Git，版本控制系统，VCS，version control system

1、什么是版本控制系统？

先说说到底什么是版本？？

比如说看下面，这个文件，就是HelloWorld.java的第一个版本

public class HelloWorld {

public static void main(String[] args) {

System.out.println("hello world......");

}

}

修改了一次代码，HelloWorld.java变成了第二个版本

public class HelloWorld {

public static void main(String[] args) {

System.out.println("hello world......");

}

public String sayHello(String name) {

return "hello, " + name;

}

}

过了几天，又修改了一下这个文件的代码，变成了第三个版本

public class HelloWorld {

public static void main(String[] args) {

System.out.println("hello world......");

}

public String sayHello(String name) {

return "hello, " + name;

}

public String sayHello() {

return "hello, welcome......";

}

}

抽象一下，什么是版本？一个代码文件，多次修改，每次修改都是这个代码文件的一个版本，每个版本的代码都是不一样的。

这个版本，就是版本控制系统中，指的那个版本。

那么版本控制是啥意思？

此时HelloWorld.java处于其第三个版本的状态，一共有16行代码。但是此时问题出来了。比如这个HelloWorld.java代码上线运行了一下，后来发现说，不对啊，这个版本的代码好像不是我们所期望的，这个时候如果需要你快速的回到上一个版本的代码，然后重新修改后再次上线。这个时候怎么玩儿？

如果你可以通过你的人脑完成几百行，几千行，甚至几万行，几百万行代码，迅速通过你的记忆，还有你手工修改，回退到上一次修改之前的那个代码状态，我可以打赌，你绝对可以上最强大脑。但是问题是，普通的人类，大脑功能没有那么的强大。

此时，你就需要进行版本控制

（1）要记录下来一个文件修改过程中的每一个版本

（2）可以针对每一个版本进行比如说，版本的回退，多个版本的代码的差异的比较。我跟大家举个例子吧，你现在在做一个代码开发，但是发现写完代码之后，坑爹了，就是怎么测试都不通过，都是各种报错。着急了，我现在写的这个代码跟之前提交的代码之间的差异是什么？我到底改了哪些东西。。

（3）自己做版本控制，行不行？没问题

（4）有问题，太麻烦了，如果你手头在做一个很大的项目，有几千个，甚至几万个，几十万个代码文件，同时最后又几百万个，几千万个版本，你怎么玩儿？通过手工，根本没法弄

那么版本控制系统又是啥？

有这么一套系统，然后呢，我们是这样，就是说每次修改代码之后，就将这次修改后的这个版本提交到那个版本控制系统中去。然后由这个版本控制系统来给我们管理我们代码中每个代码文件的所有的版本。

当我们需要进行版本控制的时候，比如回退一个版本，此时直接向那个版本控制系统下达一个命令，说，我现在需要切换回上一个版本的代码，你给我在1s之内完成这个事儿。然后版本控制系统同，就会自动找到上一次提交的版本的代码，然后恢复到你的电脑上来。

2、版本控制系统的演进历史

（1）本地版本控制系统

最早的版本控制系统了，比较知名的是RCS，就是在本地对你的文件的每次修改维护一系列的patch，每个patch就是两个文件版本之间的修改。然后如果你要回到某个历史版本上，RCS会通过path的应用来恢复任何一个时间点的文件。

这个方式比较Low，常见于上世纪八十年代，就一个程序员写一个系统，编程，只要自己可以管理自己的代码版本即可，不需要跟别人协作。

但是如果你要跟其他同学去协作共同修改一份代码，来完成某个系统的开发，此时用这种就不可能了。

（2）集中式版本控制系统

后来人们遇到的问题就是，不是只有自己需要在本地对文件版本进行控制，尤其是软件开发人员，需要多人写作，对同一份代码进行版本控制。比如每个人都要对代码进行修改，那么每个人都会更新出一个版本的代码。

这个时候就需要集中式的版本控制系统，就是弄一个服务器，上面放所有代码文件，并且进行版本控制，接着多个开发人员在本地连接服务器，进行代码从服务器检出到本地，代码的修改，以及代码修改的提交到服务器。

比较知名的有CVS和SVN，很长一段时间内，尤其在上个世纪的90年代到2000年初期，基本就是版本控制的标准。持续到了2005年，2010年之前，还是蛮广泛的。SVN，比如说，大名鼎鼎的百度，当年也是SVN去做代码版本控制。

但是集中式版本控制系统最大的问题，就在于单点故障，如果服务器故障一段时间，那么那段时间里，各个开发人员几乎啥也干不了了，没法从服务器检出最新代码，没法提交代码。如果更加坑爹的事情发生，就是服务器的磁盘坏了，结果还没有备份，那么完蛋了，所有的代码全部丢失。

（3）分布式版本控制系统

这种模式下，每个开发人员自己的本地电脑，都会从服务器检出一份完整的代码包括历史所有的版本到本地。相当于每个开发人员本地都有一份完整的代码版本的副本拷贝。此外，每个开发人员自己本地都有一个完整的版本控制系统。

如果服务器有任何故障，开发人员在自己本地可以做所有的工作，包括代码版本的切换，代码修改的提交，等等所有的版本控制操作，不会影响自己的工作。如果服务器磁盘坏了，数据丢失，那么可以将某个开发人员本地的版本库拷贝一份到服务器，去恢复数据即可。

优势是三个

1）跟集中式版本控制系统不一样的地方，在于说每个研发人员会将服务器上所有的代码和所有的版本都拷贝到自己本地来。但是集中式版本控制系统，就从服务器拉取部分代码和部分版本。

2）一旦服务器断网，故障了，我们在自己本地可以基于完整的代码和历史版本完成所有的工作；集中式版本控制系统，拉取一些代码，切换一个历史版本，都要联网，基于服务器去搞。联网的话就意味着速度很慢，既有服务器就意味着，服务器故障，就不好搞了

3）一旦服务器磁盘坏了，代码和历史版本都丢失了，没关系，直接从任何一个研发人员的笔记本电脑上，就可以恢复一份完整的代码和历史版本到服务器上去，可能会丢失一点点最新提交的代码，但是肯定是可以恢复大部分的代码的

大名鼎鼎的分布式版本控制系统，就是Git

3、Git前世今生

linux的创始人，linus，最早就是用很原始的方式在管理linux源码，开源贡献者通过diff工具把修改后的源码发给linus，然后linus手工合并所有代码，自己解决冲突。当时linus始终非常反对使用CVS、SVN等版本控制系统，因为觉得是集中式的，需要联网，很麻烦。

但是到了2002年，代码过于庞大，在社区的压力之下，linus选择了商业的版本控制系统，BitKeeper，拿到了免费的使用授权。结果不幸的是，2005年的时候，linux社区里一个哥儿们，尝试破解BitKeeper，结果被BitKeeper公司发现了，愤怒收回了对linux社区的免费使用授权。

接着linus就用2周的时间自己写了一个分布式的版本控制系统，然后linux的几百万行代码就开始使用Git进行版本控制了。一直发展到今天，Git成为了最流行的版本控制系统。直到今天，尤其是互联网公司，基本都是git，还在用svn的话，切换一下。

Git诞生的前世今生。

853769620