上一讲开始，我们已经搭建好了一款企业级的gitlab服务器，上面也建好了部门和员工，还有项目

这一讲开始，我们就基于gitlab服务器，来实战企业中针对不同规模的团队，或者不同类型的项目，实战演练不同类型的git工作流

功能分支工作流：5人以内的小项目，面向内部的那种需求不是很频繁的内部系统

集中式工作流：2人以内的项目，1人单独维护的一个项目

某2个同学，负责维护了一个小型的工具类的系统，比如发短信/邮件/消息的这么一个工具；某2个同学，负责维护大数据计算作业类的工程，比如说spark作业，spark就是一个工程，然后在里面你要去维护一套代码，但是这套代码的话呢就是实现一套不同类型的计算任务

比如spark工程中，有的作业是用来分析用户行为的；有的作业是用来分析用户活跃度的

不需要分支，直接每个人都是这个项目的master，都可以直接push代码，需求更新频率也很低，不是经常要修改代码的

两个人都直接基于master分支去做开发，也不涉及什么code review，pull request

实战一下集中式工作流：

（1）张三和李四都在master分支上

（2）张三先修改一下代码，然后直接push到gitlab上去

（3）李四也修改代码，同时跟张三修改了同一行代码，跟张三会出现冲突，此时push失败，要求先pull，pull之后会要求解决冲突

（4）解决冲突，提交代码，再次push

基于gitlab的集中式工作流，就是这么玩儿的，说实话，因为就一两个人开发，所以不需要使用太多的gitlab的功能，就是基于一个master分支开发，一条路走到黑

梳理和总结

1、不管是谁，不管在哪里，不管是哪个分支，你做提交，其实就是在自己本地的提交历史树上加一个新的commit，然后你当前的分支就指向最新的那个commit而已

2、你切换分支，新建分支，只不过是在切换指向的commit而已；因为不同的分支是指向不同的commit的，对应的是不同版本的代码；新建分支只不过是多一个指针而已，指向当前的这个commit

3、在本地基于各种分支完成了代码开发，每个分支指向对应的一个commit之后，完成了自己本地的提交历史树的修改了，就可以将本地的提交历史树，推送到远程仓库，跟远程仓库的提交历史树进行合并

4、此时远程仓库会和你的本地的提交历史树保持一致

5、其他研发人员跟你一样，同时可以从远程仓库拉取代码，也就是拉取提交历史树，跟自己本地的提交历史树进行合并，可能会出现代码冲突。你和其他人都对同一个分支进行修改，此时你们俩的提交历史是不一样的。对你来说，一个分支是指向一个commit，对他来说，同一个分支是指向了不同的commit。此时就需要对这个分支指向的两个commit进行代码合并，可能需要解决冲突。合并之后会出现一个新的commit，就是合并了你们两个人的代码的一个commit，然后分支指向那个commit。

853769620