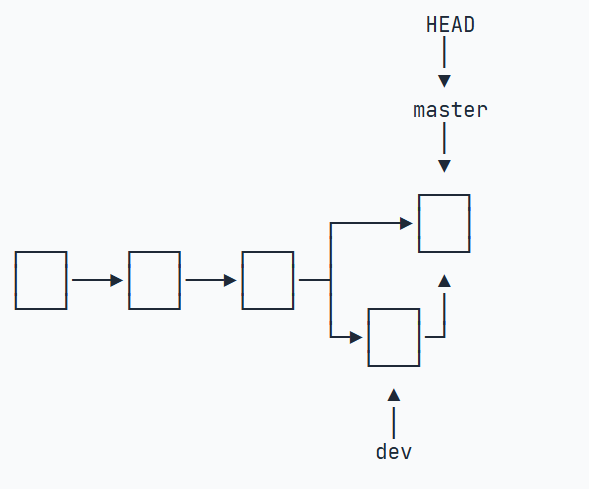
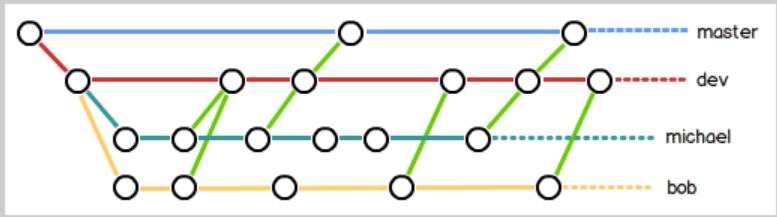
解决冲突：

1. 如果两个分支有时间线上的先后关系，采用快速合并，即merge，一个分支合并到另一个分支上。
2. 如果两个分支“并行”，也用merge，但此时会发生冲突，分支并不会归并到另一个分支上，而会产生一个箭头指向共同的后代（除非你用add，commit，那这时使用这两个指令的分支就确实归并在了该“共同的后代”上，但这个结果是add，commit的结果，不是merge的结果）。

总结：有时间先后合并在一起，没有时间先后（或有冲突）改为箭头指向共同的后代（也可能是某一分支自身）

下图（保存分支合并）





\*该语境下的主分支指当前使用merge指令的分支

快速合并：直接把分支移到主分支

非快速合并：在主分支上创建一个新的“时间轴”，copy非主分支上的内容

### **Bug分支**

工作区对于所有分支都是公用的。

Git cherry-pick和git merge：前者合并某一个修改到当前分支，后者将指定分支合并到当前分支

### 多人协作

1. 从远程库克隆到本地，会自动令本地库关联到远程库。
2. 从远程库克隆到本地，只会克隆master分支。
3. git checkout -b dev origin/dev 🡪创建远程origin的dev分支到本地。并且建立远程连接。
4. 从本地推送分支。若推送失败，说明有冲突（其他人先于你上传了），这时需要pull下来解决冲突后才能上传。
5. 使用git pull指令，失败，说明没有指定本地dev与远程origin/dev的链接。（使用的是普通的创建dev分支，如果使用git checkout -b dev origin/dev，则不会失败）

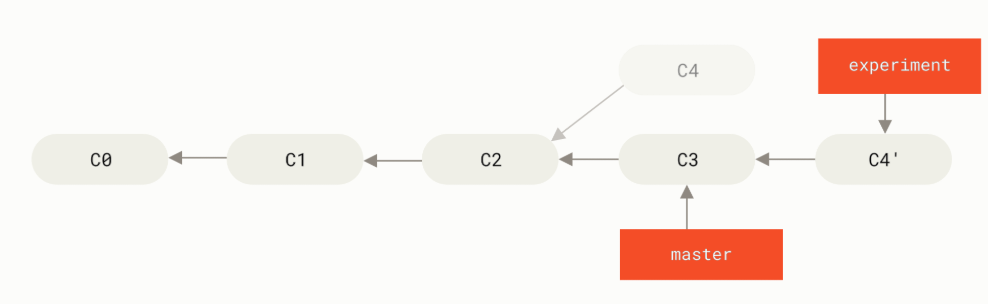
### 变基（这里的箭头和之前的反方向）

合并

### 

变基（rebase）：意即改变修改的基础

如：这里的C4本来是基于C2的基础做出的修改。使用rebase，现在C4基于C3作出修改



总的原则是，只对尚未推送或分享给别人的本地修改执行变基操作清理历史， 从不对已推送至别处的提交执行变基操作，这样，你才能享受到两种方式带来的便利。

### 标签管理

标签要主动推送至远程库

### 忽略特殊文件（.gitignore）

较简单

### 配置别名

可以在local库配置别名，也可以在配置时加上—global，这样就是对当前用户配置别名