 进行了第一次修改

备注信息

时间

你是否还记得安装软件时填写的用户名呢?
翻看安装软件那一章节

##6.8 回退至历史版本

右键--> TortoiseGit--> 显示日志

选中某个版本--> 进行如下操作

重置

重置当前分支从master到

☐ 分支(B)

☐ 标签(I)

☒ 提交(C) c630024d9103f70940b89bfb6a9b5acc3b9a1466

重置类型

☐ 软重置(S): 不更改工作区和索引

☐ 混合(M): 保持工作区不变, 重置索引文件

☒ 硬重置(H): 重置工作区和索引 (丢弃所有本地变更)

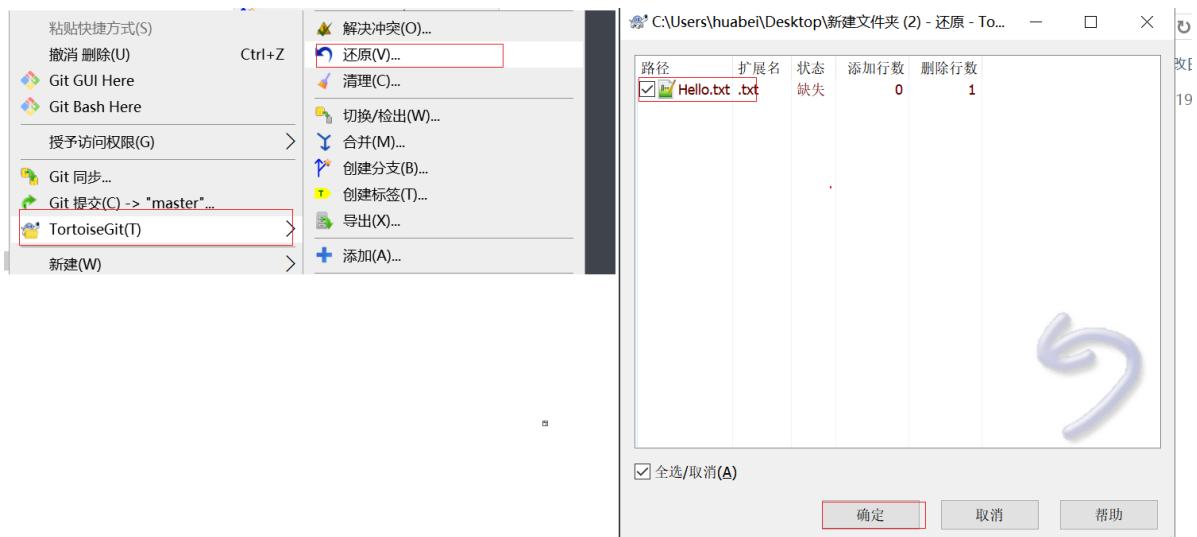
显示工作树中修改的文件(H)

确定 取消 帮助

6.9 文件删除

###6.9.1本地删除与恢复

- 1) 直接选中文件删除的话,其实只是删除了本地工作区的文件,并没有删除 仓库中的文件
此时可以回退的, 比如我们进行如下操作
- 1) 文件删除
- 2) 右键--> TortoiseGit--> 还原
此时我们发现文件又被恢复了

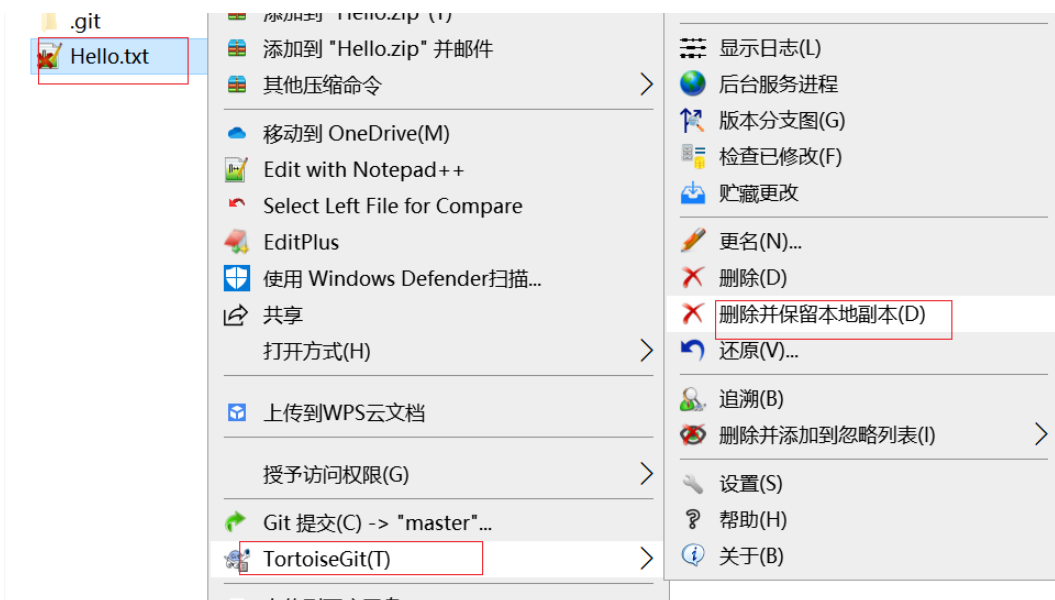


6.9.2从版本库删除

- 2) 我们如果真的想要将某个文件从服务器删除,需要进行如下操作
 - 1) 删除文件,和上面的操作一样
 - 2) 提交,此时服务文件已经删除了(历史版本还在,还是可以恢复)

6.9.3从版本库删除,但是不删除本地

我们可以如下操作,只删除服务器上的文件,但是本地文件并不删除
备注: 删除之后需要提交,才会真正的从服务器删除



6.10 忽略提交

有时候我们一些文件是不需要提交的比如说idea/eclipse 开发的代码自动生成的配置文件
如何配置不提交呢



此时我们的根目录下会生成一个.gitignore 文件

忽略文件如何阅读,常见格式

```
# 所有以.a 结尾的文件讲被忽略(递归)
*.a
# 不管其他规则怎样,强制不忽略 lib.a
!lib.a
# 只忽略 文件 TODO (注意这里是文件)
/TODO
# 忽略 build文件夹下所有内容(递归) 这里是文件夹
build/
# 忽略 doc 目录下以 *.txt 结尾的文件 (不递归)
doc/*.txt
# 忽略 doc 目录下以 *.pdf 结尾的文件 (递归)
doc/**/*.*pdf
```

当然理解了上述规则,我们也可以手动编辑该文件,而不用通过窗口化操作(如果不嫌麻烦)

7. Git 的基本使用02-TortoiseGit 操作本地仓库(分支)

7.1 分支的概念

几乎所有的版本控制系统都以某种形式支持分支。使用分支意味着你可以把你的工作从开发主线上分离开来，避免影响开发主线。多线程开发,可以同时开启多个任务的开发,多个任务之间互不影响。

7.2 为何要使用分支

先看单线程开发



思考如下现象

10.1 日 业务部门提出需求：明年元旦3天做2个促销活动

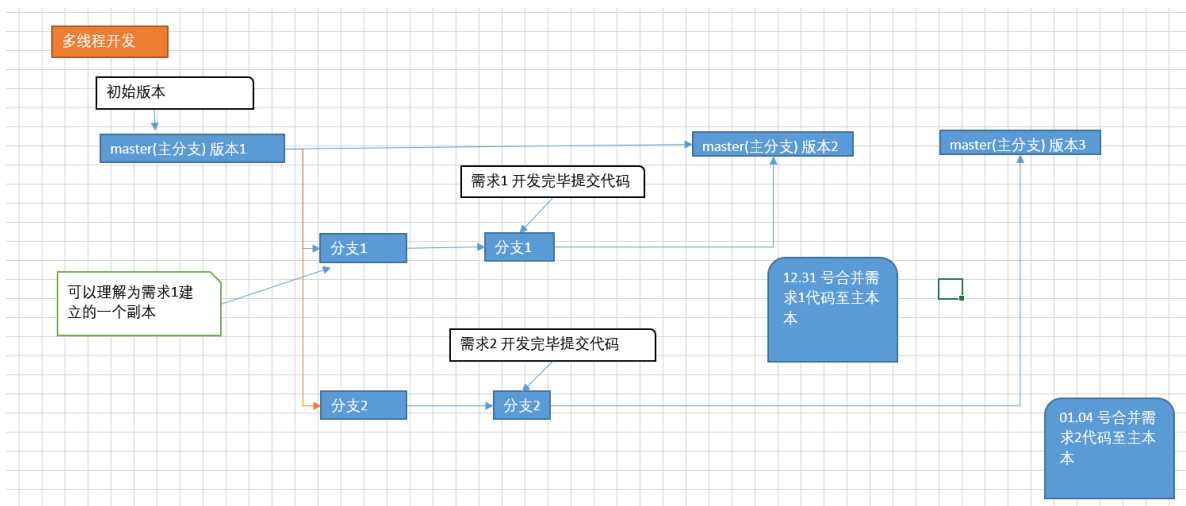
1) 12.31 号上线活动1，

2) 1.4 号上线活动2，同时 要求撤销 活动1

你所在 部门领导 为了保证能顺利完成,要求 11.15 号完成 上述连个功能的开发工作

此时作为开发人员:我要面临两个文件，活动1 的代码,即要存在(12.31 要用)又要不存在(1.4 号要求删除)，我们怎么做？

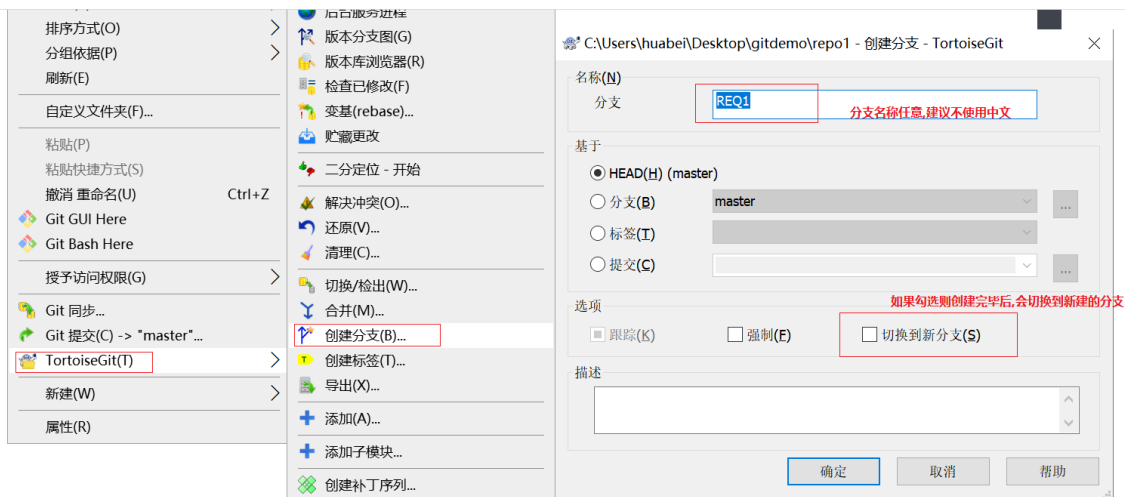
显然比较棘手,如果使用分支(可以理解为将代码复制一份)将很好解决



7.3 创建分支

到现在为止,我们一直使用的时主分支(master)

在主分支上操作创建分支



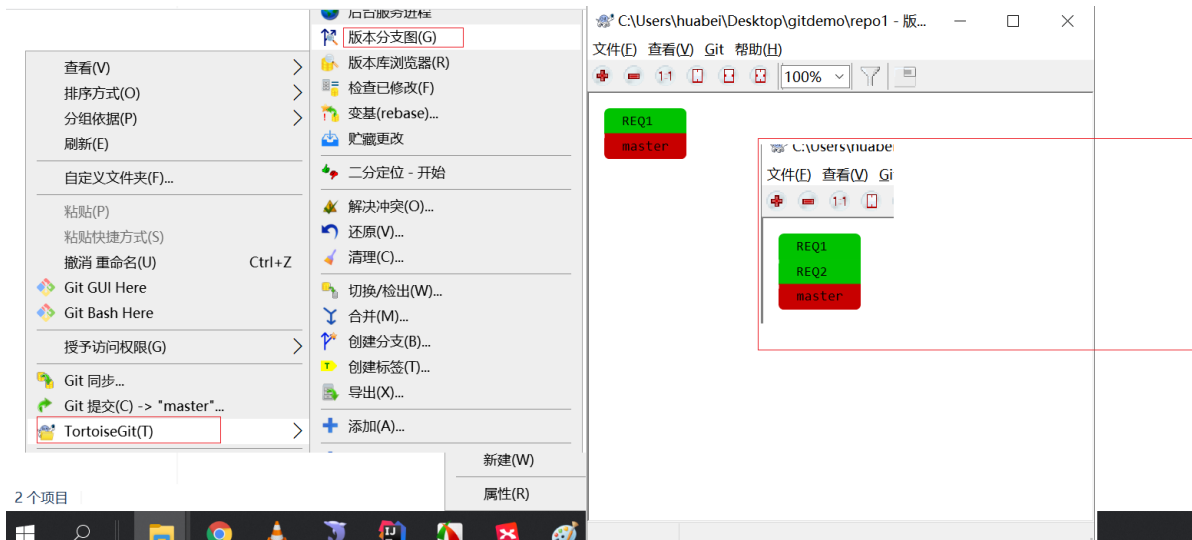
7.4 分支的查看切换

7.4.1查看分支

查看版本分支图,此时我们看到有两个分支

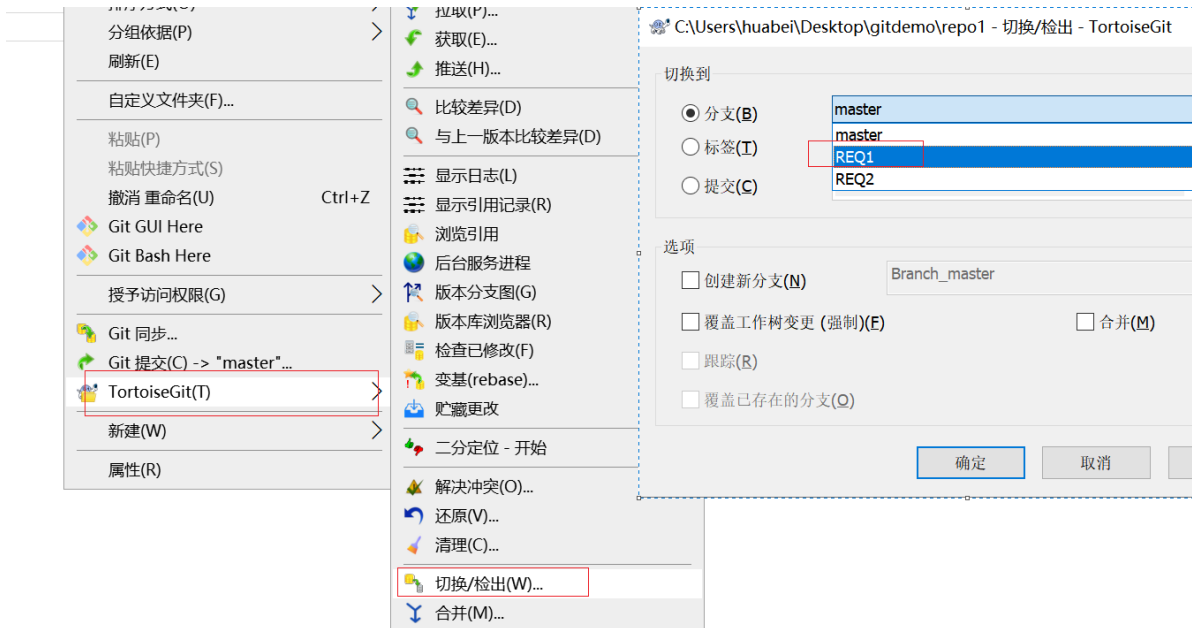
当然,我们可以创建多个分支

可以看到多个分支的图形



7.4.2 切换分支

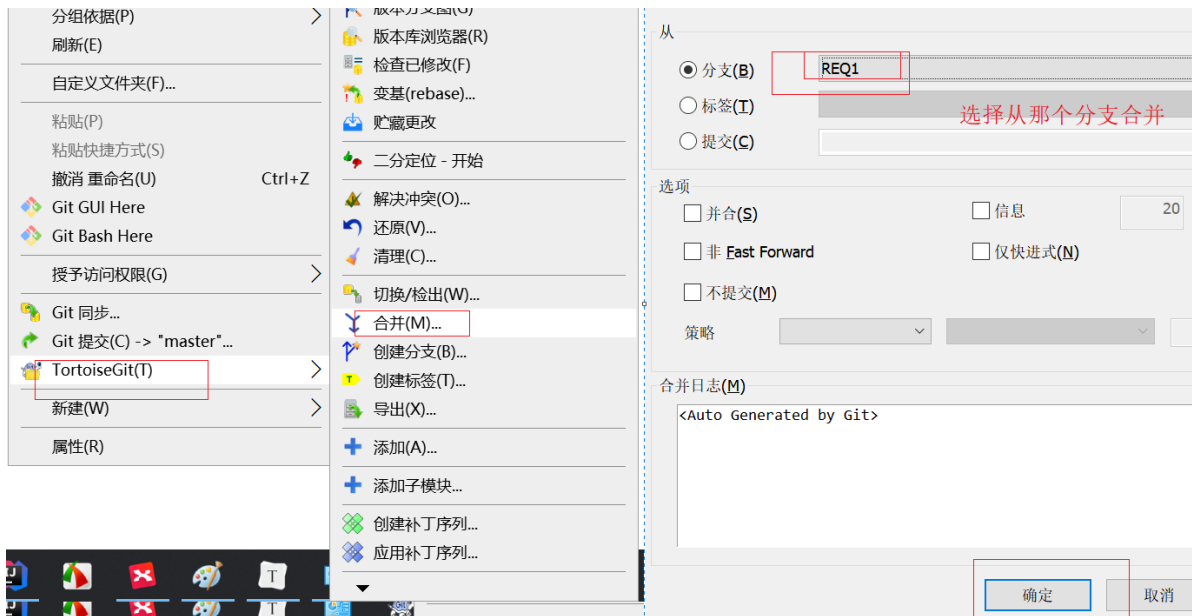
右键--> 检出



##7.5 分支的合并与删除

7.5.1 合并

我们将代码切换到分支1,然后写属于需求1 的代码并提交
当我们把需求1 开发完毕如何把需求1 的代码合并到主分支呢?
-->1 切换到 主版本
-->2 右键 合并即可将需求1 写的代码合并至主分支
-----此时我们看到代码自动合并到了**master**分支



7.5.2删除分支



5.冲突的处理

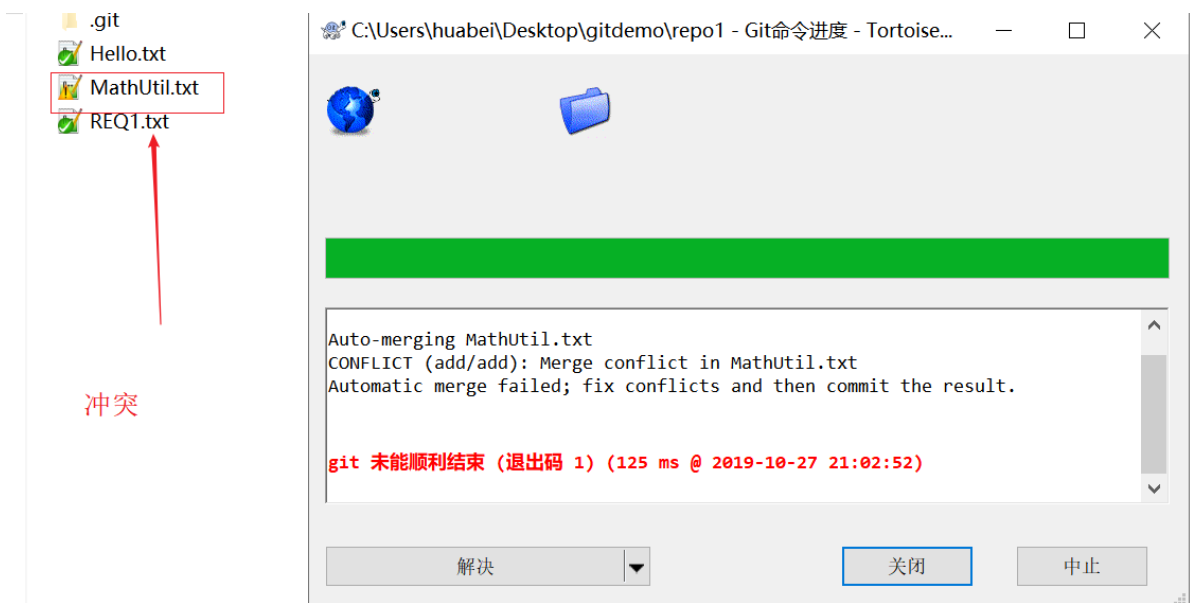
5.1)冲突的概念

现象演示

开发人员A 开发需求1,开发了一个工具类 `MathUtil`,里面第一行写了一个方法 `add(int [] args)`
 同时开发人员B 开发需求2,开发了一个工具类 `MathUtil`,里面第一行写了一个方法 `add(int a int b)`
 他们在互相不知道对方需求的情况下同时提交了代码到自己的分支

思考此时如果我们把需求1 和需求2 同时都合并到主分支上,主分支的 工具类 `MathUtil` 的第一行应该使用谁的代码?

此时主分支是不能智能判断第一行使用谁的代码,合并时会报错,我们叫做冲突。



5.2) 如何处理冲突

分析一下冲突的原因：

开发人员之间彼此没有沟通导致的同一个时间节点修改了同一个地方的代码,合并是冲突

思考：

我们能直接把某个开发人员开发的代码删除吗？

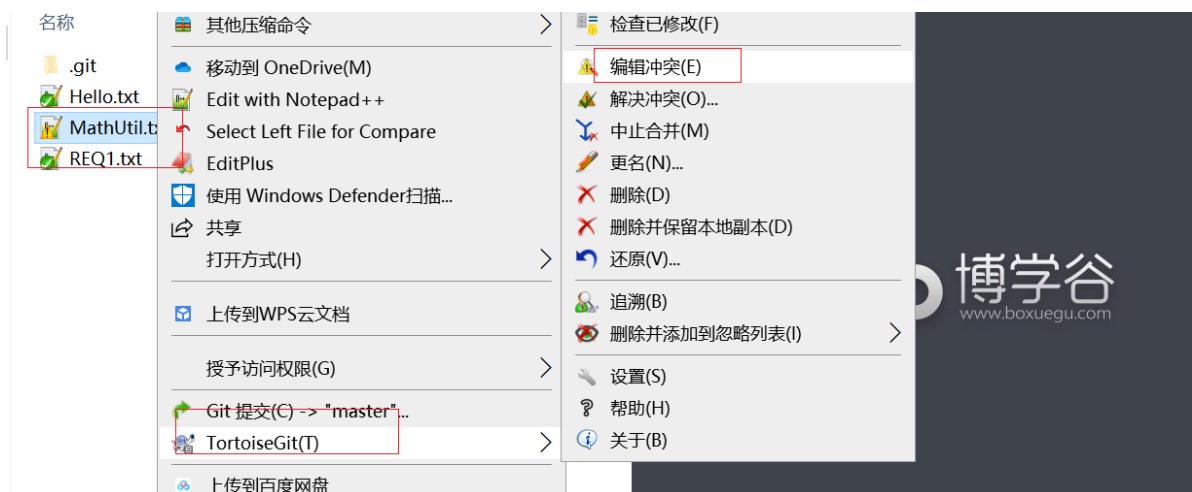
显然不能

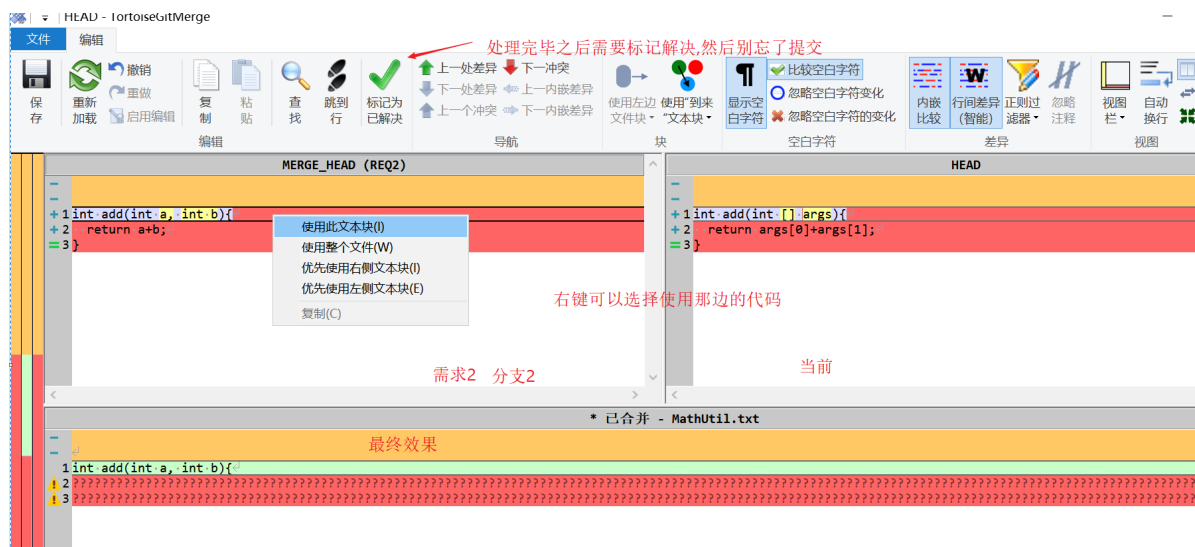
所以在处理冲突时,第一步应该时找开发另一个需求的人员沟通,之后才是处理冲突

--> 选中冲突的文件(带黄色感叹号的文件都是冲突的文件,如果有多个需要逐一处理)

--> 右键--> 编辑冲突,

-->处理完毕后,标记已解决





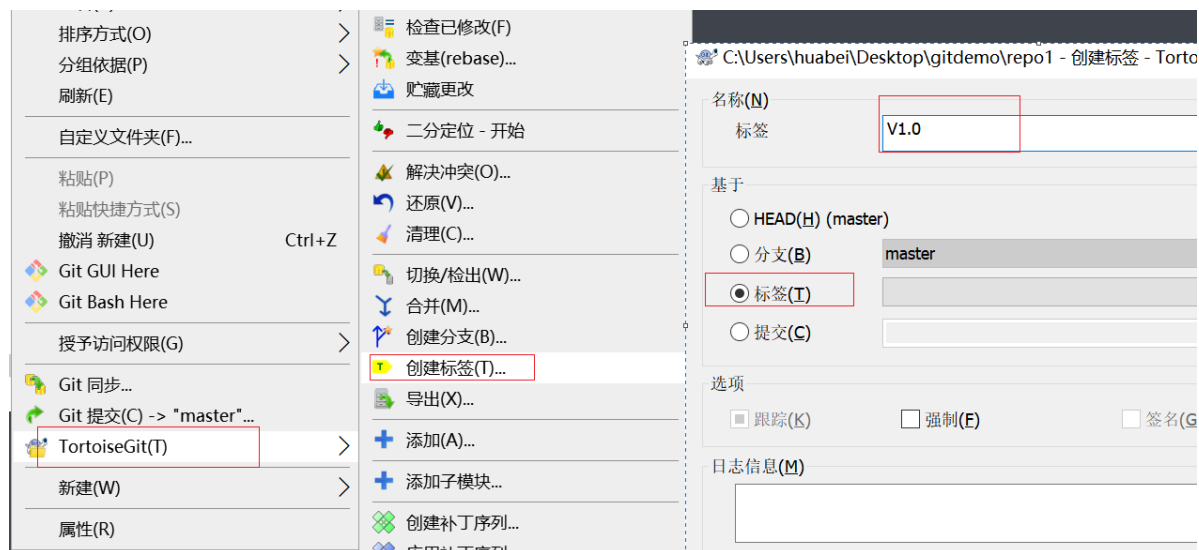
8.tag 标签

8.1 标签的概念

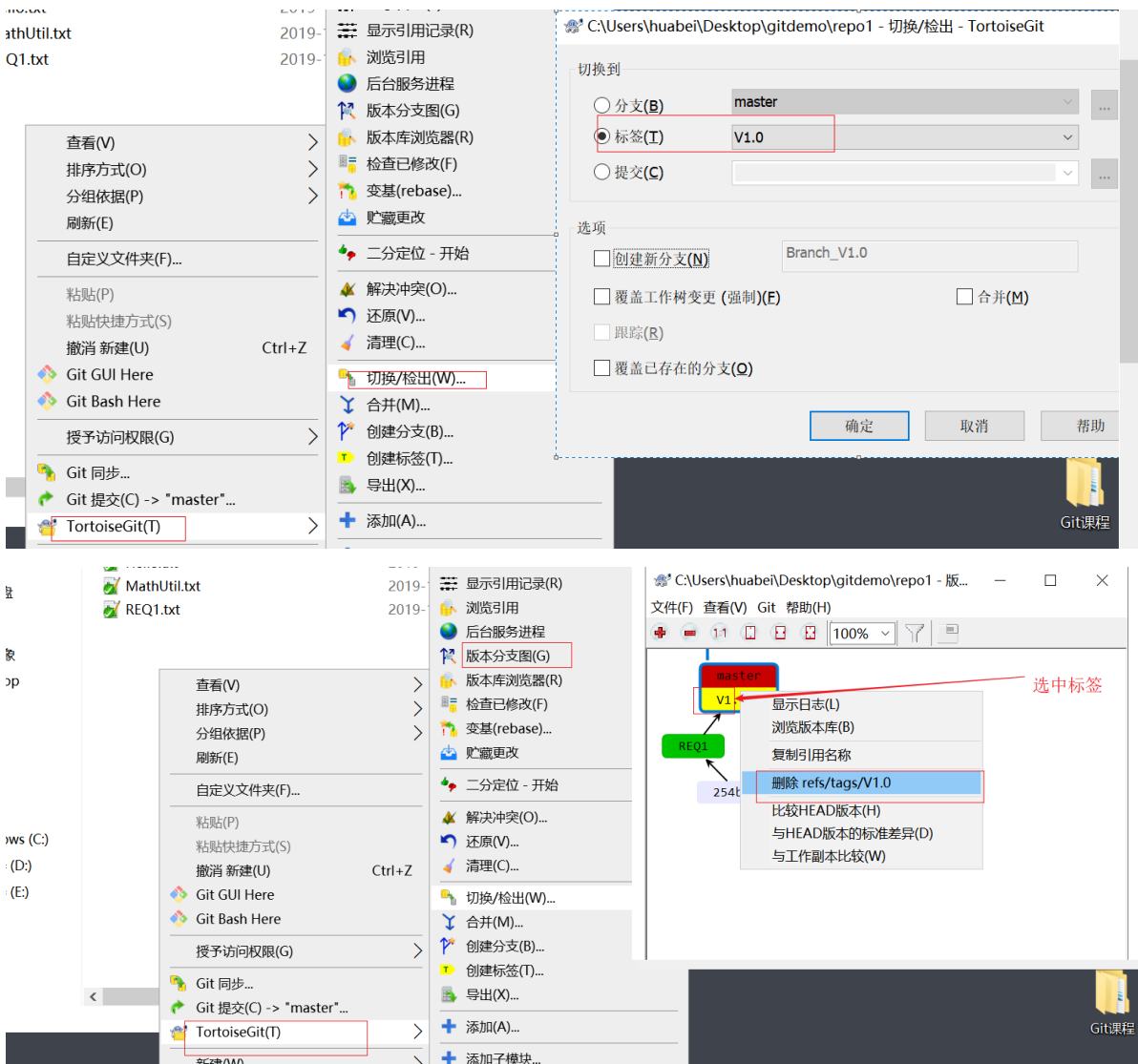
如果你的项目达到一个重要的阶段,并希望永远记住那个特别的提交快照,你可以给它打上标签(tag) 比如说,我们想为我们的项目发布一个"1.0"版本。 我们给最新一次提交打上(HEAD)"v1.0"的标签。 标签可以理解项目里程碑的一个标记,一旦打上了这个标记则,表示当前的代码将不允许提交

8.2 标签的创建(tag)

标签的创建和分支的创建操作几乎一样



8.3 标签的切换与删除



9. 远程仓库

我们的代码不能总是放在本地,因为总是放在本地,一旦电脑出现故障,数据将丢失,怎么共享呢,这里我们需要一个服务器,我们可以把代码放到服务器上,然后让别人下载,这样我峨嵋你既可以备份代码,也可以进行团队协作开发

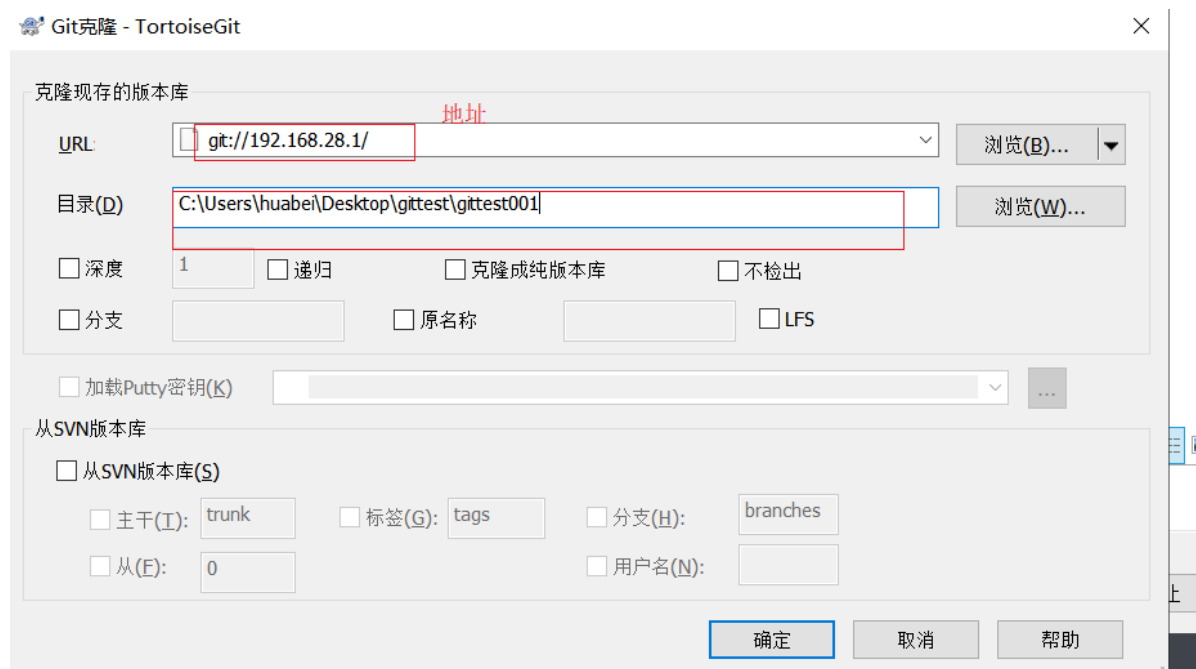
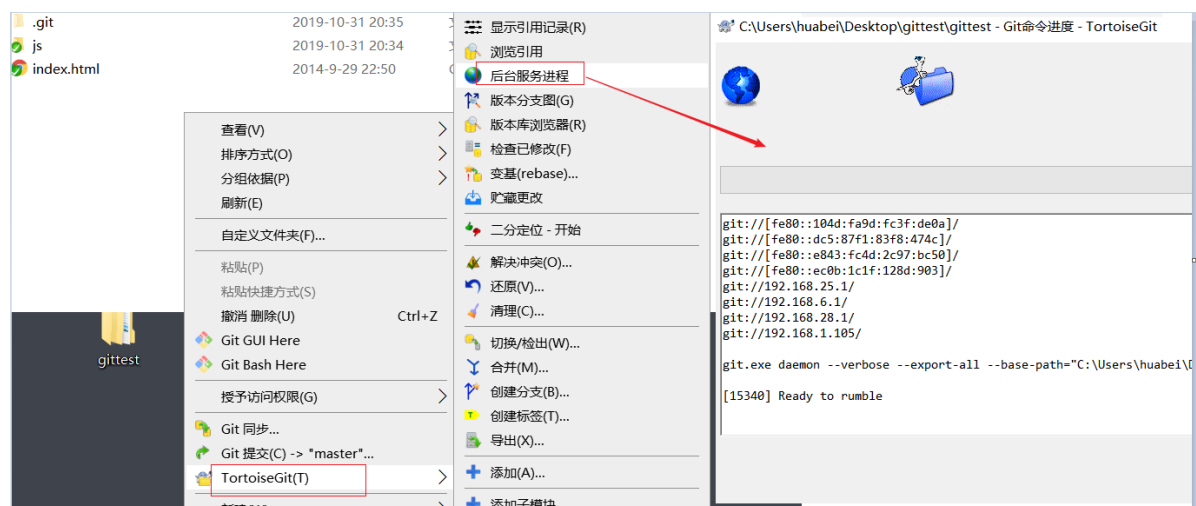
9.0 局域网仓库

实际上我们可以搭建一个单机的局域网服务器共享我们的代码

9.0.1 本地相对路径,多个文件夹之间共享代码



9.0.2开启局域网共享代码



局域网这种共享是没有安全控制的,都可以访问,如果想要搭建一个可以控制权限的服务器需要借助第三方软件

gitblit,可以自行搜索搭建

9.1 常用远程仓库托管服务

除了自己搭建服务器,其实我们可以使用一些免费的远程仓库,远程仓库有很多,常见的免费互联网远程仓库托管服务如下:

www.github.com
www.gitee.com
www.gitlab.com

github 是一个基于git实现在线代码托管的仓库,向互联网开放,企业版要收钱。

gitee 即码云,是 oschina 免费给企业用的,不用自己搭建环境。

gitlab 类似 **github**,一般用于在企业内搭建git私服,要自己搭环境。

GitHub(gitee)、GitLab 不同点:

- 1、**GitHub**如果使用私有仓库是需要付费的,(2019年开始私有仓库也是免费的但是只能3个人协同开发,想要更多需要收费),**GitLab**可以在上面搭建私人的免费仓库。
- 2、**GitLab**让开发团队对他们的代码仓库拥有更多的控制,相对于**GitHub**,它有不少的特色:
 - (1)允许免费设置仓库权限
 - (2)允许用户选择分享一个project的部分代码
 - (3)允许用户设置project的获取权限,进一步提升安全性
 - (4)可以设置获取到团队整体的改进进度
 - (5)通过innersourcing让不在权限范围内的人访问不到该资源

鉴于国内用户可能网络不好,这里我们使用gitee(码云) 来讲解我们的课程,其他可自行找资料学习非常类似

9.2 码云账号注册



码云 Gitee

云端软件开发协作平台

阴明 掘金创始人

许多年轻的开发者使用码云共享和协作,码云也是难得的为中国开发者的软件服务。期待未来码云继续发展,成为中国开发行业发展之基石!

注册

已有帐号? [点此登录](#)

传智播客

https://gitee.com/ itcastgit ?

soyo@mailseo.net

识别码: SJH

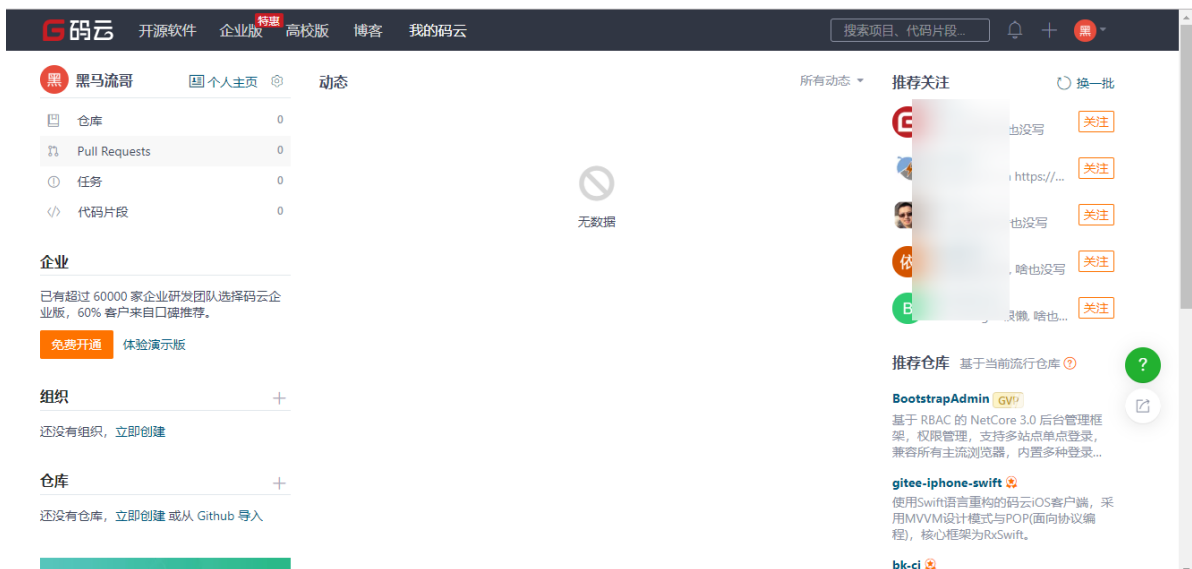
298 秒后可重发

密码不少于6位

☐ 我已阅读并同意 [使用条款](#) 以及 [非活跃帐号处理规范](#)

立即注册


填写邮箱发送验证码,然后可以注册账号,主页如下



9.3 创建远程仓库




新建仓库

仓库名称 

gittest

仓库名称

归属

 传智播客

路径

/ gittest

仓库地址: <https://gitee.com/itcastgit/gittest>

仓库介绍 非必填

用简短

非必填项

是否开源



私有



公开

私有仓库表示只能团队内部人员查看使用
共有仓库表示互联网所有人都能免费下载和使用

私有仓库的非仓库成员无法访问该仓库的代码和其他任何形式的资源

私有仓库最多支持 5 人协作 (如拥有多个私有仓库, 所有协作人数总计不得超过 5 人)

企业仓库, 更适合使用码云企业版, [了解更多 >>](#)

选择语言

请选择语言


添加 .gitignore

请选择 .gitignore 模板




使用Readme文件初始化这个仓库



使用Issue模板文件初始化这个仓库 



使用Pull Request模板文件初始化这个仓库 

这里无需勾选,勾选会生成一个初始化文件

选择分支模型 (仓库初始化后将根据所选分支模型创建分支)

单分支模型 (只创建 master 分支)

 导入已有仓库

创建

© Gitee.com

关于我们

Git 大全

代码片段

OpenAPI



官方技术交流QQ群: 830782254

使用条款

Git 命令学习

码云封面人物

帮助文档



git@oschina.cn  码云Gitee

意见建议

代码克隆检测

GVP项目

在线自助服务

企业版售前及售后使用咨询: 400-606-0200

合作伙伴

APP与插件下载

码云博客

更新日志

各个类型仓库之间的区别

	社区版		企业版
	海量优质开源项目、代码片段 与数百万开发者一起，发现、记录、成长		项目管理、代码托管、文档协作，一站式解决方案 帮助团队专注开发、高效协作
使用场景	公开仓库	个人私有仓库	协作开发
协作人数	不限	≤ 5 人	20 ~ 100 人+
仓库总容量	5G	5G	20 ~ 100G+
单仓库大小	500M	500M	1 ~ 3G
单文件大小	50M	50M	100 ~ 300M
附件总容量	3G	3G	10 ~ 50G +



快速设置—如果你知道该怎么操作，直接使用下面的地址

HTTPS SSH

此处就是我们的仓库地址

我们强烈建议所有的git仓库都有一个 README, LICENSE, .gitignore 文件

Git入门? 查看 帮助, Visual Studio / TortoiseGit / Eclipse / Xcode 下如何连接本站, 如何导入仓库

简易的命令行入门教程:

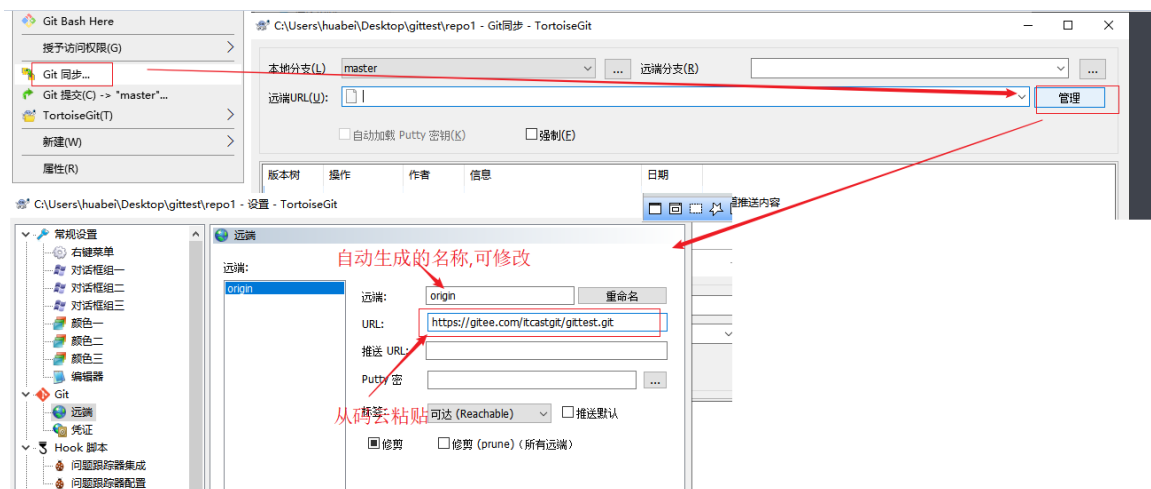
Git 全局设置:

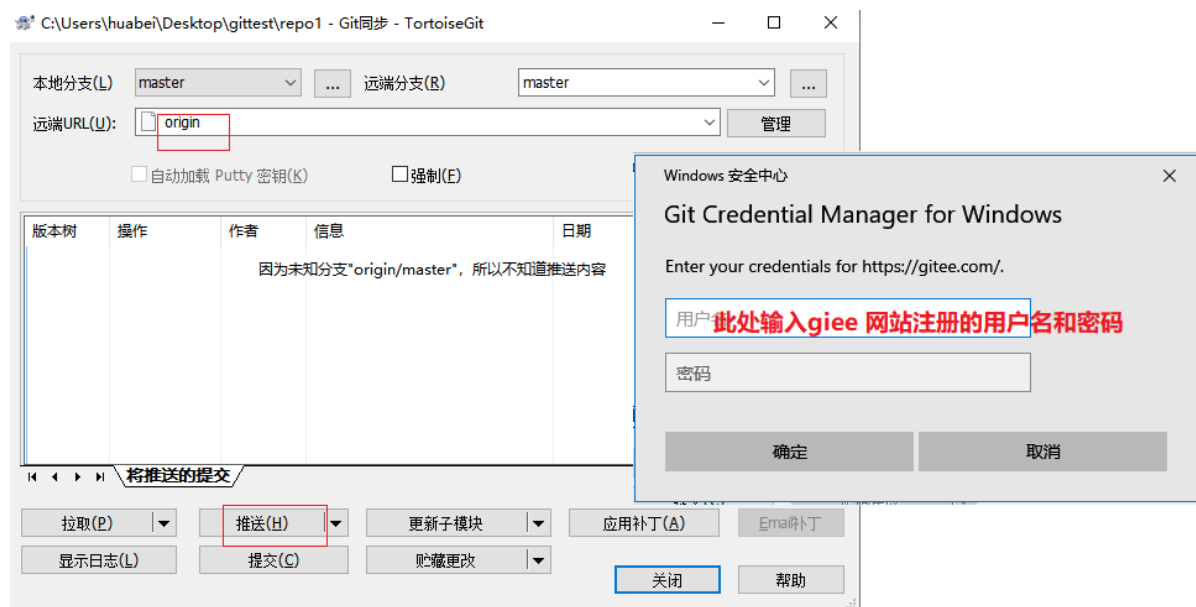
```
git config --global user.name "传智播客"
git config --global user.email "soyo@mailseo.net"
```

创建 git 仓库:

```
mkdir gittest
cd gittest
git init
```

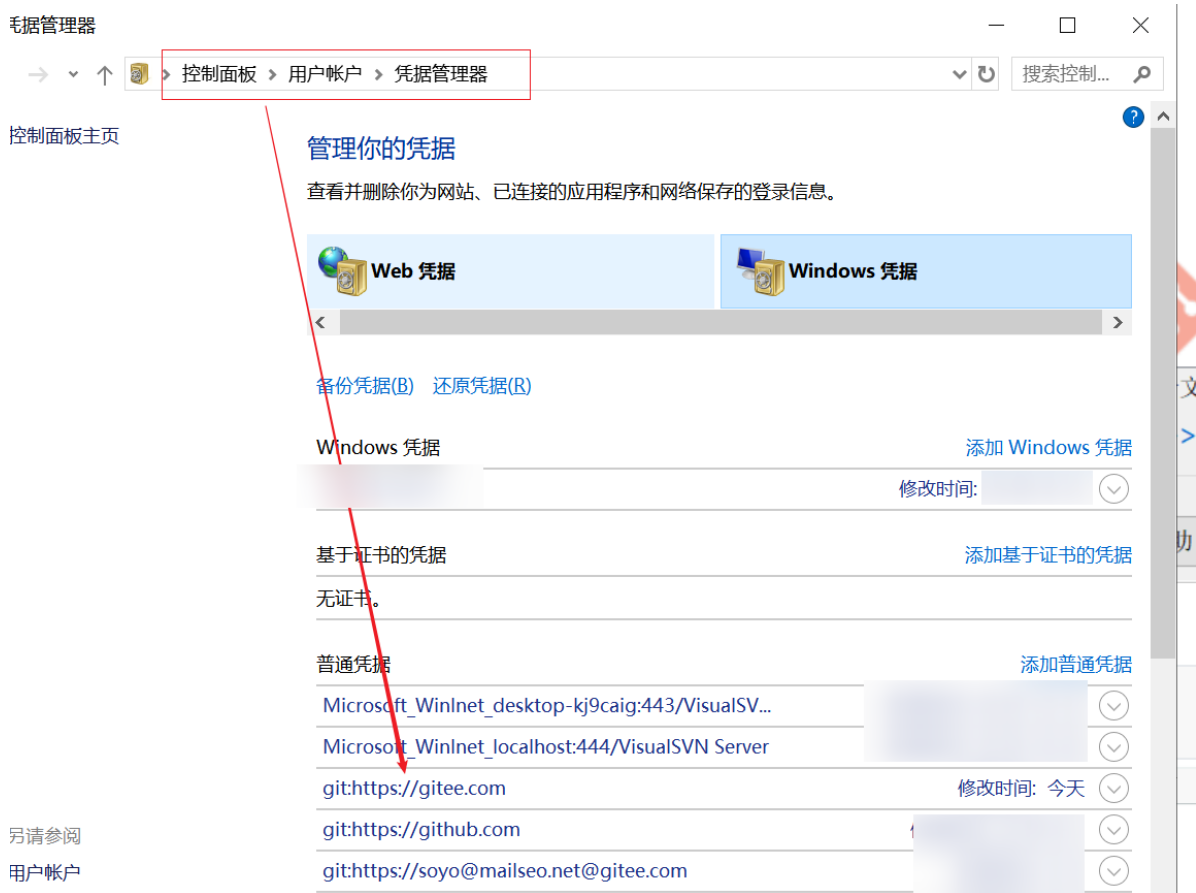
9.4 把本地代码推送到远端





此时我们刷新仓库发现代码已经存在了

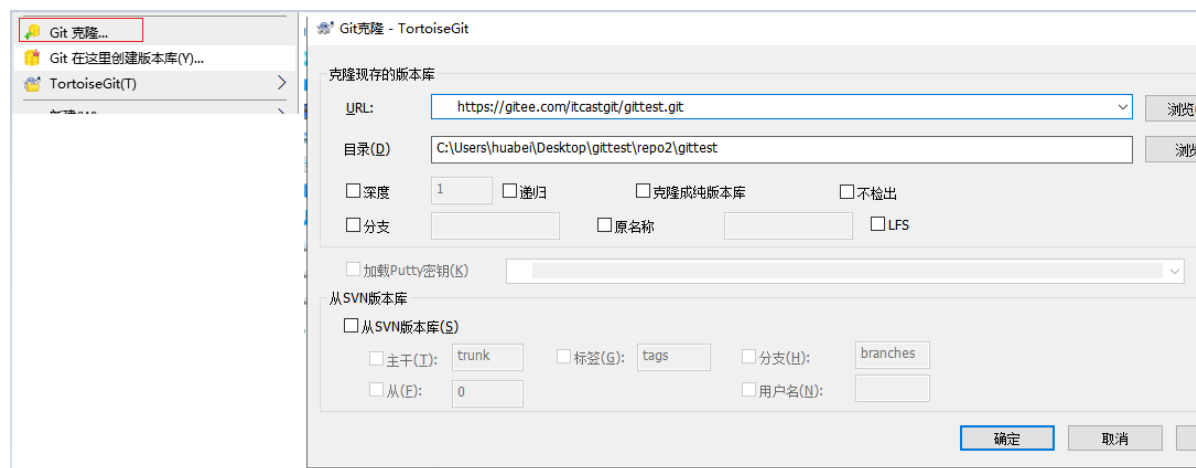
我们填写的用户信息,会被保存在本地,下次提交无需填写用户名和密码



9.5 从远程仓库克隆代码

我们同样可以从库下载代码,

新建一个文件夹 repo2 ,进入然后进行如下操作



此时我们发现我们的代码已经被下载下来了

9.6 代码的修改与提交,查看历史

- 1) 此时我们修改代码就不能仅仅是提交到本地了,提交完毕应该推送到远端服务器
- 2) 此时如果别人从远端仓库下载最新的代码其实是可以看到我们的代码修改记录的
`git -->`显示日志


```
huabei@DESKTOP-KJ9CAIG MINGW64 ~/ssh
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/huabei/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/huabei/.ssh/id_rsa.
Your public key has been saved in /c/Users/huabei/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:WehRrZ/JR6cEsK3gE5lkbG/b3KXScFqEqjlr2S6xckg huabei@DESKTOP-KJ9CAIG
The key's randomart image is:
```

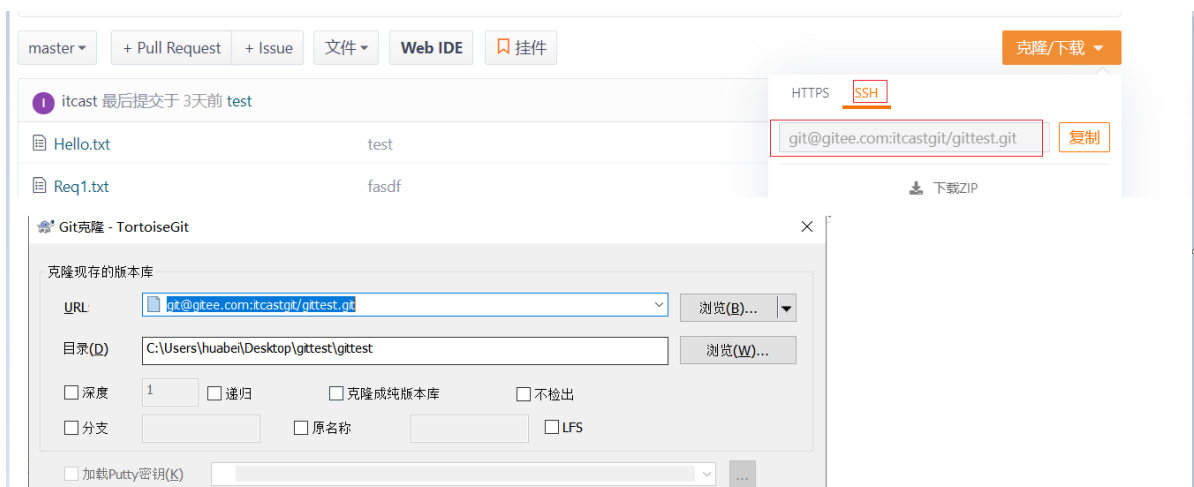
```
.ssh
├── id_rsa
└── id_rsa.pub
```

9.9 ssh 密钥配置



9.10 ssh 方式克隆/提交代码:

配置完成之后我们克隆我们之前的项目



修改后直接提交推送即可成功,,git 会自动去.ssh 目录找我们的私钥进行匹配

9.11. 远程仓库的其他操作

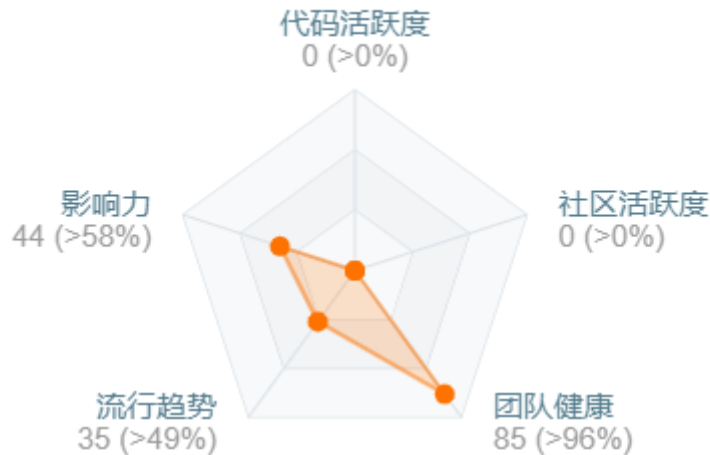
概念



当我们从 gitee 上查看别人的项目的时候我们可能会看到上图中的按钮

指数：

是gitee 网站根据当前项目的各项指标计算出来的一个值



代码活跃度：与代码提交频次相关

社区活跃度：与项目和用户的issue、pr互动相关

团队健康：与团队成员人数和稳定度相关

流行趋势：与项目近期受关注度相关

影响力：与项目的star、下载量等社交指标相关

Star：

点赞，注意这里的并不像朋友圈那样容易获得点赞，圈内人还是很克制的

watch：

如果你watch 了某个开源项目，那么这个项目后续所有的改动你将收到通知

Fork：

将别人的代码克隆到你自己的仓库

作用一：如果担心某个优秀的项目别人突然有一天不开源了，你可以fork到自己的仓库

作用二：修改别人的代码

以linux 为例，你其实不是linux 社区的开发人员，但是你 又想为linux 开发做贡献(维护代码)

你并没有权限，怎们办？

你可以先把linux 开源的代码 fork 到你自己的仓库，此时你就可以操作自己的仓库进行修改代码

了

如何让别人合并你修改好的代码呢？

我们注意项目的上方有一个 " Pull Request" 这个按钮的意思是 "请求求别人合并你修改的代码"

当我们发起一个 Pull Request 时，项目的拥有者将收到 Pull Request请求，然后将根据你的提交代码的质量决定是否合并

项目操作

1)我们可以删除修改我们自己仓库的基本信息

2. 我们可以邀请其他人成为项目的开发人员或者管理人员

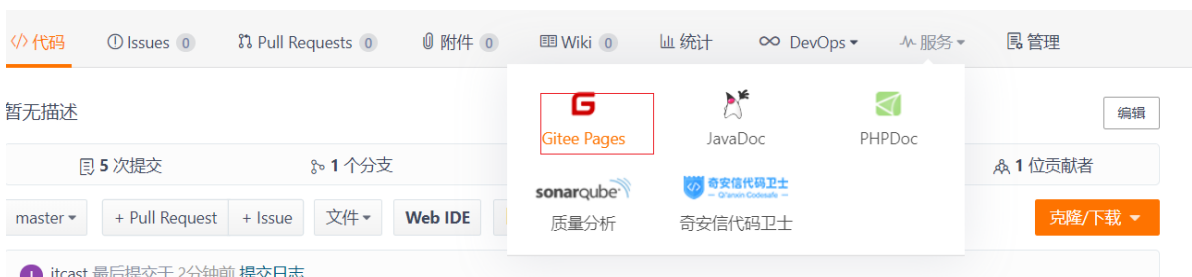


我们可以删除修改我们自己仓库的基本信息



9.12 利用 gitee 搭建个人主页

- 1) 将静态资源上传至仓库
 - 2) 选择服务 **pages** 即可部署
- 注意 1) 必须有个 **index.html** 文件
- 注意 2) 只能搭建静态网站, 动态网站请租赁服务器搭建提供服务
- 注意 3) **gitee** 要求必须绑定手机号



点击开启后gitee 会自动生成一个域名

[http://t\[redacted\].gitee.io/gittest](http://t[redacted].gitee.io/gittest)

直接访问即可

此时我们已经在git 上部署了一个静态的网站

🔍 不安全 | [t\[redacted\].gitee.io/gittest/](http://t[redacted].gitee.io/gittest/)



10.命令行-- git基本操作

10.1 介绍

上述我们的操作 使用的是客户端TortoiseGit 操作的git ,实际上底层依旧是使用的命令行帮我们执行,在早期 git 并没有窗口化工具,开发人员只能使用命令行模式

实际上,如果你掌握并熟练使用了命令行模式操作git 的话,你会发现某些操作命令行比窗口化操作要简单
所有你在工作中会发现高深的技术人员可能会喜欢命令行模式提交git

##10.2 环境配置

当安装Git后首先要做的事情是设置用户名称和email地址。这是非常重要的，因为每次Git提交都会使用该用户信息

```
#设置用户信息
git config --global user.name "itcast"
git config --global user.email "itcast@itcast.cn"
#查看配置信息
git config --list
git config user.name
#通过上面的命令设置的信息会保存在~/.gitconfig文件中
```

##10.3 初始化本地仓库 init

```
# 初始化仓库带工作区
git init
# 初始化仓库不带工作区
git init --bare
```

##10.4 克隆 clone


```
# 从远程仓库克隆
git clone 远程Git仓库地址
例如: git clone https://gitee.com/itcast/gittest.git
```

##10.5 查看状态 status

```
# 查看状态
git status
#查看状态 使输出信息更加简洁
git status -s
```

##10.6 add

```
# 将未跟踪的文件加入暂存区
git add <文件名>
# 将暂存区的文件取消暂存 (取消 add )
git reset <文件名>
```

##10.7 commit

```
# git commit 将暂存区的文件修改提交到本地仓库
git commit -m "日志信息" <文件名>
```

##10.8 删除 rm

```
# 从本地工作区 删除文件
git rm <文件名>
# 如果本工作区库误删, 想要回退
git checkout head <文件名>
```

11. 命令行--git 远程仓库操作

11.1 查看远程

```
# 查看远程 列出指定的每一个远程服务器的简写
git remote
# 查看远程 , 列出 简称和地址
git remote -v
# 查看远程仓库详细地址
git remote show <仓库简称>
```

11.2 添加/移除远测仓库

```
# 添加远程仓库
git remote add <shortname> <url>
# 移除远程仓库和本地仓库的关系(只是从本地移除远程仓库的关联关系, 并不会真正影响到远程仓库)
git remote rm <shortname>
```

11.3 从远程仓库获取代码

```
# 从远程仓库克隆
git clone <url>
# 从远程仓库拉取（拉取到.git目录，不会合并到工作区，工作区发生变化）
git fetch <shortname> <分支名称>
# 手动合并 把某个版本的某个分支合并到当前工作区
git merge <shortname>/<分支名称>
# 从远程仓库拉取（拉取到.git目录，合并到工作区，工作区不发生变化）= fetch+merge
git pull <shortname> <分支名称>
git pull <shortname> <分支名称> --allow-unrelated-histories # 强制拉取合并
```

注意：如果当前本地仓库不是从远程仓库克隆，而是本地创建的仓库，并且仓库中存在文件，此时再从远程仓库拉取文件的时候会报错（fatal: refusing to merge unrelated histories），解决此问题可以在git pull命令后加入参数--allow-unrelated-histories (如上 命令)

```
# 将本地仓库推送至远程仓库的某个分支
git push [remote-name] [branch-name]
```

12. 命令行-- 分支

```
# 默认 分支名称为 master
# 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
# 创建分支
git branch <分支名>
# 切换分支
git checkout <分支名>
# 删除分支(如果分支已经修改过,则不允许删除)
git branch -d <分支名>
# 强制删除分支
git branch -D <分支名>
```

```
# 提交分支至远程仓库
git push <仓库简称> <分支名称>
# 合并分支 将其他分支合并至当前工作区
git merge <分支名称>
# 删除远程仓库分支
git push origin -d branchName
```

13. 命令行 --tag

```
# 列出所有tag
git tag
# 查看tag详细信息
git show [tagName]
# 新建一个tag
git tag [tagName]
```

```
# 提交指定tag
$ git push [仓库简称] [tagName]
# 新建一个分支，指向某个tag
$ git checkout -b [branch] [tag]
# 删除本地tag
$ git tag -d [tag]
# 删除远程tag (注意 空格)
$ git push origin :refs/tags/[tag]
```

14. 案例

企业中我们是如何开发的

- 1) 入职第一天,管理人员分配/git账号密码
- 2) 开发人员下载代码即文档/ 根据文档将环境搭建成功
- 3) 团队一般会给你讲讲项目相关的支持
-
- 4) 你接到第一个需求(或者某个功能,一般要经过沟通,分析,设计...等过程)
- 5) 创建**feature**分支(一般一个需求对应一个**feature**,命名格式上标注该需求的id)
- 6) 开发需求,本地测试,提交代码到当前需求对应的**feature**分支,
一般来讲为了避免将测试代码提交,需要提交前,检查如下步骤
 - 6.1) 是否多提交了某个文件,比如测试文件
 - 6.2) 是否漏提交文件
 - 6.3) 打开每一个应该提交的文件,判断是否多提交了一行代码,是否少提交了一行代码,是否删除了本该存在的代码
- 检查完毕提交代码
- 7) 合并分支至**test**分支-- 测试人员会在**test**分支中测试
- 8) 测试人员测试bug ,开发者在**feature**分支上继续修改,提交
- 9) 测试人员测试通过 ,**test**分支会被测试人员合并到**develop**开发分支,再次测试
- 10) **develop**分支最终会被合并到**master**主分支

