**Git分支**

**【1】(同时推进多个任务时利用分支完成后合并推出新版本)**

**主干是客户的线上版本不可拿来直接开发修改，这时候我们添加新功能也好修改bug也罢，都需要开一个单独的分支，不能影响主干的正常运行，分支简单理解为主干的副本**

**【2】**

图表, 图示

中度可信度描述已自动生成

**上图主干master其他的三个分支就是用来完成三个不同的更新，最后改完还是要合并到主干上的，最后生成新版本。**

**(简单理解成游戏的体验服开发不能影响游戏的主服务器正常运行)**

文本

描述已自动生成

**【3】分支命令**

**Git branch 分支名**

**Git brabch -v**

**Git checkout 分支名**

**Git merge 分支名**

**新建分支 查看分支 切换分支 将指定的分支合并到当前分枝上**

**【4】正常合并分支:**

**注意是将指定的分支合并到当前分支上，新功能做好之后需要合并到主干master上面，我们在主干上运行git merge 分支名 命令合并时写上注释(一定要写清楚)**

**【5】冲突合并分支**

**两条分支对于同一文件的同一位置做出了完全不同的修改**

**此时git无法帮我们决定取舍，我们需要认为决定谁去谁留**

**解决冲突:**

**1直接运行正常合并命令git checkout 分支名**

 **2冲突出现**

**3vim打开发生冲突的文件,决定取舍即可**

**4搞完之后直接git add .添加后提交即可**

**发生冲突的文件有几个改几个**

文本

描述已自动生成

**修改界面-----当前主干上的和分支上的，手动删除一个**

文本

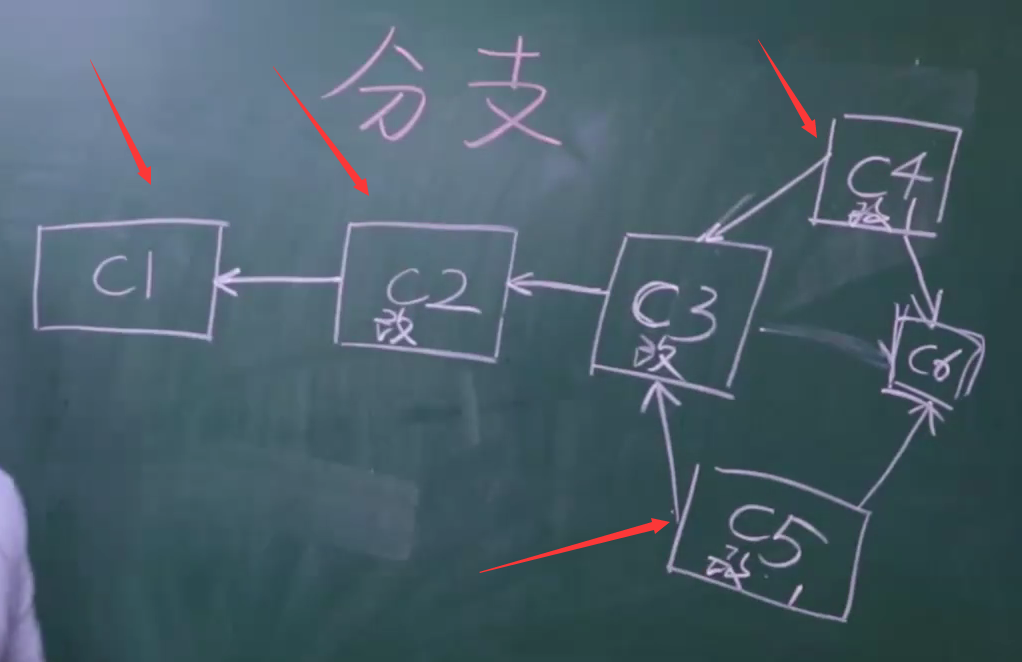
描述已自动生成

**Git内部存储结构：假设我们的项目有两个版本的话**

**Version1.0 Version2.0**

**那么我们的内部存储结构是怎样的呢？假设1.0版本上传了100个文件的话，2.0又基于1.0添加了10个文件修改了20个文件，那么我们的2.0版本只存储30个文件(修改和添加的) ，之后的每个版本都是这样的存储结构，这就是（主干），是因为内部有个指针指向上个版本（2.0指向1.0），以后要是拿2.0版本的话通过2.0自身添加和修改的（代码30个）加上指针指向1.0文件代码(100个)拿到2.0的全体的内容**

**每一次版本迭代都是通过指针指向上个版本，上个版本的内容(未修改的)加上此版本基于上个版本修改的内容拿到此版本的全体内容**



**如上图所示，c4版本和c5版本都是基于c3版本来开发的（指向c3），这两个就是分支(脱离主干)，一般用于单独隔离开发某些大项目，完事之后在合并到主干里，也或许是基于c3开发出了两个功能，最多的是正在编辑项目4.0，突然c3出了bug(由于c3在线上，所必须马上解决)，这是我们不可能放弃编辑了一半的c4，所以开展c5版本专门针对修复c3的bug每一个分支都是一个单独隔离空间，你可以利用它做任何你想做的事情（并不会对主干有任何影响）**