**一、集中式和分布式区别：**

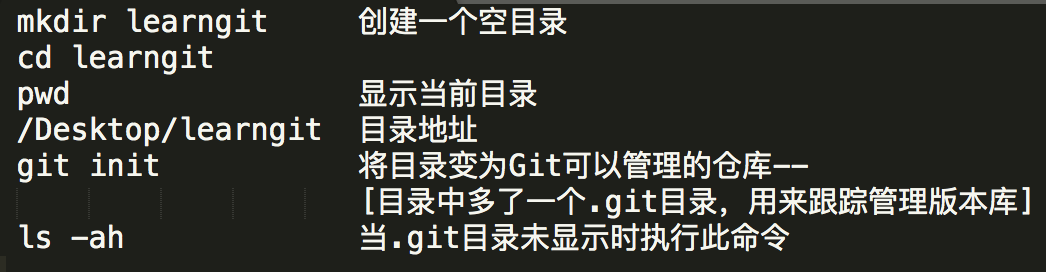
1. 集中式版本控制系统，版本库都放在中央服务器上，工作时用的都是自己的电脑，需要先从中央服务器取到最新版本，集中式版本控制必须联网才能工作；分布式版本控制系统没有“中央服务器”，每个人的电脑都是一个完整的版本库，所以不需要联网。

2. 和集中式版本控制系统相比，分布式版本控制系统的安全性高很多，因为每个人的电脑里都有完整的版本库；而集中式版本控制只要中央服务器出现问题，所有人都无法干活

**二、创建版本库：**

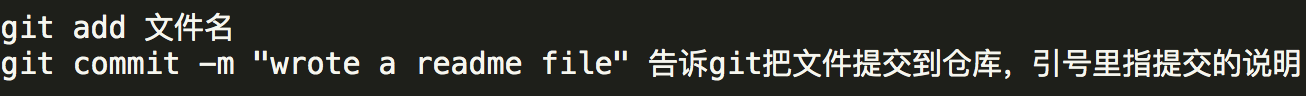
简介：版本库又名仓库，可以简单理解为目录，这个目录里的每个文件都可以被git管理起来，每个文件的修改、删除、git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”

命令代码：



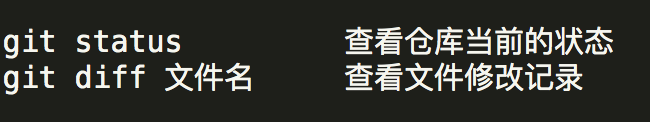
注意：版本控制系统没法跟踪Word文件的改动

命令代码：



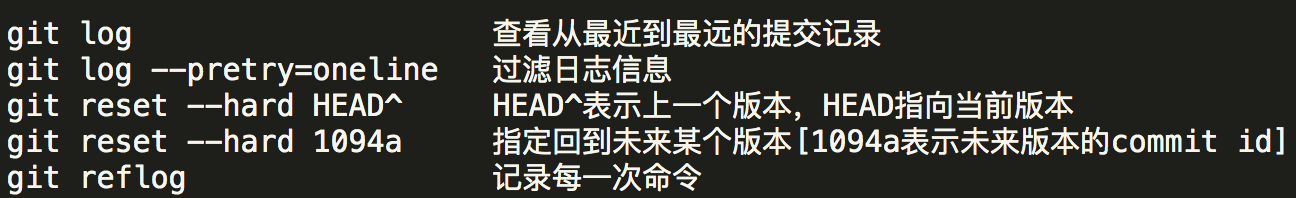
注意：commit可以一次提交很多文件，所以可以多次add不同的文件

命令代码：



**命令代码--版本回退**

简介：一旦你把文件改乱了，还可以从最近的一个commit恢复，然后继续工作



**三、工作区和暂存区**

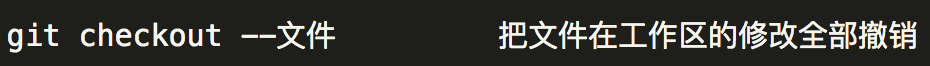
Git的版本库里存了很多东西，最重要的称为stage/index的暂存区，Git会为我们自动创建一个分支master，以及master的每一个指针HEAD

注意：Git跟踪并管理的是修改，而非文件

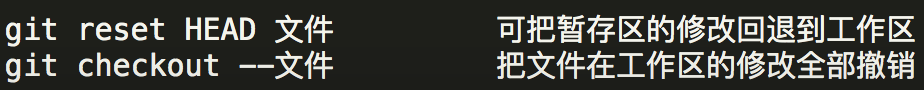
**四、回退命令**

**命令代码--取消修改**

**1. 当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时：**

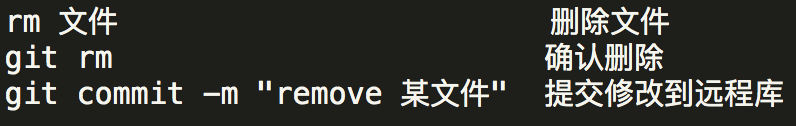


**2. 当你不但改乱了工作区某个文件的内容，还添加到了暂存区，想丢弃修改，分两步：**



**命令代码--删除**

1. 确认删除某个文件



2. 删错了

image.png

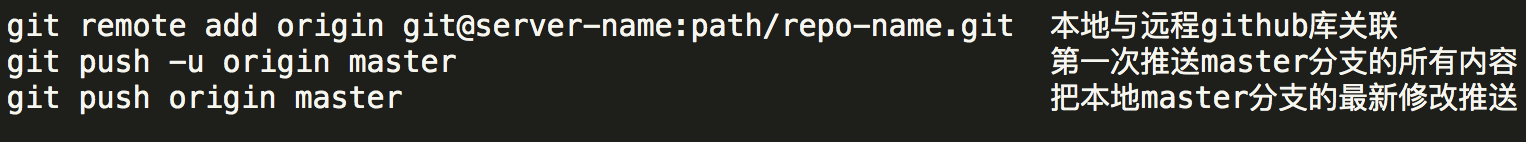
注意：如果一个文件已经被提交到版本库，你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失最近一次提交后你修改的内容

**五、远程仓库：**

简介：找一台电脑充当服务器的角色，每天24小时开机，其他每个人都从这个“服务器”仓库克隆一份到自己的电脑上，并且各自把各自的提交推送到服务器仓库里，也从服务器拉取别人的提交

Github：Git仓库托管服务，GitHub需要SSH key，因为它需要识别出你推送的提交确实是你推送的，git支持SSH协议

命令代码：



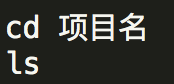
**新项目开发步骤**

1. 创建远程库

2. 克隆代码

image.png

3. 查看项目目录结构



注意：Git支持多种协议，默认的git://使用ssh，也可以使用https等其他协议，https除速度慢以外，最大的麻烦是每次推送都必须输入口令，但是某些只开放http端口的公司内部就无法使用ssh协议而只能用https

**六、分支管理**

简介：分支在实际中开发中的应用是，假如你新开发了一个功能，但是需要较长时间才能完成，第一周写了50%代码，如果立即提交，由于代码还没写完，不完整的代码库会导致被人不能干活，如果等代码全部写完再一次提交，又存在丢失每天进度的风险。当有分支后，你可以创建一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，知道开发完毕后，再一次性合并到原来的分支上，这样既安全，又不影响别人工作。

master：主分支

HEAD：严格来说不是指向提交，而是指向master，master才指提交，HEAD指向当前分支

描述：当我们创建新的分支，例如dev时，Git新建一个指针叫dev，执行master相同的提交，再把HEAD指向dev，就表示当前分支在dev上。假如dev上的工作完成了，就可以把dev合并到master上，合并最简单的方法是直接把master指向dev的当前提交。

注意：Git合并分支很快，改改指针，工作区内容不变

命令代码--分支管理：

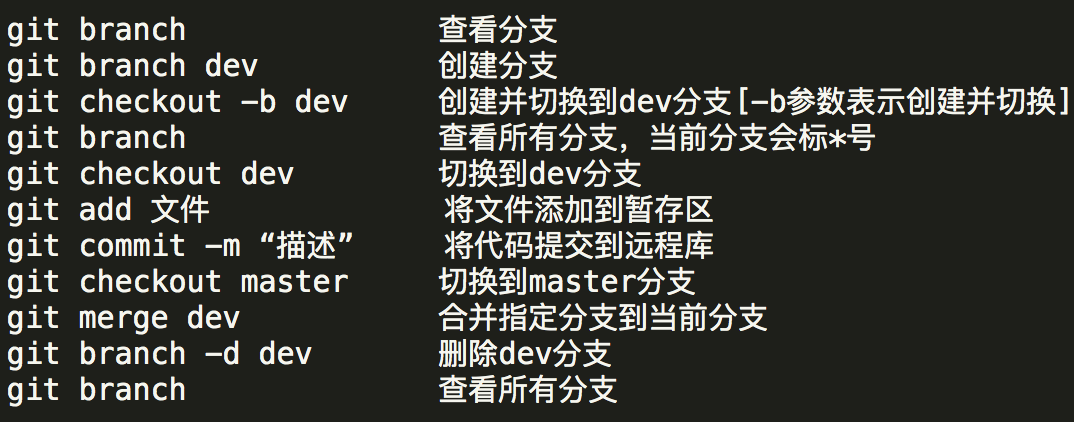


image.png

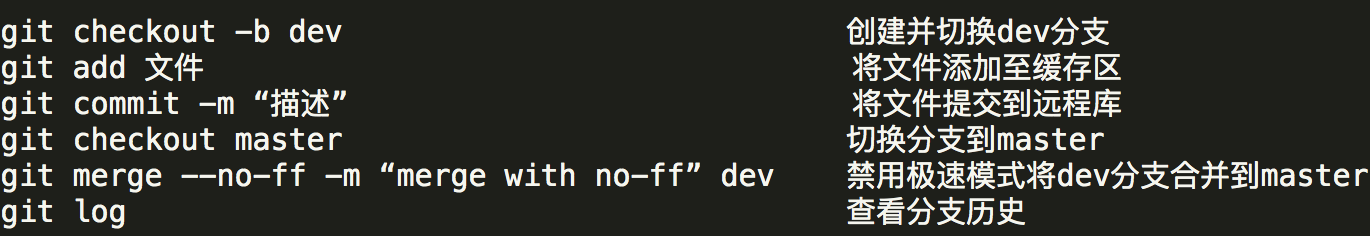
**解决冲突：**

当Git无法自动合并分支时，就必须先解决冲突，解决冲突后，再提交，合并完成。

解决冲突：把Git合并失败的文件手动编辑为希望的内容，再提交

image.png

**分支管理策略**：通常合并分支时，git会用Fast forward模式，但这种模式下，删除分支后会丢掉分支信息，所以禁用Fast forward



**实际开发中应按照如下操作：**

1. master分支是最稳定的，仅用来发布新版本，平时不能再上面干活

2. 干活都在dev上，dev分支是不稳定的，当发布时，将dev分支合并到master上

3. 每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并

**Bug分支：**

简介：每个bug可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除

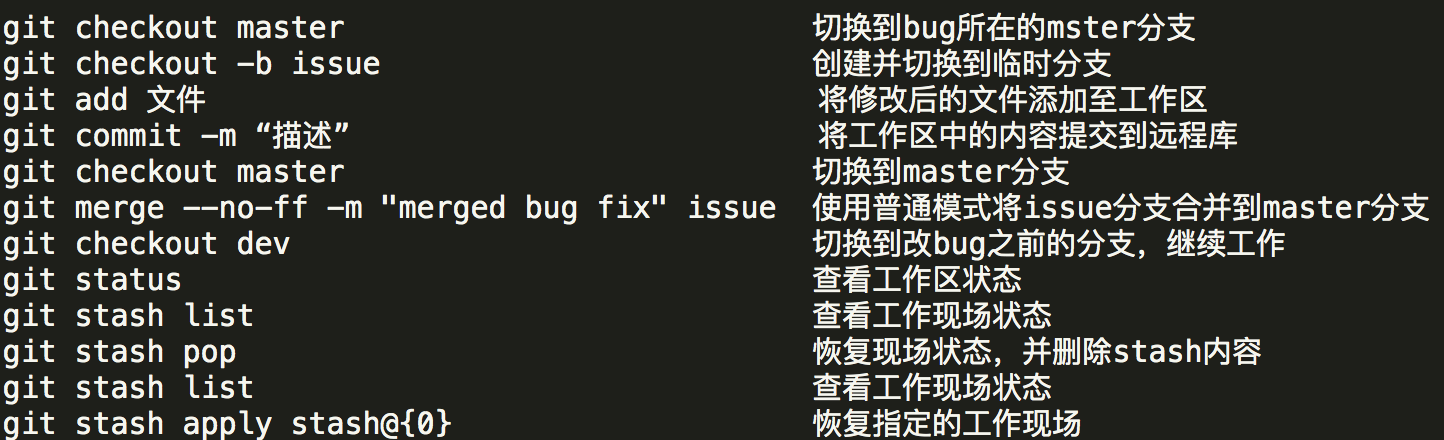
1. 当临时接到一个bug时

image.png

2. 将进行一半的工作储存起来

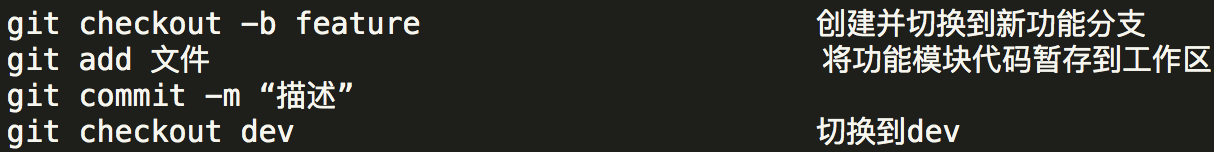
image.png

3. 确定在哪个分支上修复bug，假定需要在master分支上修复，就从master创建临时分支



**Feature分支：**

简介：添加一个新功能时，不希望把主分支搞乱，每添加一个新功能，最好新建一个feature分支，完成开发后合并并删除分支。



当准备合并时接到通知，新功能取消，因为机密文件必须删除

image.png

**多人协作：**

简介：当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。

**1. 查看远程分支**

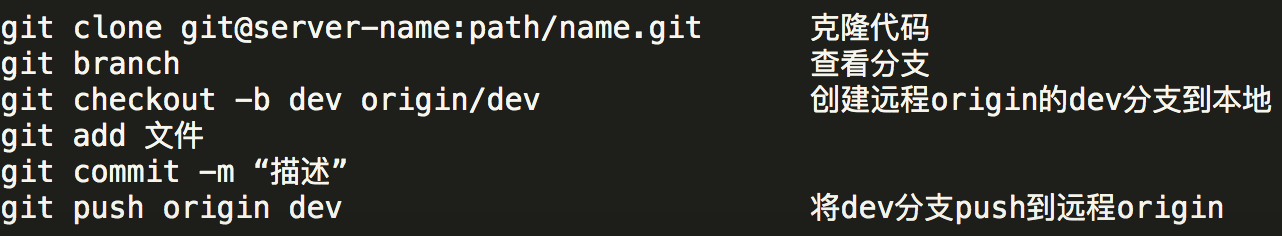


**2. 推送分支**

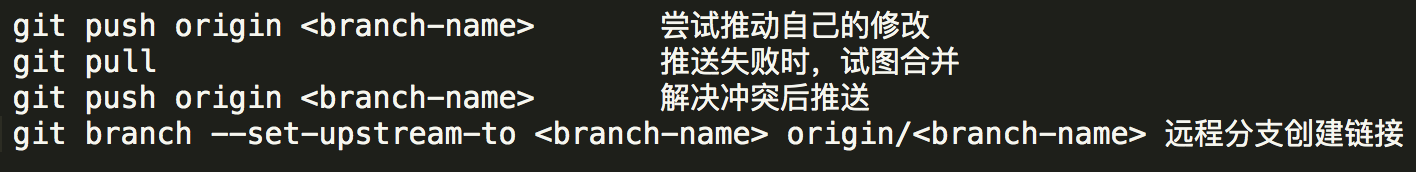
image.png

**3. 抓取分支**

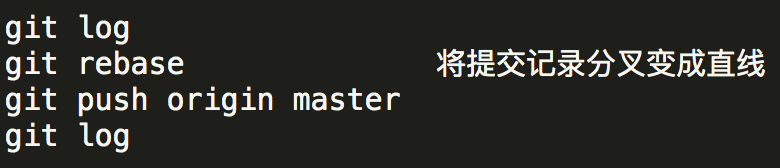
1⃣ 当你的同事正在推动dev分支的修改，你做同样的操作是就会push失败



2⃣ 解决冲突



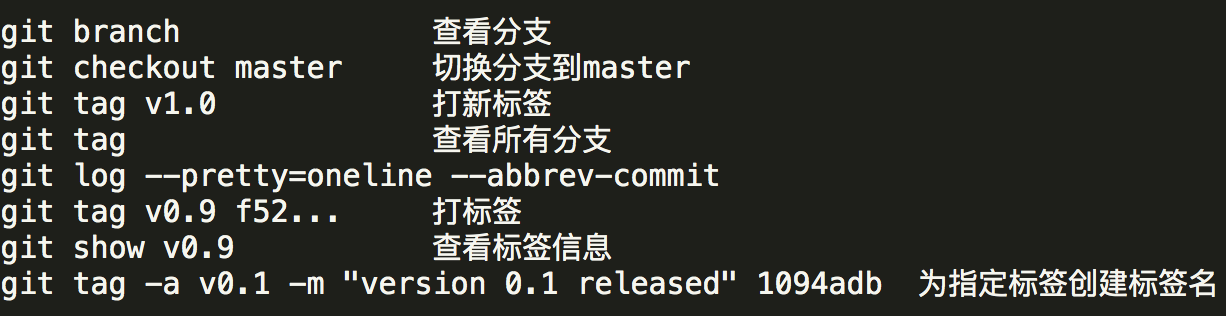
3⃣ rebase：把本地未push的分叉提交历史整理成直线，rebase的目的是使我们在查看历史提交的变化时更容易



**七、标签**

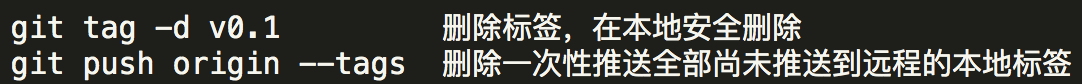
tag：一个容易记住的有意义的名字，他跟某个commit绑在一起

**1. 创建标签**

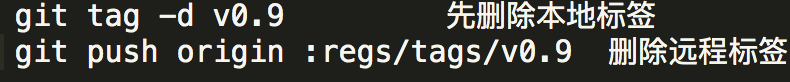


注意：标签总是和某个commit挂钩，如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签

**2. 操作标签**



如果标签已经推动到远程则删除应按如下操作：



**八、参与开源项目**

**1. github：**

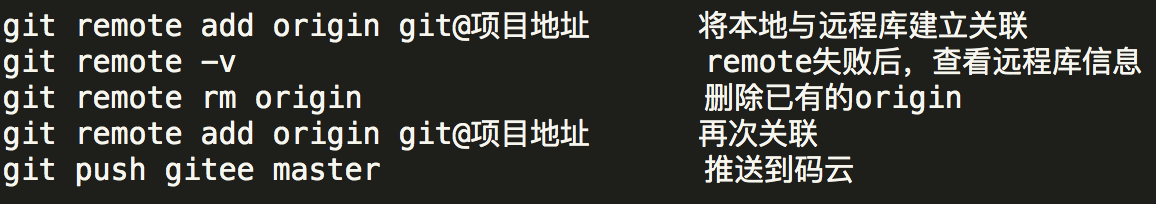
Fork在自己账号下克隆一个仓库

git clone git@自己的仓库地址

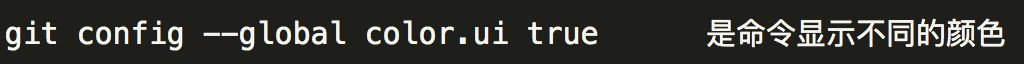
新增或修改项目后，推送到自己仓库

pull request给官方仓库来贡献代码

**2. 码云**



**九、自定义Git**



**忽略特殊文件：**

1. 忽略操作系统自动生成的文件，比如缩略图等

2. 忽略编译生成的中间文件、可执行文件等，即如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库

3. 忽略自己带有敏感信息的配置文件，比如存放口令的配置文件

image.png

注意：.gitignore文件本身要放到版本库里，并且可以对.gitignore做版本管理

**注意：**

**1. 请勿使用图形化git工具或者网页端的gitlab的一些git操作，由于图形化操作很可能会造成一些错误操作，所以，请使用git命令行工具，包括代码冲突修改等操作**

**2. 一个需求做下来后期有相关改动最好都在一个分支上继续进行，如果在两个分支上进行，可能会出现代码冗余的情况**

每次提交时，将commit合并成一个上传，使用git rebase合并多次commit：

<https://blog.csdn.net/yangcs2009/article/details/47166361>

**练习网站：**

Git练手网站：<https://learngitbranching.js.org/>

注意：

1. 即使创建再多分的支也不会造成存储或内存上的开销，并且按逻辑分解工作到不同的分支要比维护那些特别臃肿的分支简单的多

2. HEAD总是指向当前分支上最近一次提交记录，大多数修改提交数的Git命令都是从改变HEAD的指向开始的

**补充：**

git stash:将本地代码stash到仓库中

git stash show: 显示stash合并到本地代码后，修改概述

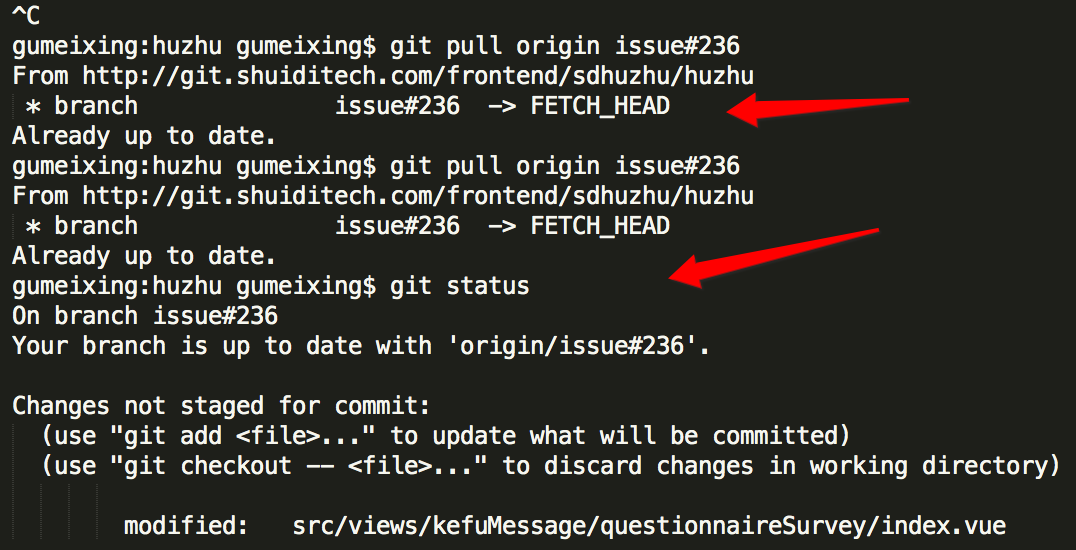
git stash pop: 在合并代码的同时，把仓库中对应的内容弹出，如果只想查看，而不想弹出内容，可以使用git stash apply命令进行操作。

**9月6号之合代码之旅！**

由于我重新在gitLab上clone了代码，到本地后直接npm run dev运行不起来，后来npm install依赖，发现还是运行不起来，结果关了vpn后就可以了，我以为是vpn的原因，原来是我重置用户名后gitLab上的用户名和邮箱（即可能与gitLab失去了关联）。

git diff --- 查看发生冲突的位置

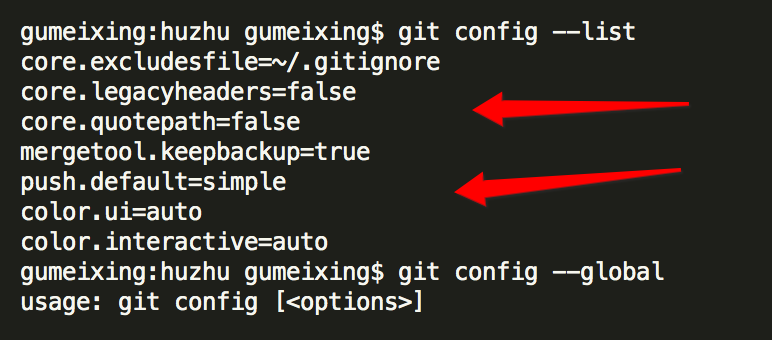
git pull origin issue#236 ----- 拉取远程代码发生冲突



一通git add ./git commit -m ''之后无效

git cofig --list ---- 检查你的设置

git config --global --- 检查全局配置



发现没有userName和eamil配置

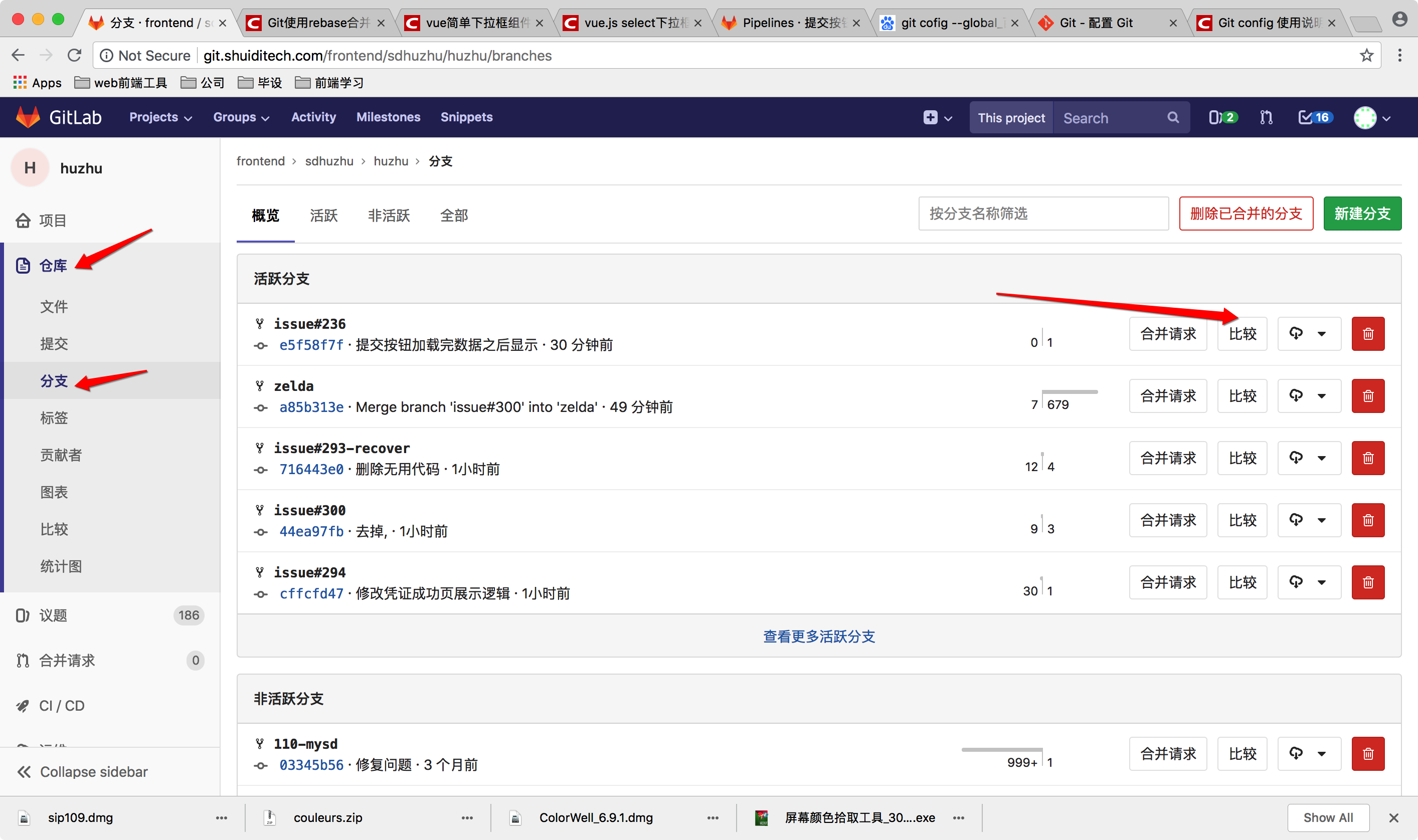
接下来配置用户名和邮箱：

git config --global user.name 'gumeixing'

git config --global user.email 'gumeixing@shuidihuzhu.com'

之后再git add . git commit -m '' git push origin issue#236

成功



点击仓库--分支--比较

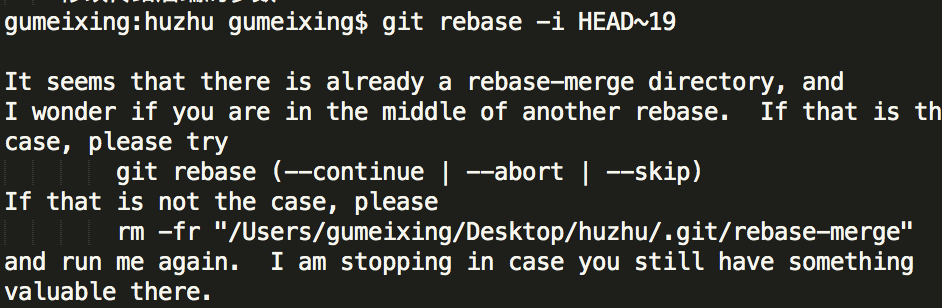
发现我居然commit了20个，哇这样一发布居然都是我的commit，太乱了

接下来就是rebase登场的时候了：

git log --- 查看所有的改动记录

将这些commit合并

git rebase -i HEAD~20



有错

git rebase --abort ---是无风险的操作，会回到rebase操作之前的状态（2个分支的commits毫发无损）

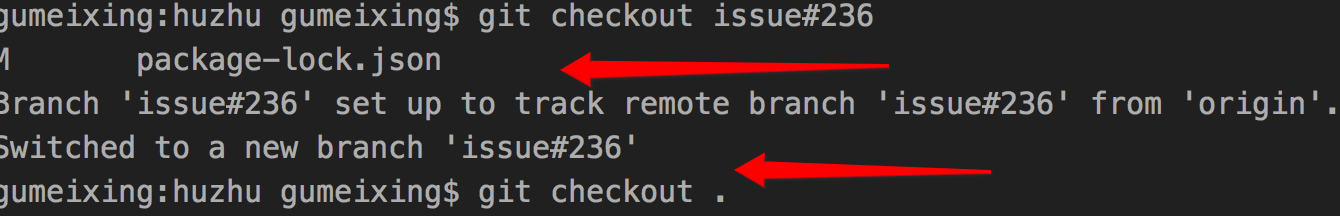
git rebase --skip --- 是高风险的操作，引起冲突的commit会被丢弃（这部分代码修改会丢失）

多次rebase之后无果直接重新clone了一份互助代码

npm install ----- 安装依赖

发现修改了package-lock.json文件

现在直接切换到我的分支会有问题



然后很惊喜的收获是：

直接git checkout . ---- 忽略这部分改动

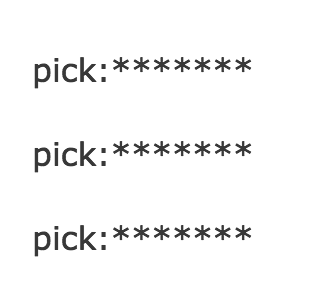
git checkout issue#236 --- 切换到我的分支

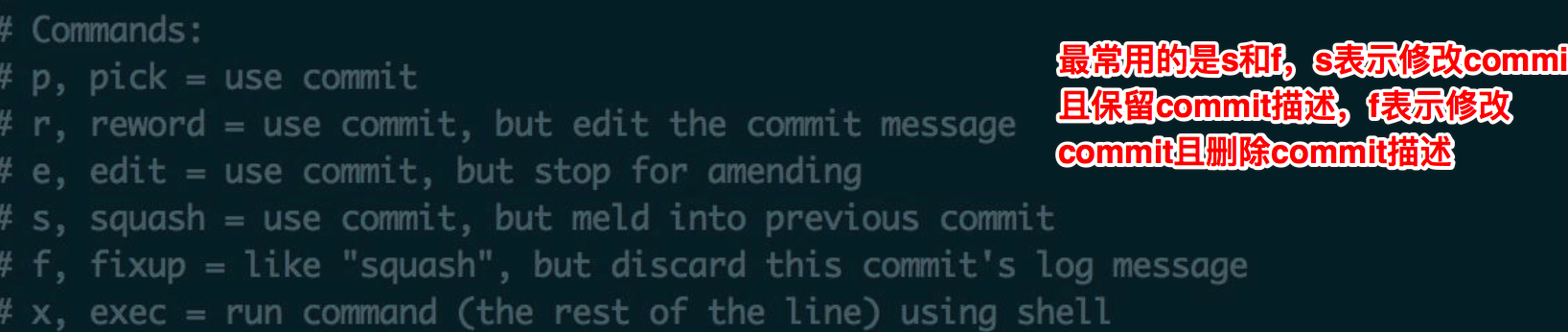
git pull origin issue#236 --- 拉去我远程分支的代码

终于可以合并commit了

git rebase -i HEAD~20 ----修改历史提交信息

（其中-I的参数是不需要合并的commit的hash值，也就是我们要rebase到的commit对象）

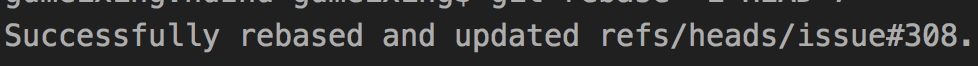




点“i”可以编辑

如果要改哪个，就把那行的pick改成f或者s，（f是指删除commit记录代码保留，而s表示保留commit且代码保留），然后点击**“esc”，:wq**保存退出。

*相关参考链接：*[*https://blog.csdn.net/crazyzhang1990/article/details/42780285*](https://blog.csdn.net/crazyzhang1990/article/details/42780285)



出现上图中的问题表示修改记录成功

最后提交修改

git push origin issue#236这时推时推不上去的

强制推：

git push origin issue#236 -f ---- 强制推代码,并把远程的覆盖

补充：

退出的vim语言是：:q!

保存的vim语言是：wq

git pull origin master --rebase --- 好像是查看可以合并的分支吧

强哥说在本地切换分支不建议直接使用 git checkout 分支名

建议使用fetch，意思是获取远程的issue#236分支后在本地命名为issue#236

git fetch origin issue#236:issue#236

哇 跟大神学习就是不一样~~~吃的好饱，嘻嘻

窍门：忘建分支，直接在master上改了很多东西

解决方式：

直接在master上新建一个分支然后切过去即可

git checkout -b issue#256

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

每次开发新功能时，需先在代码仓库建立自己的issue，然后从master拉取最新代码，之后新建一个branch。开发完成后先进行git diff操作查看变更代码。

##################################

在合并master分支前，将commit信息合并为一个。

git log -------- 查看提交历史

git rebase -i HEAD~数字 ---------提交记录次数

保存成功后git会压缩提交历史，如果有冲突，修改的时候需要注意，保留最新的实例

git add .

git rebase --continue

**git切换分支保存修改代码的方法：**

**原文链接：**[**https://blog.csdn.net/anhenzhufeng/article/details/78052418**](https://blog.csdn.net/anhenzhufeng/article/details/78052418)

方法一：

及时commit代码--在new分支上把已经开发完后才能的部分代码commit掉，不push，然后奇幻岛另一个分支上修改代码，做完了commit，所有分支互不影响

方法二：

命令行输入git stash或者git stash save "修改的信息"，直接git stash的话git stash的栈会直接给你一个hash值作为版本的说明，如果用git stash save “注释”,git stash的栈会把你填写的“修改信息”作为版本说明。

当你回到old分支修改代码完成，回到当前分支时，输入:

git stash pop 或 git stash list 或 git stash apply stash@{0} 就可以回到保存的版本了。

（git stash pop的作用：将git stash栈中最后一个版本取出来，git stash apply stash@{0}的作用是可以指定栈中的一个版本，通过git stash list可以看到所有的版本信息）

然后选择一个需要的版本执行，git stash apply stash@{0}

??：

git stash：存储当前工作目录

git stash list：查看之前存储的所有版本列表

git stash pop {stash\_id}/git stash pop stash@{0}

git stash drop [stash\_id] -----如果不指定id，则默认删除最新的存储进度

git stash clear：清除所有的存储进度

??：

如果git add .之后发现漏掉了文件，撤回存入暂存区的文件：

git commit --amend

git rebase报错问题：

git rebase -i HEAD~5 后保存报错，意思是不能删除第一个commit的信息

image.png

运行git rebase --abort

然后 git log查看提交信息

修改希望的commit信息

然后保存

git push origin issue#323 -f

拓展：

git rebase --abort 回到rebase操作之前的状态，之前的提交不会丢弃

git rebase --skip 会将引起冲突的commits丢弃掉

git rebase --continue 用于修复冲突，提示开发者，一步一步地有没有解决冲突

记10.30日一个惊险动作:

gitLab上发现有别人的commit信息之后，按照之前老师教的方法：

git checkout zelda

git pull orign zelda

git pull origin master

git push origin zelda

结果一刷新页面结果发现，有报错信息：惊恐万分，于是请教老师

建议重新发一次zelda：

就好了

git rebase和git merge的区别：

**参考资料：**[**http://www.cnblogs.com/kidsitcn/p/5339382.html**](http://www.cnblogs.com/kidsitcn/p/5339382.html)

共享分支：那些存在于远端并且允许成员进行pull操作的分支

**代码版本回退：**

[**https://www.jianshu.com/p/4ac5f52d9210**](https://www.jianshu.com/p/4ac5f52d9210)

**git log**

**// 获取需要获得的git信息**

**git reset --hard 需要回退的版本commit信息**

**// 注意: 版本回退只能退到比当前版本低的代码版本**

**git push -f origin issue#版本**

[**https://www.cnblogs.com/wancy86/p/5848024.html**](https://www.cnblogs.com/wancy86/p/5848024.html) **回退到指定版本**

**记一次合并代码带来的惨痛教训：**

1. git pull origin master -----本地分支可以直接pull master，出来的提示直接:wq

2. 解完冲突一定要确认无误再push

**git reset --hard origin/master**