就是上一讲结束之后，其实日常工作中，我们最常用的git这块的技术和工作流程，包括背后的原理，就全部结束了

写代码，提交代码，推送代码到远程仓库保存一下

多人协作的流程，搭配上我们的企业中的软件开发流程

背后的原理：三个区域，提交历史树

但是。。。git还远远不止这些东西啊。。。

尤其是说，在日常的开发过程中，可能我们不一定会每天都用到部分高级技巧，但是在你的整个项目发展的生命周期过程中，很可能遇到一些特殊得情况，此时是需要使用git的一些高阶技巧，来辅助我们完整一些实际工作中遇到的问题

从19讲一直到32讲，都是git的高阶实战技巧，我讲解的时候，会做到一个的标准，绝对不会说是一个一个git命令和语法去讲解，那样的话太枯燥了，你都不知道怎么用

我在讲解git高阶实战技巧的时候，一定是说，我过往的某个实际工作场景中，真正遇到过的问题，作为一个企业实战场景引出来，接着来学习某个git高阶技巧如何来解决这个问题，而不是会像其他的一些课程或者讲师那样，一个命令一个命令的去讲解

=================================================================================

比如我们在用reset回退到某个版本的时候，可能会使用commit的完整的40位的hash值来指定某个commit，但是实际上，我们是有更好的一些办法来选择一个或者多个commit。

版本的回退和前进，git reset --hard commit标识符

当时我们用的是完整的commit标识符，40位的SHA-1 hash值，但是在实际工作中，我们一般是不会用git的40位的SHA-1 hash值

1、短SHA-1值

我们不一定就是要用40位完整的SHA-1 hash值来选择一个commit，也可以只是提供hash值的前几位就可以了，至少要4位以上，同时在git仓库中就这前几位可以唯一定位一个commit就ok

就是说，我们如果提供hash值的前7位，git会自动唯一定位出来这个前7位对应的是哪个commit

实验一把

先用下面的命令查看最近几次commit：git log

然后再看一下某个commit的具体信息：git show 前7位hash置

所以正是因为这个，我们一般用git log，都会带上一个--abbrev-commit，再加上--oneline，--graph的选项，然后自动就会显示每个commit的短hash值，默认就是7位的

2、使用分支的指针指代commit

背后的原理大家都很清楚了，git最重要的原理

（1）三个区域（三棵树）：工作区，暂存区，仓库

（2）commit树+分支指针：无论在哪个分支上做开发和提交，都是在这个项目公共的一个commit树上在添加不同的commit，形成一颗越来越长的树。分支只是指向某个commit的一个指针而已。然后每个commit就代表了那个时刻下，项目的一个完整的历史快照版本。

因为分支实际上就是一些指针，所以分支是指向commit的，也可以直接用分支名称来指代它目前指向的那个commit

比如使用命令：git show master，就可以查看master分支指向的那个commit

3、git reflog

git log和git reflog的区别是啥？？？？

git log，是你当前的分支指向的commit之前的所有commit会显示出来，你的分支指向的那个commit之后的commit是不会显示的

git reflog，是会显示最近几个月，完整的一个commit历史的

所以说，在当前分支上，如果要回退到过去的某个历史版本的话，那么此时应该是用git log就可以

但是如果要回到未来的某个版本，是用git reflog

git会自动记录和追踪过去几个月内，你的HEAD指针和分支指针移动过的每个commit

用git reflog命令就能查看完整的一份commit记录，而使用git log就只能查看当前分支指向的那个commit之前的commit记录

reflog是每个研发人员自己本地的电脑上的记录，跟git log显示出来的提交历史不同，不会同步到远程仓库上去的

4、选择历史版本的特殊语法

git show HEAD^

git show HEAD~2

HEAD，指针，指向的是当前的那个分支，当前的那个分支又是指向某个commit

HEAD，就可以指代当前你所处的那个commit对应的版本

HEAD^，就是说，HEAD对应的commit的上一个commit，HEAD^^，上两个commit，HEAD^^^，上三个commit

HEAD~5，就是说HEAD之前的第5个commit

5、commit与分支

实际工作场景中经常会用到的一个技巧，比较两个分支之间差了哪些commit还没有合并？？

回答一个问题：我当前开发的这个feature/001分支，有哪些commit还没有被合并到master分支？

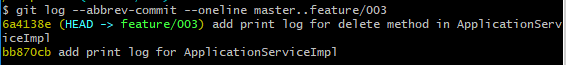
我要看一下，当前开发的这个feature分支，有哪些commit还没有被合并到master分支中去？如果知道了以后，心里就有数了，大概知道自己跟master分支之间差了多少代码。。。。

可能你开发一个feature分支，都很多天了，然后中间一度停下来，去做了另外一些事情，比如说其他的feature分支，或者是休假了。回来以后都忘了自己跟master分支之间差了哪些代码了，此时你就可以用这个方法，看一下

使用master..feature/001语法，就可以查看哪些commit是feature/001分支可以触达的，但是是master分支无法触达的，此时就可以看到feature/001分支做的那些改动哪些还没有合并到master分支里去

当前，我们处于feature/003这个分支，然后我们此时就要看一下说，我都忘了，我在这个分支上开发了多少代码，多少个commit了

git log --abbrev-commit master..feature/001



同样可以看一下，feature/001分支比master分支落后了哪些commit，不是需要跟master保持一下同步

git show commit标识符

还可以看一下，master分支有哪些commit是feature/003分支所没有的，这个是个什么场景啊？这个场景很实用的，就是一般如果你开发一个feature分支时间过长，比如说都开发了几个礼拜了，master分支肯定代码都变化了好多了

这个时候你心里有点虚，你就担心说，会不会到时候我往master分支一合并，大量的代码冲突，心里有个底，会先看一下谁提交的，哪些commit在master上有，但是我自己的feature/003分支是么有的

git log --abbrev-commit feature/001..master

如果说你发现master上有好多的commit，别人提交了，你还没有，此时你应该做一下将master代码反过来merge到你的feature/003分支上去，让你的feature分支的代码跟master分支的代码保持一致，保持同步

此外，还可以看一下，哪些本地master分支的commit需要被推送到远程仓库中去，这个的意思就是说，如果你在本地的master上做了不少的提交和修改，可以看一看有哪些代码是还没有推送到远程仓库里去的

git log --abbrev-commit origin/master..HEAD

这个用法其实常见于不是master分支，就是如果你在本地开发一个feature分支，然后开发了几天，结果不小心忘了推送commit到远程仓库，某一天你想起来了，此时你就可以看一下，你忘了推送多少个commit到远程仓库的feature分支去

==================================================================

比较少用

还可以看一下，在两个feature分支中都有的commit，但是在master分支中还没有

git log --abbrev-commit feature/001 feature/002 --not master

同时也可以看一下哪些commit是存在于两个分支中的某一个，但是不是两个分支都有的

git log --abbrev-commit feature/001...feature/002

853769620