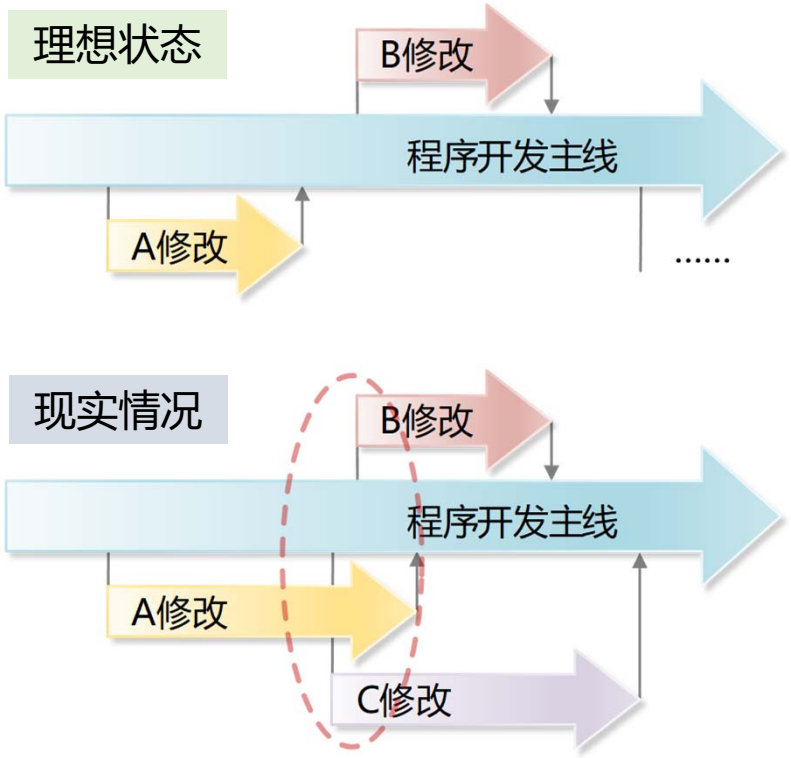
版本控制
Git & GitHub

Git 的参考资料

- VS Code+Git
- https://git-scm.com/downloads
- 书籍：Pro Git

假如没有版本控制

- 一个大型的软件项目通常是由一个团队来共同分析、设计、开发和维护的，每人负责其中的若干子模块。多人同步开发一套代码可能存在冲突。

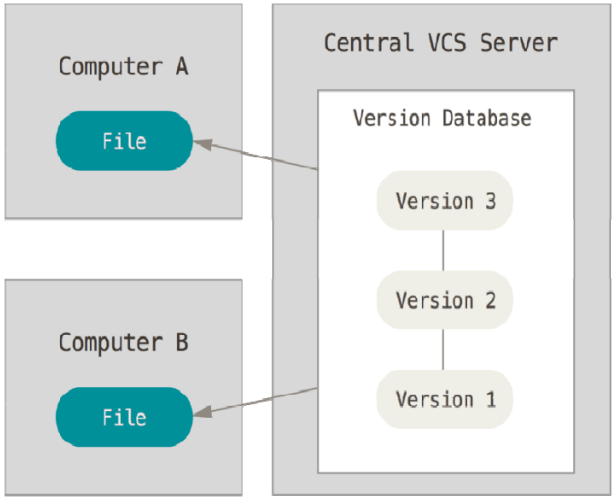


- 没有人能保证自己的开发过程不会犯错，假如没有版本控制，将某个文件或整个项目回退到某个历史状态将会变得非常困难。
- 假如没有版本控制，每个人都有权限查看并编辑整个项目中的所有代码文件。无意识地编辑他人代码可能会导致出错，有意识地篡改他人代码可能引入信息安全问题。
- 假如没有版本控制，一边进行大版本更新的开发，一边进行小版本升级的开发，将会变得非常混乱。

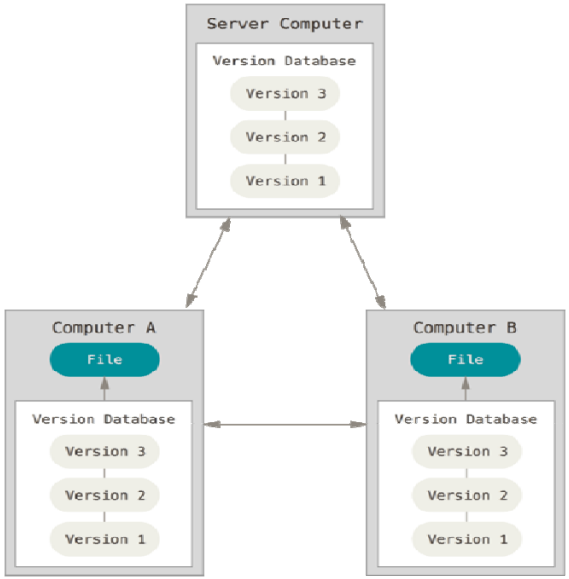
版本控制类型

基于文件的手动版本控制：最简单的版本控制——手动备份复制粘贴。非常容易出错，一不小心会写错文件名或意外覆盖文件，也很难直观看到谁在什么时间修改了哪些内容。受此启发，也出现了一些软件专门用于记录文件历次修改之间的差异，并将其保存，用于回溯。

集中式版本控制系统：设置一个的服务器用于保存所有文件的全部修改版本，而协同工作的软件工程师通过客户端程序连到这台服务器，下载最新版的文件或者提交新修改。缺点是一旦服务器崩溃，所有人都无法正常上传下载代码。

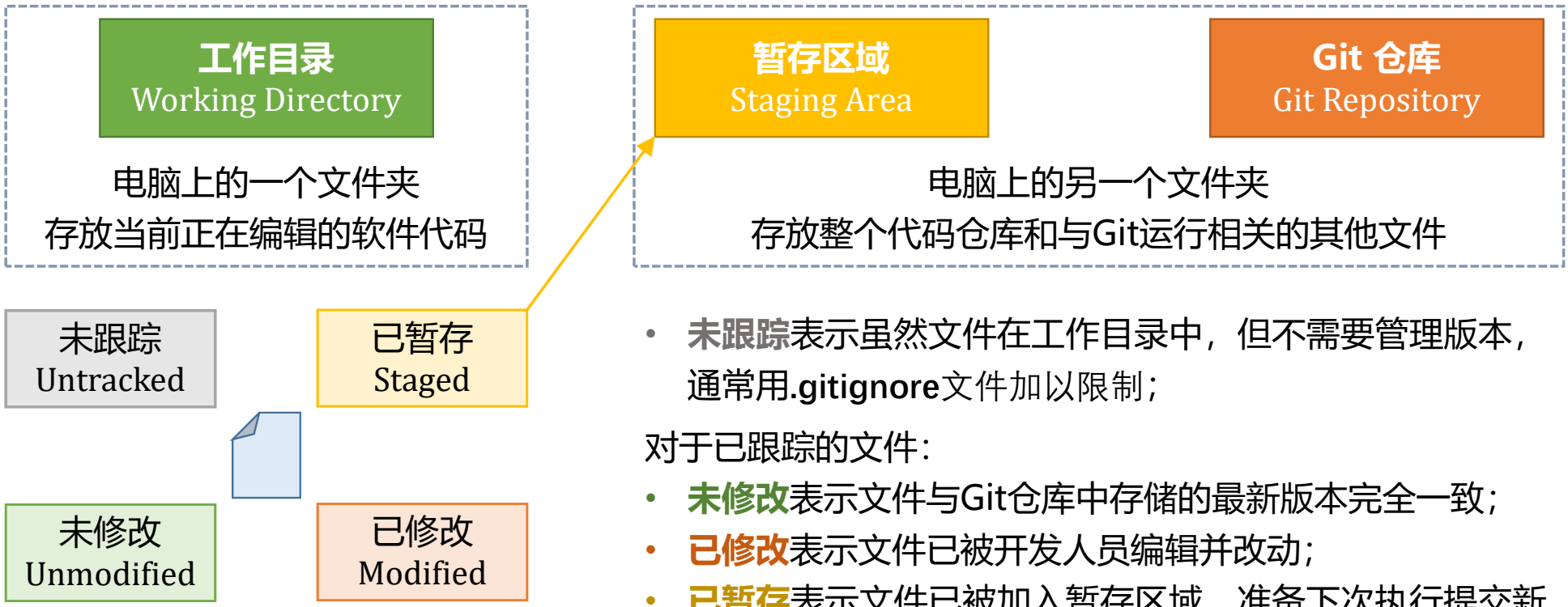


分布式版本控制系统：客户端并不只下载最新版本的文件，而是把代码仓库完整镜像下来。任何一台服务器故障，事后都可以用任何一个镜像出来的本地仓库恢复。且通常允许与不同的远端仓库交互。从而允许和不同工作小组的人相互协作。



Git 的工作区域与文件状态

一个广泛应用的分布式版本控制系统



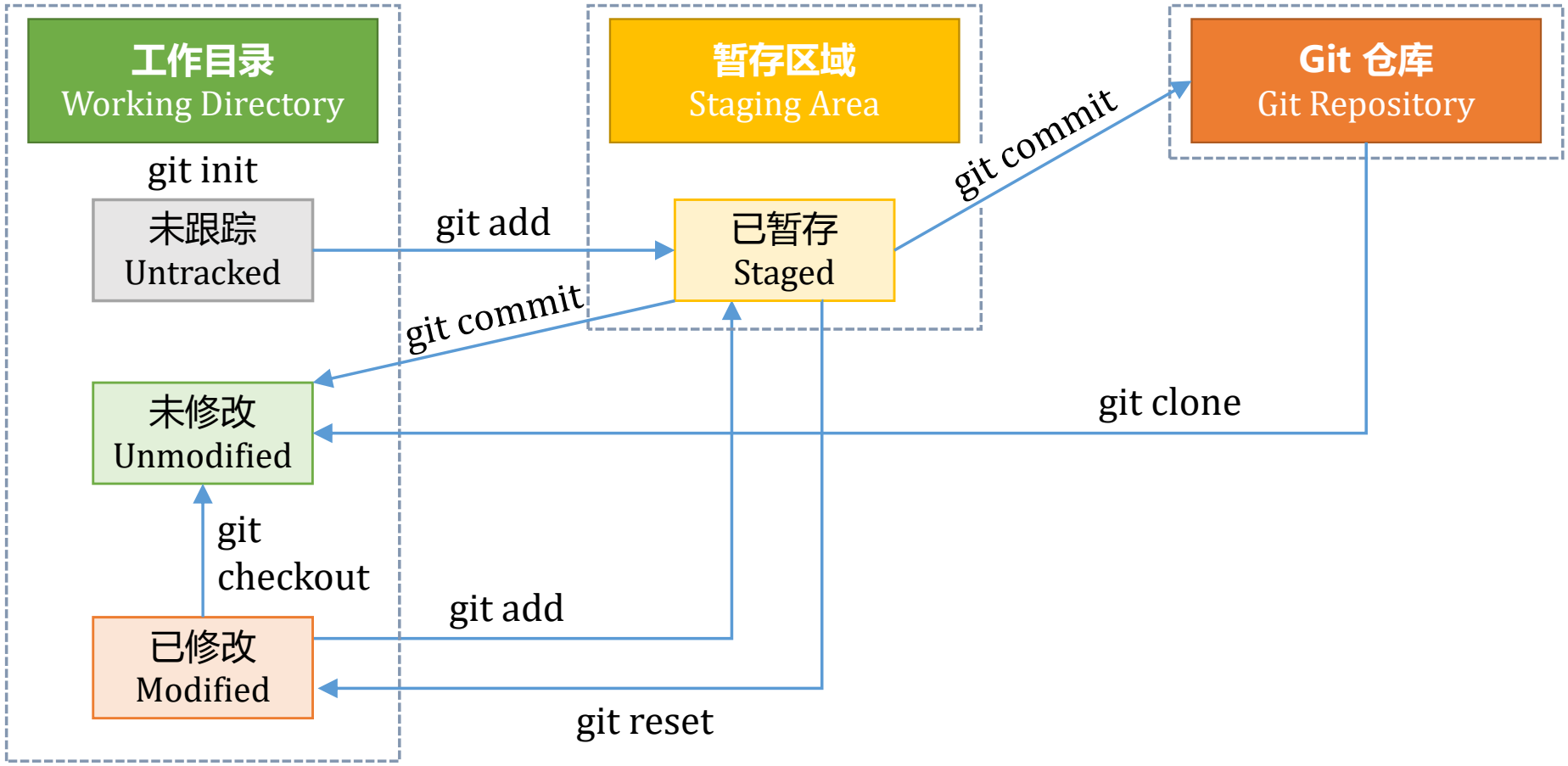
文件的4种状态

- **未跟踪**表示虽然文件在工作目录中，但不需要管理版本，通常用**.gitignore**文件加以限制；

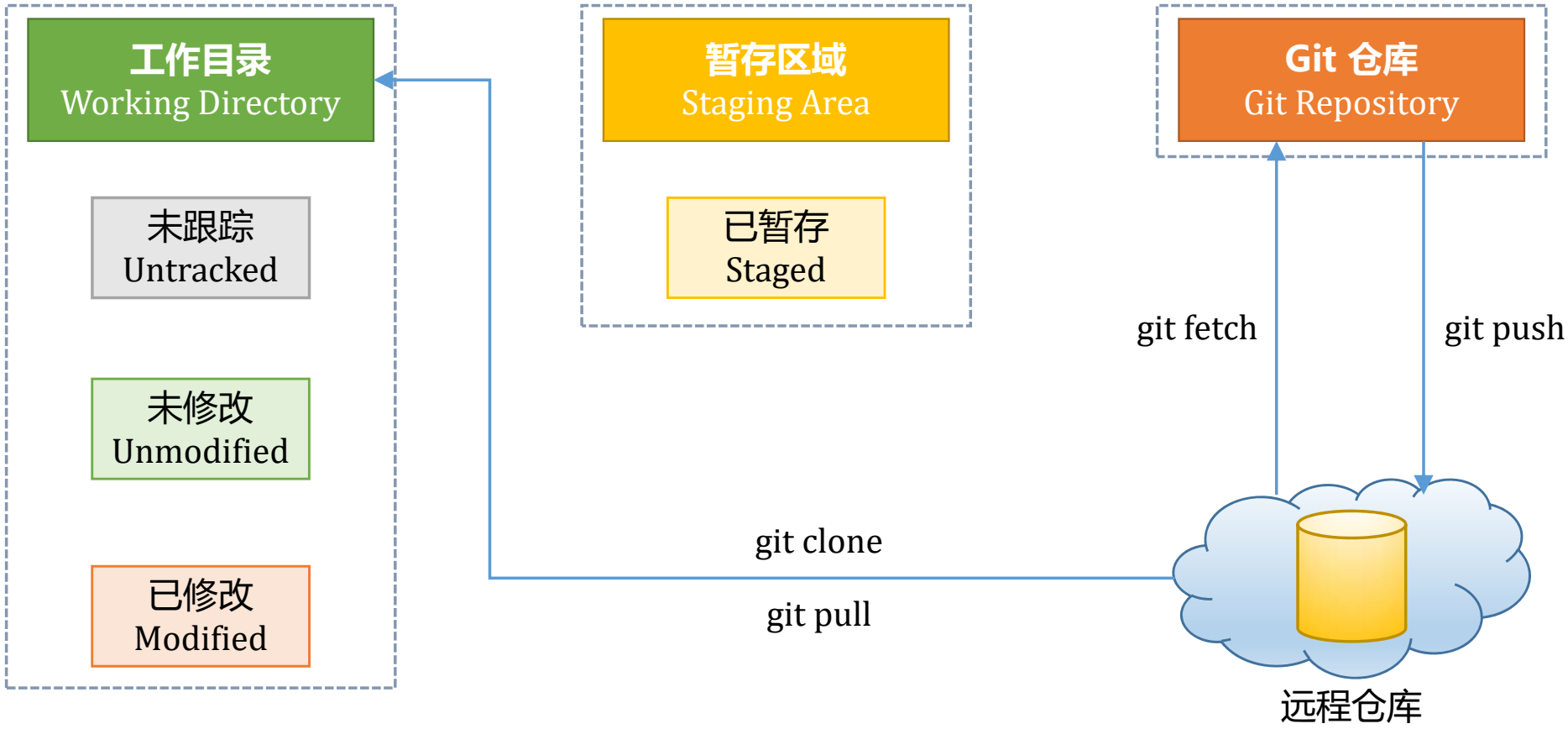
对于已跟踪的文件：

- **未修改**表示文件与Git仓库中存储的最新版本完全一致；
- **已修改**表示文件已被开发人员编辑并改动；
- **已暂存**表示文件已被加入暂存区域，准备下次执行提交新版本命令时存入Git仓库。**仅有暂存区域中的文件才会被提交至Git仓库中，所以修改后的文件需要先暂存。**

Git 的基本操作


工作目录
Working Directory
暂存区域
Staging Area
Git 仓库
Git Repository
git init
未跟踪
Untracked
git add
已暂存
Staged
git commit
git commit
未修改
Unmodified
git clone
git
checkout
git add
已修改
Modified
git reset

Git 的远程操作

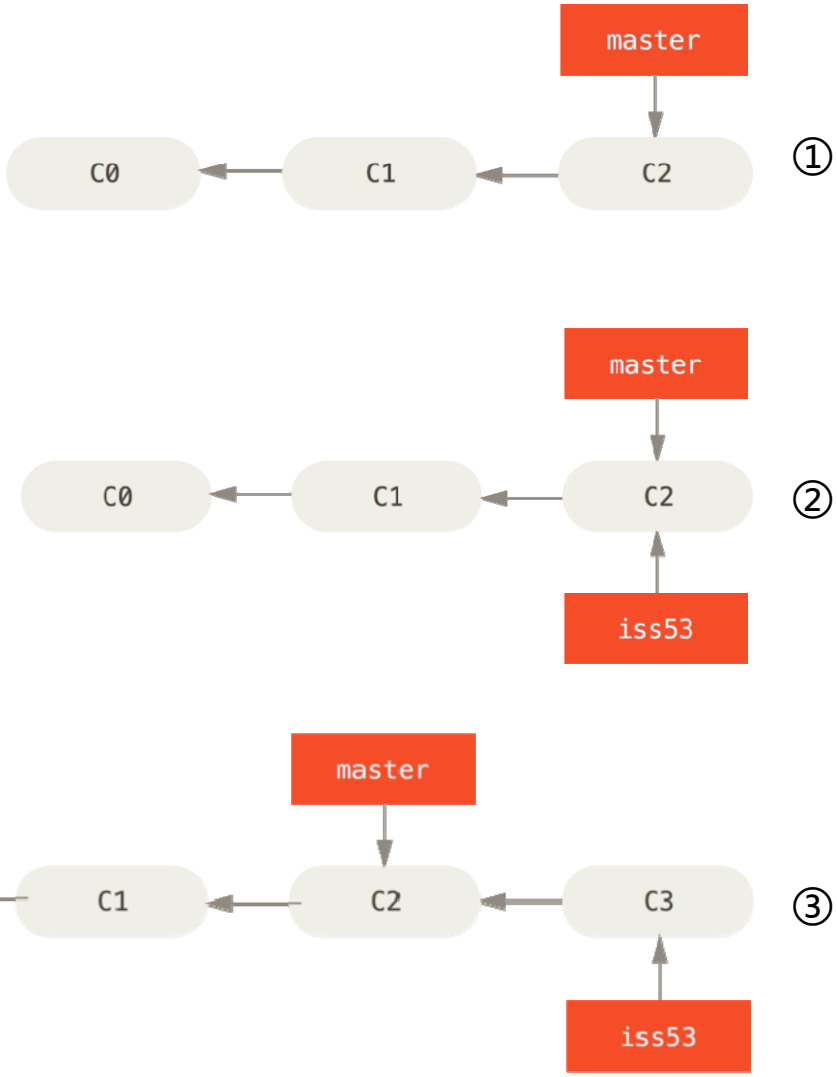

工作目录
Working Directory
未跟踪
Untracked
未修改
Unmodified
已修改
Modified
暂存区域
Staging Area
已暂存
Staged
Git 仓库
Git Repository
git fetch
git push
git clone
git pull
远程仓库

Git 的分支操作与冲突解决

- 当执行git init命令创建仓库时，会默认创建一个主分支master。每次执行git commit提交的新版本，都会使master向前移动一个版本。如图①，用户在初始版本C0的基础上提交了两次，分别对应C1、C2版本。当前主分支master位于C2版本。
- 此时，你计划进行一项新功能的开发，你可以新建一个分支iss53。执行下列命令你将得到图②。

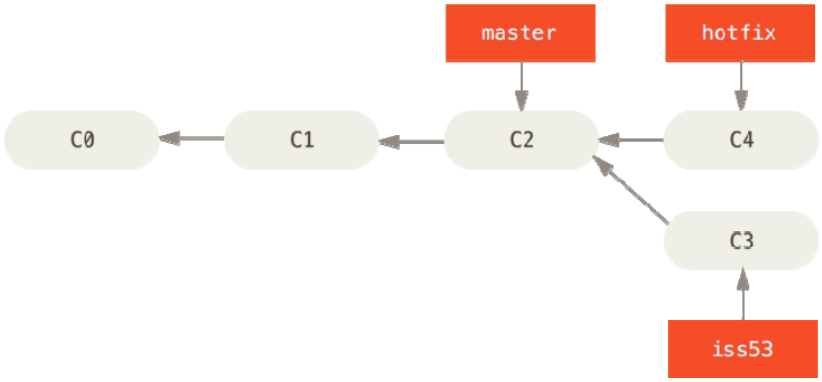
git branch iss53	新建iss53分支
git checkout iss53	将新分支检出到工作目录

- 此时你可以继续在iss53分支上工作。当你提交新版本C3时，iss53分支向前移动，而主分支master仍然留在C2。



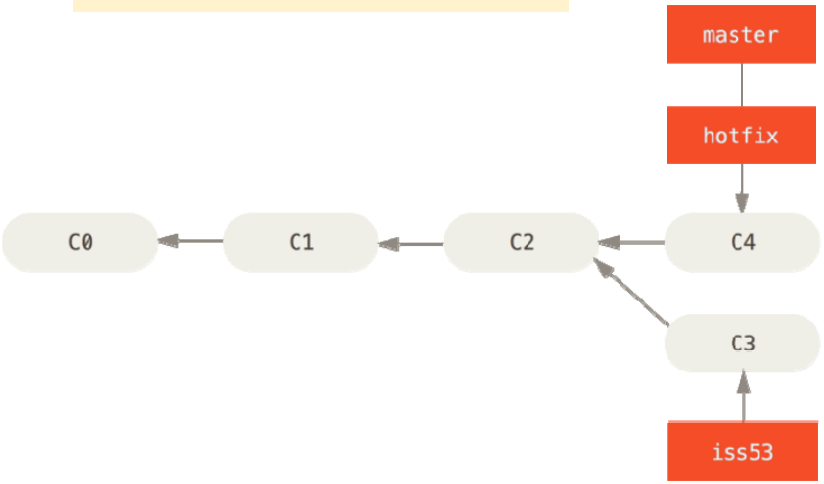
Git 的分支操作与冲突解决

- 此时你接到电话，说你提交的C2版本中有一个bug，需要马上解决。此时你可以执行git checkout master切换回主分支，再执行git branch hotfix新建一个新分支hotfix，用于修改bug。
- 当你修改完bug，在hotfix分支上提交C4版本时，你的工作流如图所示。



- 由于这个bug是针对C2版本做的微调，并不涉及到iss53分支，所以当bug修改完成后，你希望将hotfix分支和master分支合并。此时需要执行

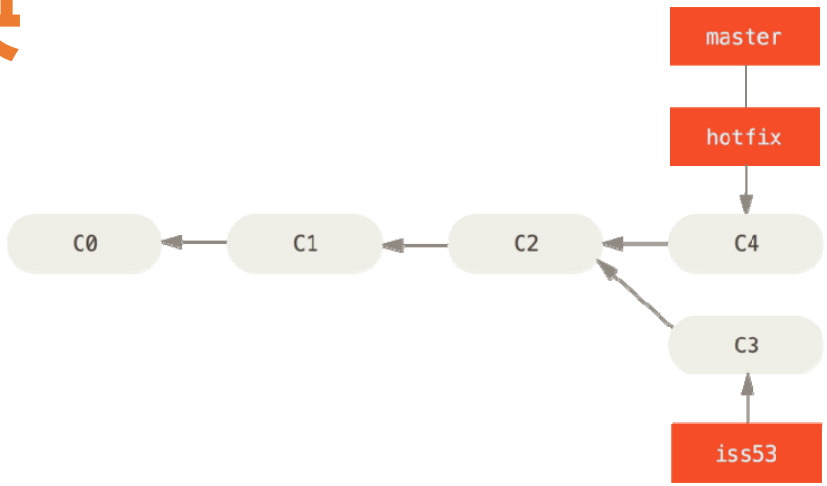
```
git checkout master
git merge hotfix
```



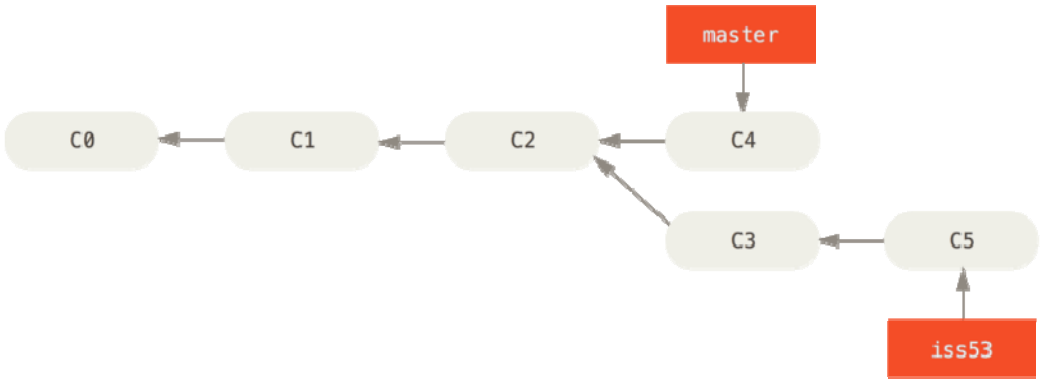
Git 的分支操作与冲突解决

- 由于hotfix分支就是在master分支上衍生出来的（即master是hotfix的直接上游），分支合并将非常顺利地自动完成。
- 此时master和hotfix指向同一个版本C4。执行下面的命令可以删除hotfix分支。

```
git branch –d hotfix
```

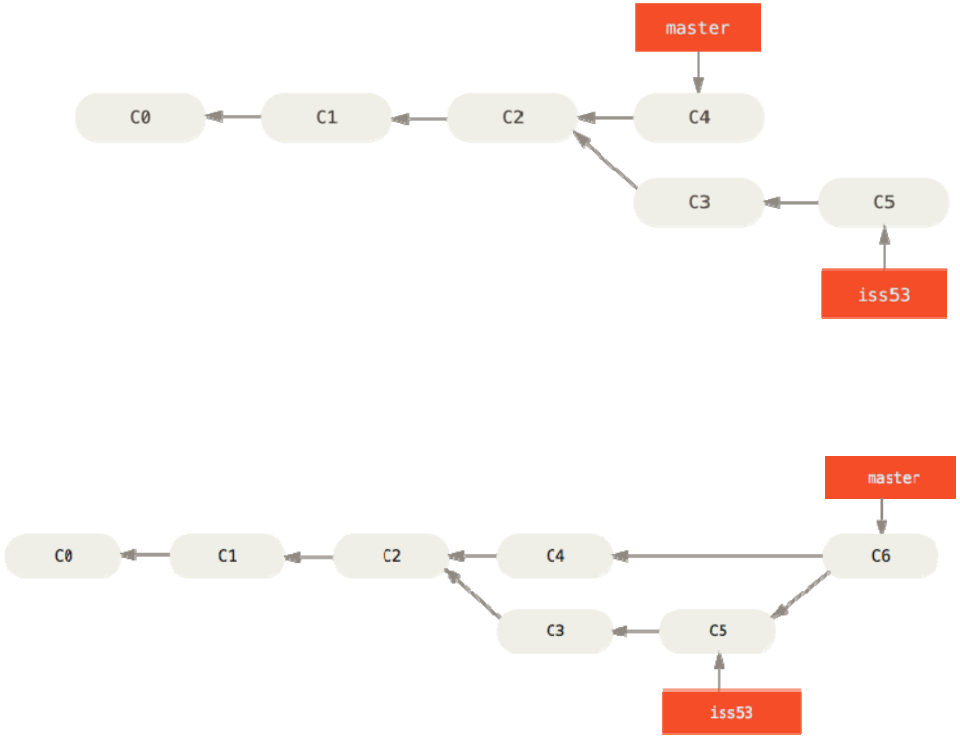


- 这时你想继续开发新功能。你只需执行git checkout iss53，即可回到之前的C3版本。
- 修改并提交新的C5版本，iss53分支将会继续向前移动。



Git 的分支操作与冲突解决

- 新功能开发完成，需要将iss53分支与master分支进行合并。由于此时master分支不是iss53分支的直接上游，当执行git merge命令时，Git会首先找到它们的共同上游C2，并与master的最新版本C4和iss53的最新版本C5进行比较。
- 如果C4和C5分别对C2中的不同文件或是相同文件的不同区域做了修改，Git可以自动将这些修改合并进C2中，不需要任何人工干预。
- 如果C4和C5都针对C2中相同的位置进行了不同的修改，Git不会自动合并分支，而是生成一个冲突文件，提示用户手动修改分支。用户修改完成后，可执行git add和git commit命令，手动完成分支合并。



- GitHub是全球最大的Git版本库托管商，是成千上万的开发者和项目能够合作进行的中心。
- 大部分Git版本库都托管在GitHub，很多开源项目使用GitHub实现Git托管、问题追踪、代码审查等工作。
- 所以，尽管这不是Git开源项目的直接部分，但如果想要专业地使用Git，你将不可避免地与GitHub打交道。

Fork & Pull Request

- GitHub为来自世界各地的开发者一起工作提供了良好的沟通与合作机制。
- 假如你对别人GitHub帐户名下的一个开源项目很感兴趣，希望在其中做些贡献，你并不需要提前联系开发者，把你添加为Contributor。你只需要点击页面上的Fork（派生）按钮，这样你的名下就会有一套完全一样的工程。你对你自己名下的工程有全部的读写权限，可以任意修改提交；
- 修改完成后，通过Pull Request（合并请求），可以让原始开发者看到你的修改，同意合并、拒绝合并，或是在平台上与你讨论。如果他觉得有必要合并你的修改，所有的操作也非常简单。