1. git的大致理解：Git是分布式，安全性要高很多，不需要联网就能工作，还有及强大分支管理。
2. 创建版本库：$ cd 某文件

$ git init

把文件的添加到仓库：$ git add 某文件

$ git commit -m”要写的备注”

1. 修改后查看工作目录的状态：$ git status

让我们得知目标文件有没有被修改，但看不了具体内容。

看修改的具体内容：$ git diff

四、版本回退

看更改历史记录：$ git log

简化前者的输出信息可用：$ git log –pretty=oneline

把当前版本回到上一个已提交状态：$ git reset --hard HEAD^

“--soft”是回退到上个版本未提交状态

“--mixed”是回到上个版本添加但未提交状态

用于显示文件内容：$ git cat

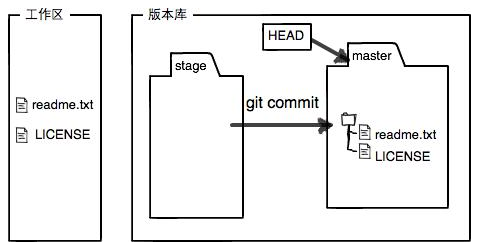
若要回到特定版本：$ git reset –hard commit id

Commit id 是SHA1计算出来的非常大的数字，为后期多人在同一个版本库中做铺垫，防止版本号冲突。

查看每一次命令：$ git reflog

1. 工作区和暂存区

即git add 是将修改放到暂存区（stage），git commit是一次性把暂存区的所有修改提交到分支。



1. 管理修改

Git跟踪并管理的是修改，而非文件。

Git commit只会把stage的修改给提交了，并不会把工作区的修改提交。

1. 撤销修改

丢弃工作区的修改：$ git checkout -- 文件名

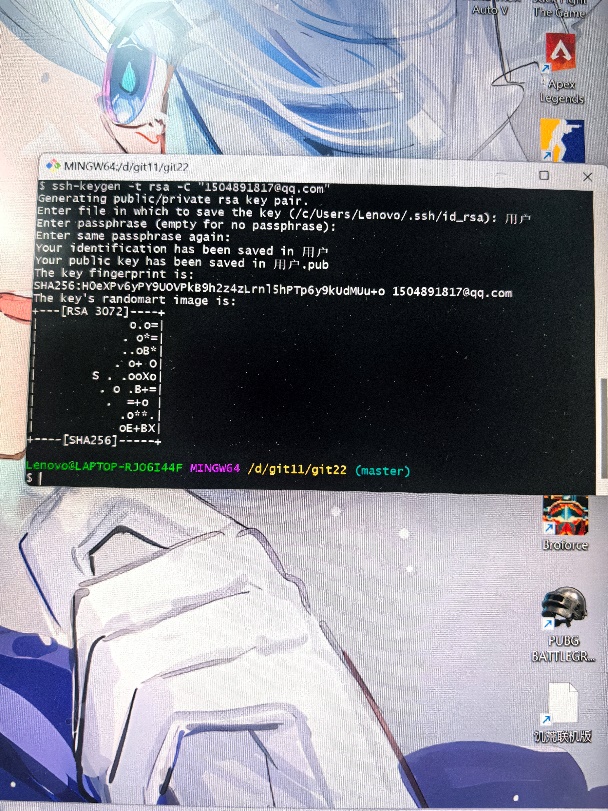
撤消暂存区的修改：$ git reset HEAD <文件名>

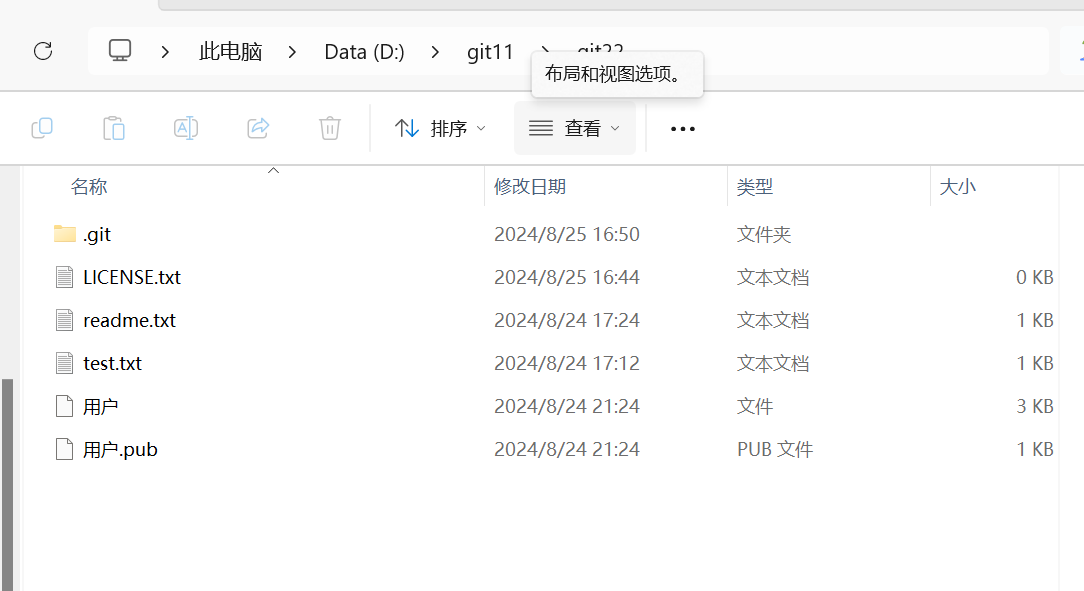
1. 删除文件

删除工作区的文件: $ rm 文件名

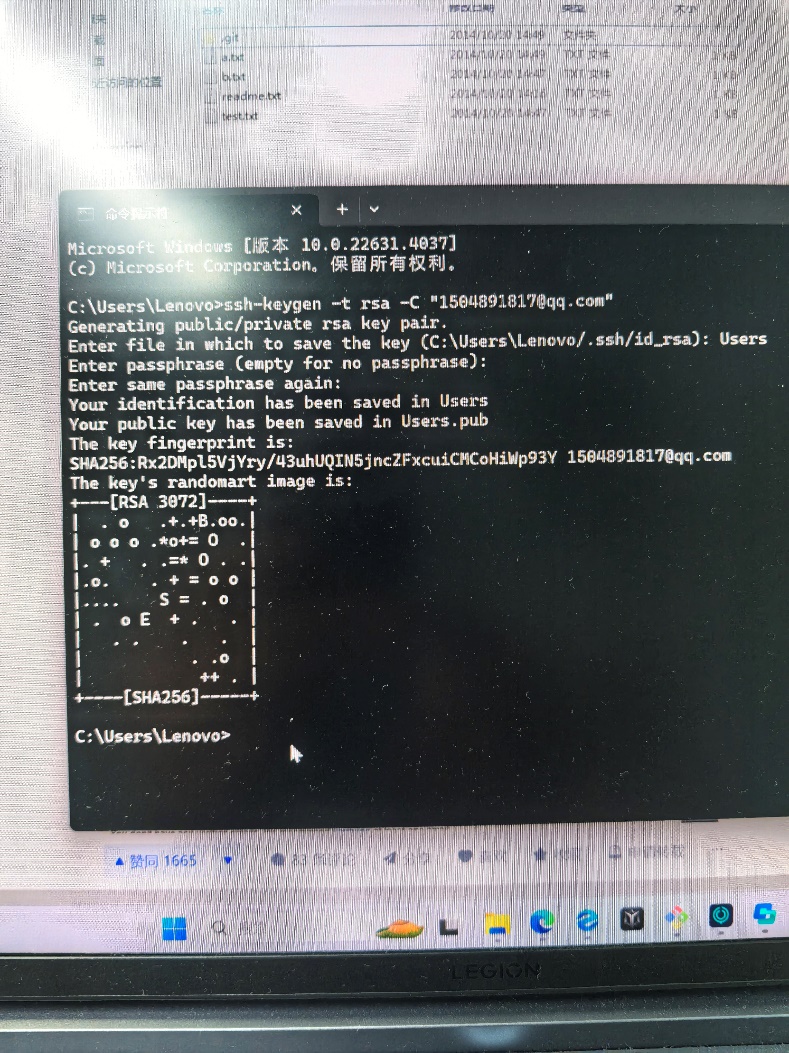
删除版本库的文件: $ git rm 文件名

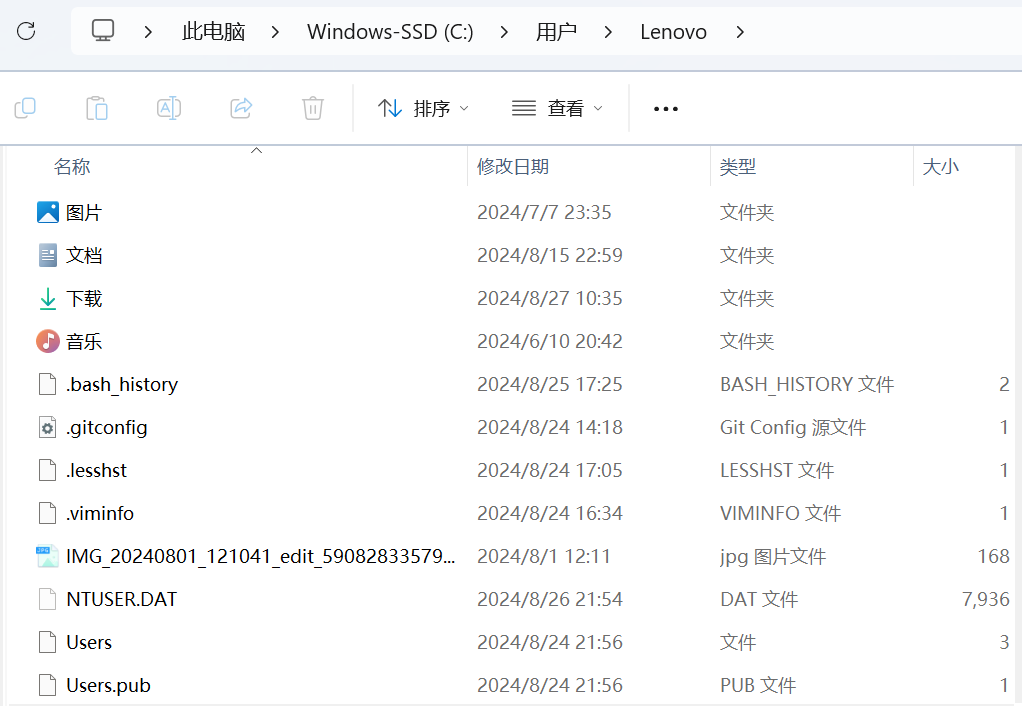
1. 远程仓库

我在创建SSH Key时，我按照说明一路回车。我在网上搜索相关信息得知”.ssh”是在用户目录之下的，所以它喊我输入文件，我就输入了“用户“，于是就发生了……



这个key出现在了master文件里面，后来我又在网络上搜索，想到“用户“有时是以”Ussers“的形式出现，于是我在命令提示符中用了相似的操作，又发生了以下情况……





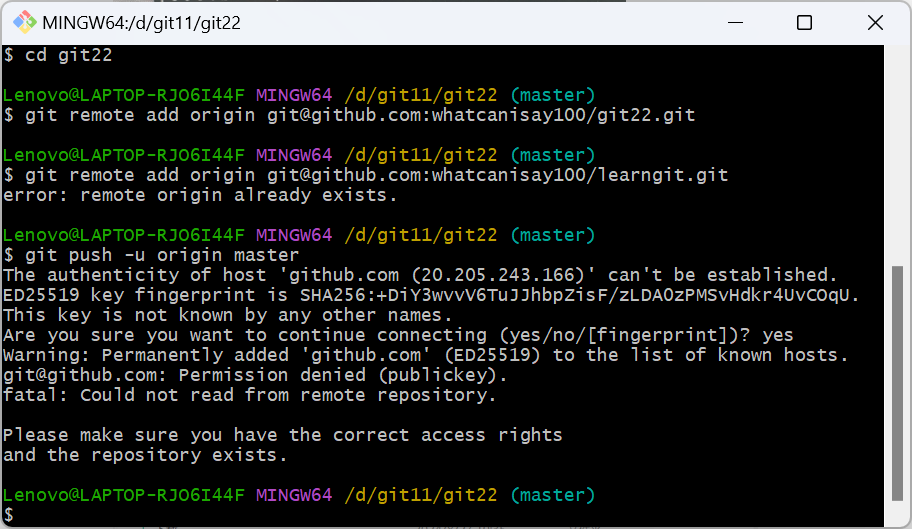
它这次确实出现在了“用户‘下面的“Lenovo”里面，并且这两次文件的名字都不叫“id\_rsa”，而是叫我输入的名称，我现在有两个key，我着实不懂背后的原因。

1. 添加远程库

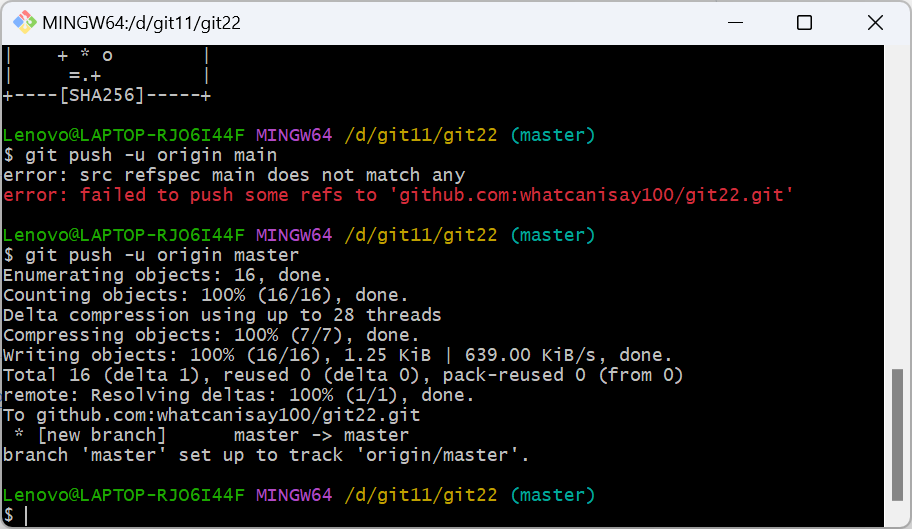
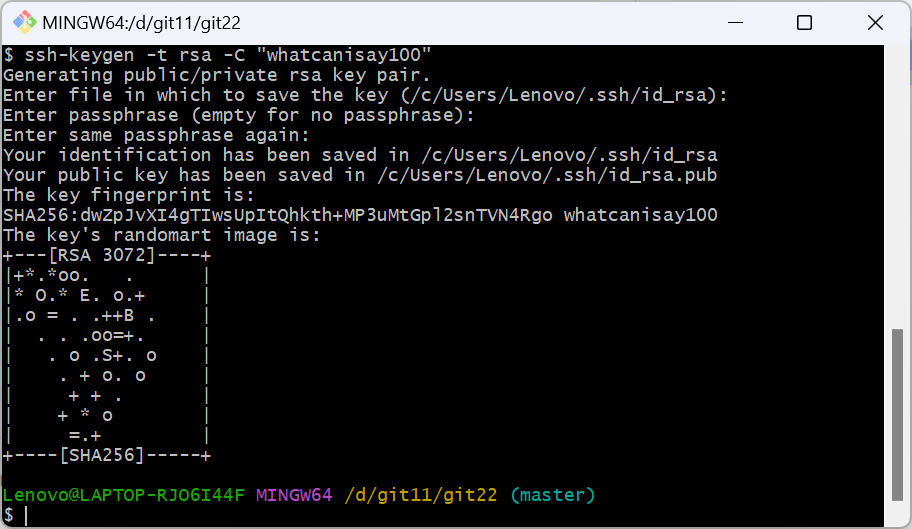
我在完成此操作时，在github上创建了一个名叫“git22“的库，因为我把工作区的git22文件设为了master，接着在git bash上输入了

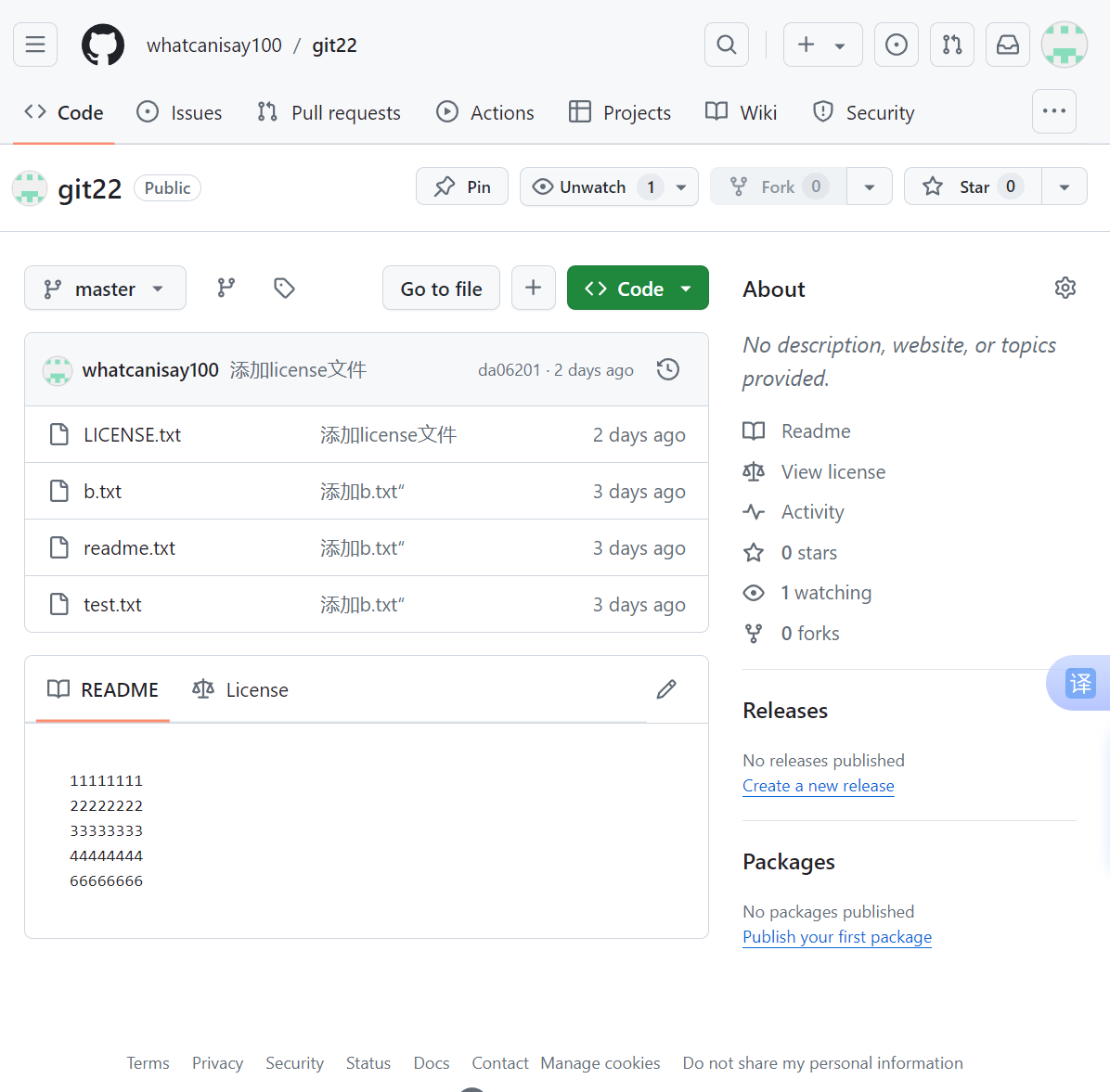
$ git remote add origin [git@github.com:michaelliao/git22.git](mailto:git@github.com:michaelliao/git22.git)

于是就发生了这种情况

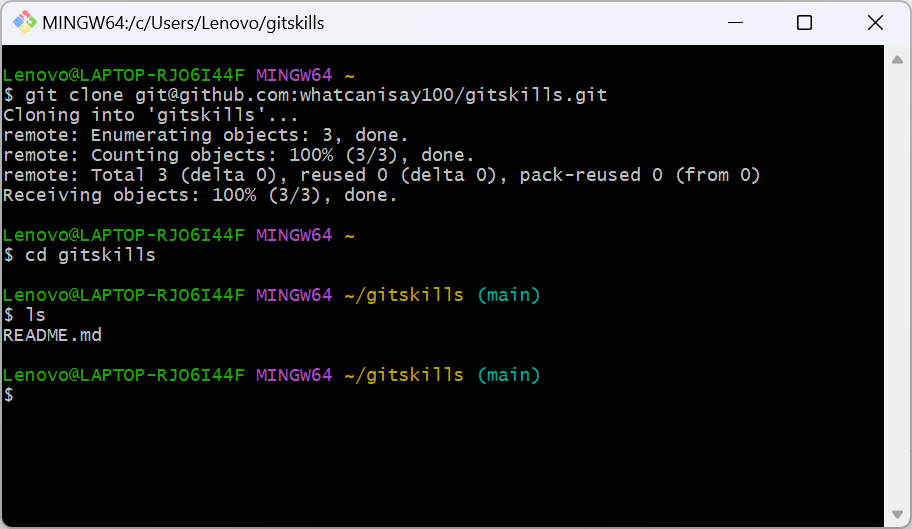


让我确定我有正确的访问权限和存储库存在，于是我撤回了我的key，并重新创建了一个，终于成功了





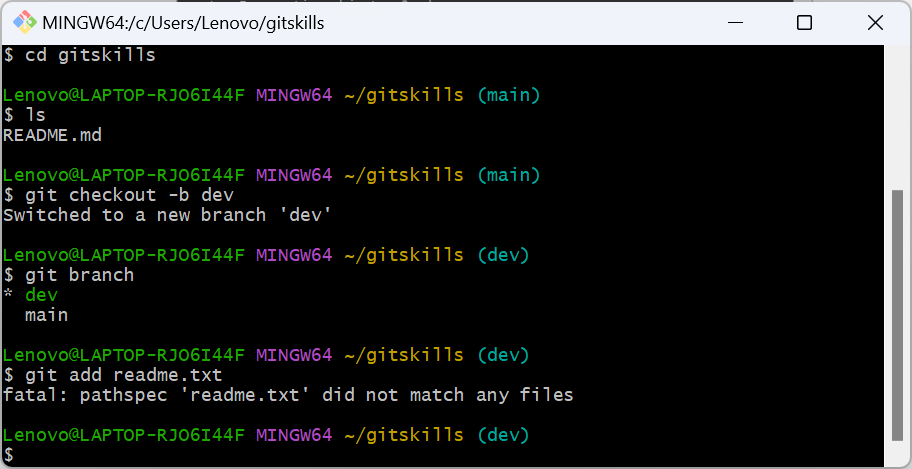
十一、从远程库克隆



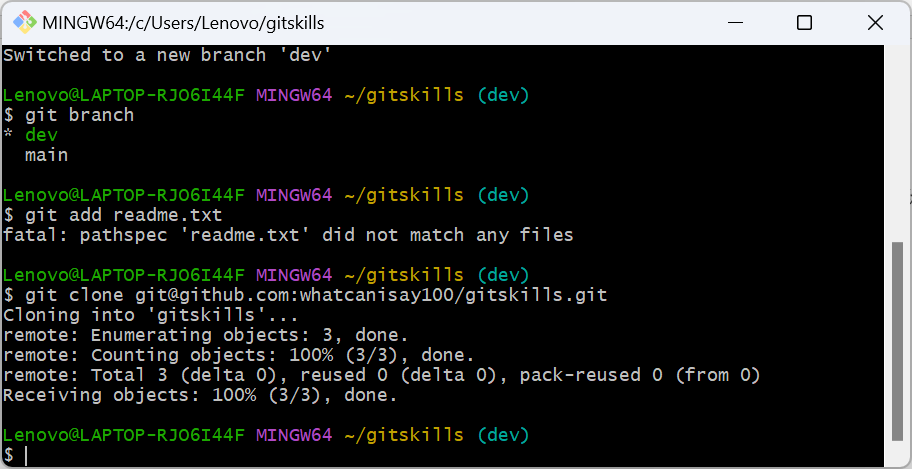
按照说明顺利完成，这对于一个项目的分工协作有着重要意义。

十二、创建与合并分支

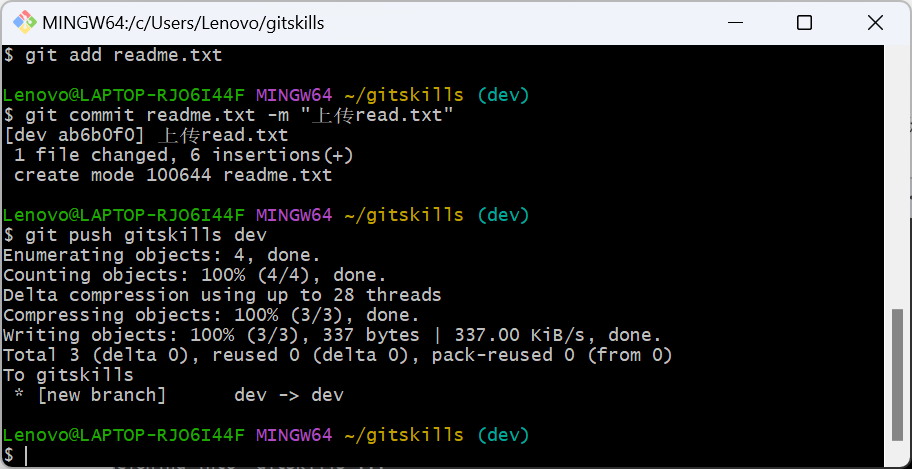
我在按照说明创造一个新的分支后却遇到了问题，在我修改工作区文件，想要传到暂存区时，发现没有匹配的文件



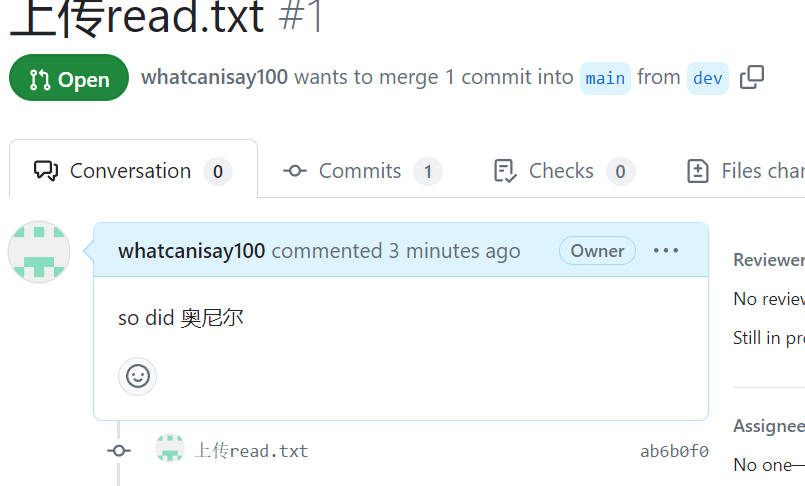
后来我查阅资料得知，在github中创建一个新库后，在工作区并不会产生一个与之名称相同的文件，加深了我对github的了解。接着我尝试将仓库克隆到本地计算机。



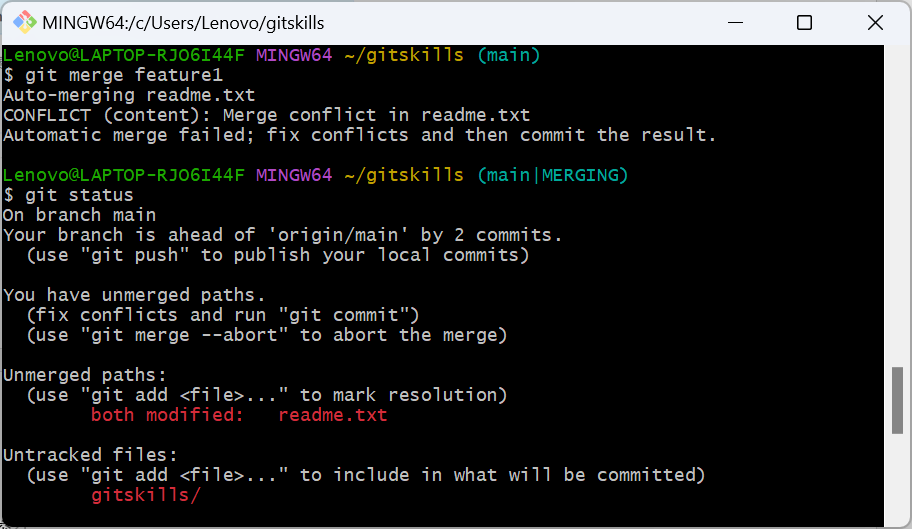
将readme.txt复制粘贴到dev，再上传到库中就可以了。



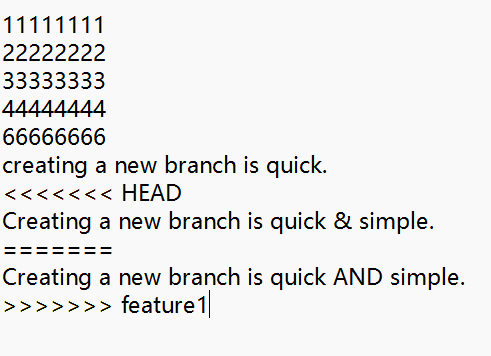
但我在github中的gitskills分支是main，对dev的修改不知为何变成了把对dev的修改整合到了main分支中，很神奇。



十三、解决冲突



按照廖雪峰的说明到了这一步，由于合并出现问题，gitskills出现了“MERGING”的标志。





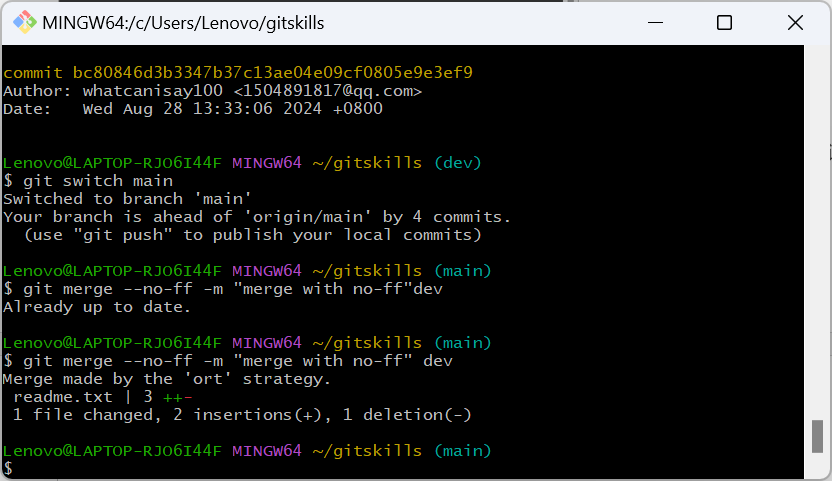
此节完成比较顺利，最后通过图表的形式能直观的看出分支变化的脉络。

十四、分支管理策略

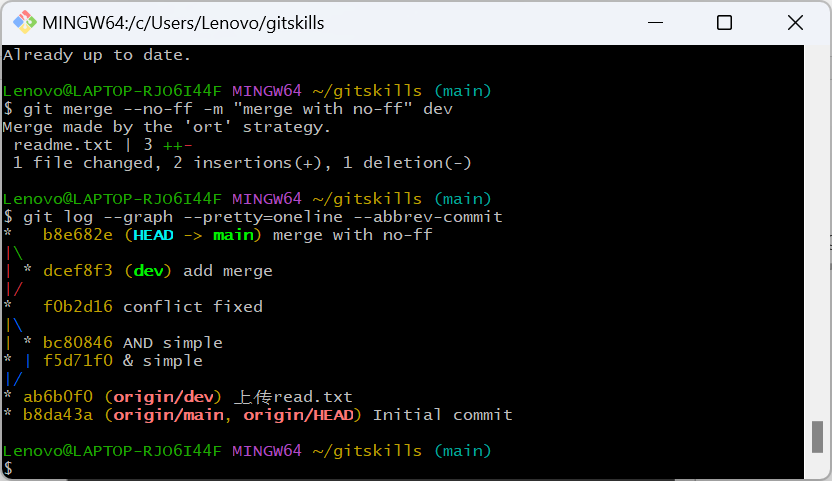
注：用”fast forward”模式，删除分支后，会丢掉分支信息。

直接输入git log 会陷入很长的修改日期长列中，此时需要在冒号后输入”q”以便推出该状态。

按照廖雪峰的教学一步步完成，发现廖雪峰的笔记中是“recursive策略合并”，而我的git bash上显示的是”ort策略合并”，通过夸克我大体得知ort是一种更加新颖优越的策略。



最终得到的结果与廖笔记上的大体相同

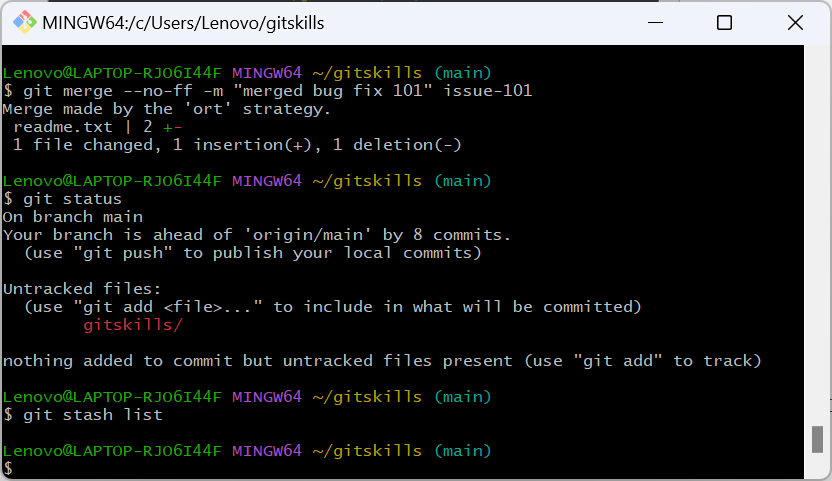


十四、Bug分支

Git stash apply:恢复stash内容，且stash内容不删除。

Git stash pop:恢复发同时将stash内容也删除了。

但我没有用git stash pop，在我输入git stash list后，并没有产生任何stash内容，不知道在哪个地方搞错了。或许是我一开始使用stash之前就没有做任何任务。



十五、Feature分支

Git branch -D <name>:用于丢弃一个没有被合并的分支。

Feature分支和bug分支是类似的，通过分支开发新功能或是修bug，然后合并，再删除分支。

十六、多人协作

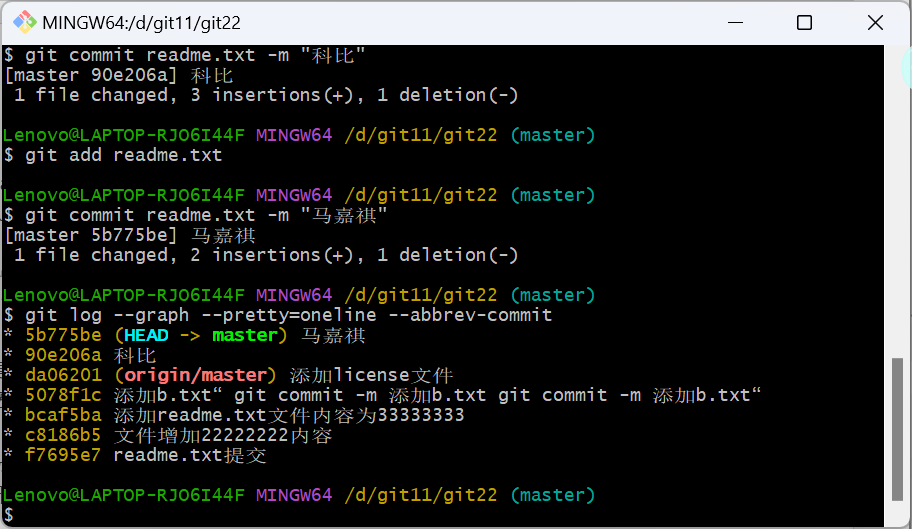
Git pull:从远程库下载更新内容，并立即将对应内容更新到本地仓库，可用于解决远程库的合并冲突。

将远程库分支创建到本地,以便修改dev，再上传到远程：git checkout -b dev origin/dev

若推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并。

十七、Rebase

多人工作在同一分支上协作时，遇到冲突后会让分支看起来乱，而rebase操作就能让git提交记录变成一条干净的直线。

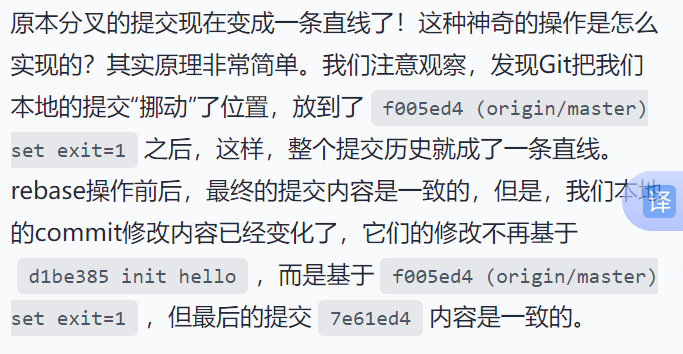


(HEAD -> master)表示我当前在master分支上，并且master分支包含了我最近的提交，还可以帮助我快速定位到当前分支的最新点。

疑问点：



我们在本地库上进行了两次提交，因为其他人先推送到了远程库，通过git pull合并了远程库更新的内容，则我们本地库与远程库的差别应该还是两个提交，为什么会比远程分支快三个提交？



这段话指的时我提交这个步骤是在A之后，还是B之后的改变，最终结果不变，但修改历史是变了的，是这个意思吗？

十八、标签管理

Tag不同于commit号的复杂不易记住，它是一个让人容易记住的有意义的名字，与某个commit绑在一起。

十九、创建标签

给某次提交打标签：git tag 标签 commit id

查看标签的信息：git show 标签

创建带有说明的标签：git tag -a v0.1 -m “version 0.1 released” commit id

二十、操作标签

删除标签：git tag -d 标签

推送某个标签到远程：git push origin 标签

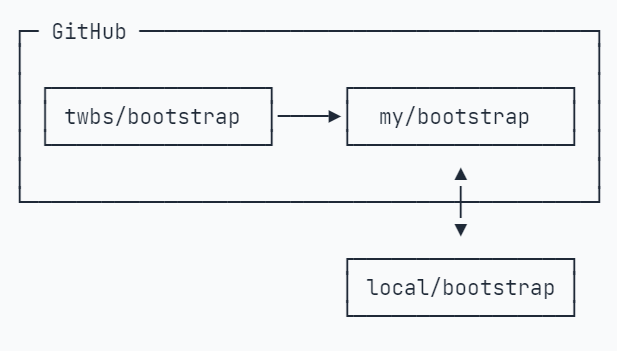
一次性推送全部尚未推送的本地标签：git push origin –tags

删除远程的标签：先删本地标签：git tag -d v0.9

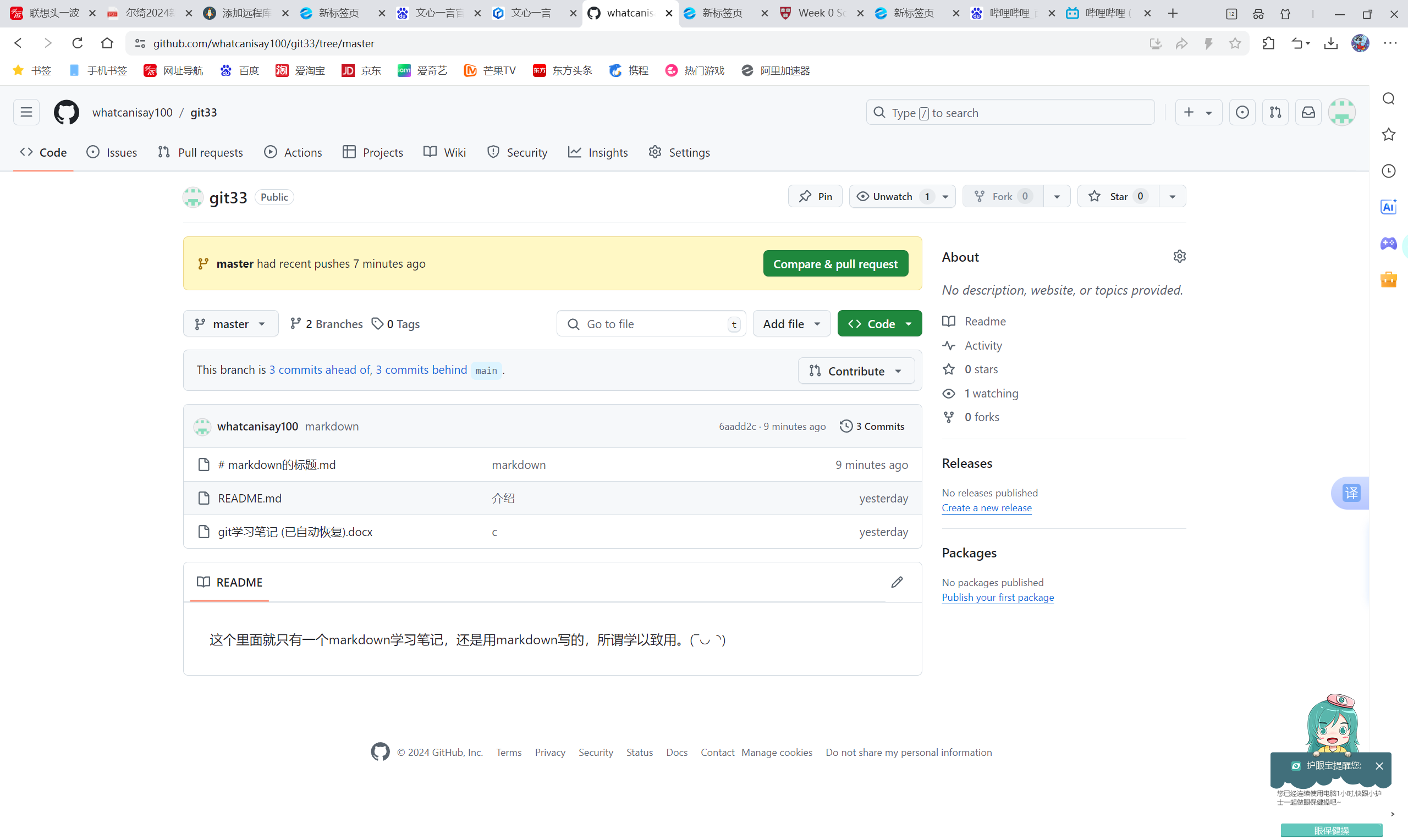
再从远程删除：git push origin :refs/tags/标签

二十一、使用github

若要参加开源项目，先要用“fork”在自己账号下克隆一个，然后从自己的账号下克隆。关系如下图：



最终：



https://github.com/whatcanisay100/git33/tree/master