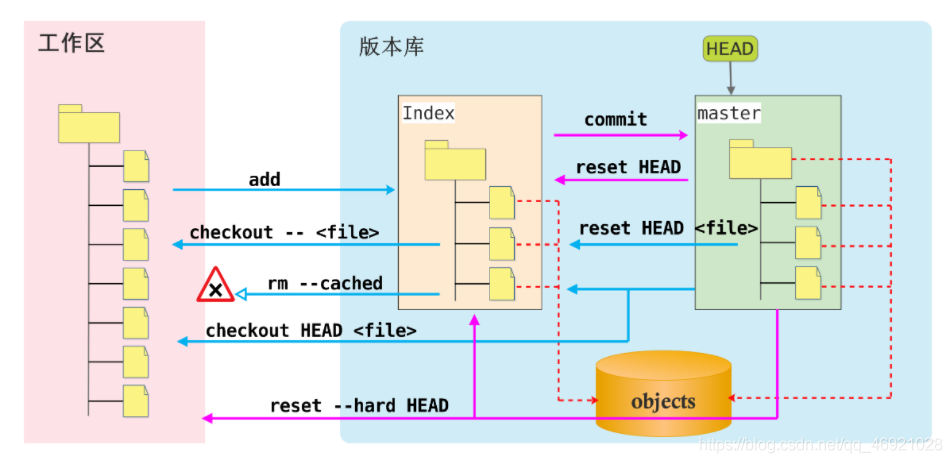
* 必看网站：https://learngitbranching.js.org/?locale=zh\_CN

# 主要内容



# merge和rebase

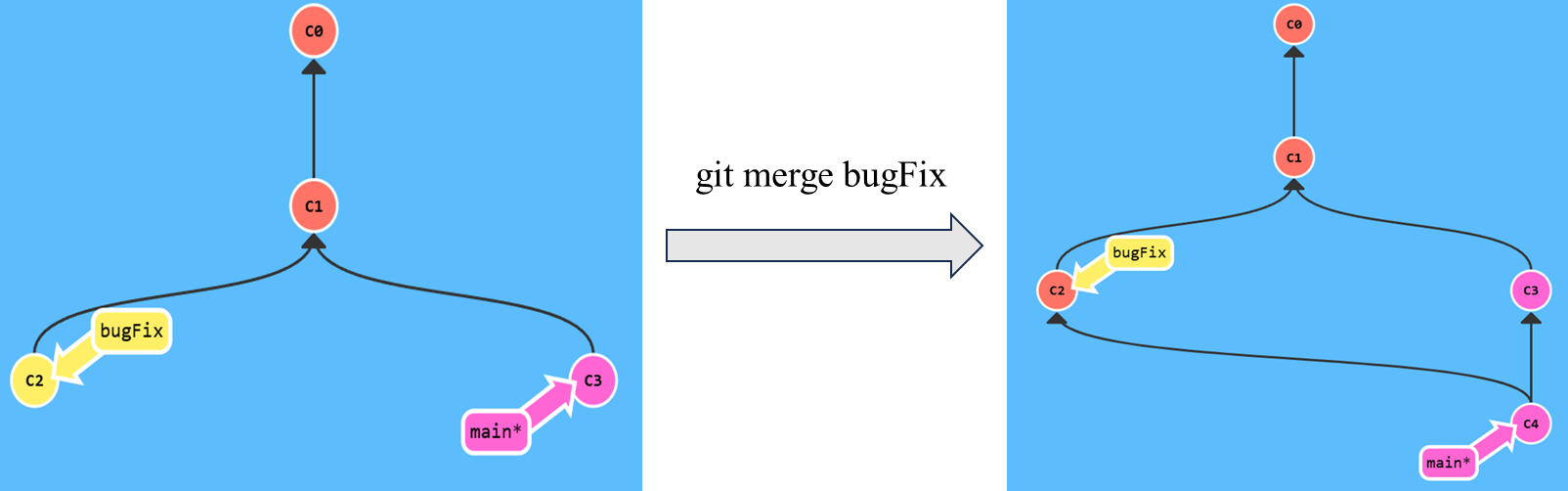
## merge

这个操作是会“保留”操作记录，即能到是如何操作的。

# 假设当前是在main分支上，合并其他分支（例如bugFix）

**git merge bugFix**

这将合并bugFix到main分支上 ，如图所示



## rebase

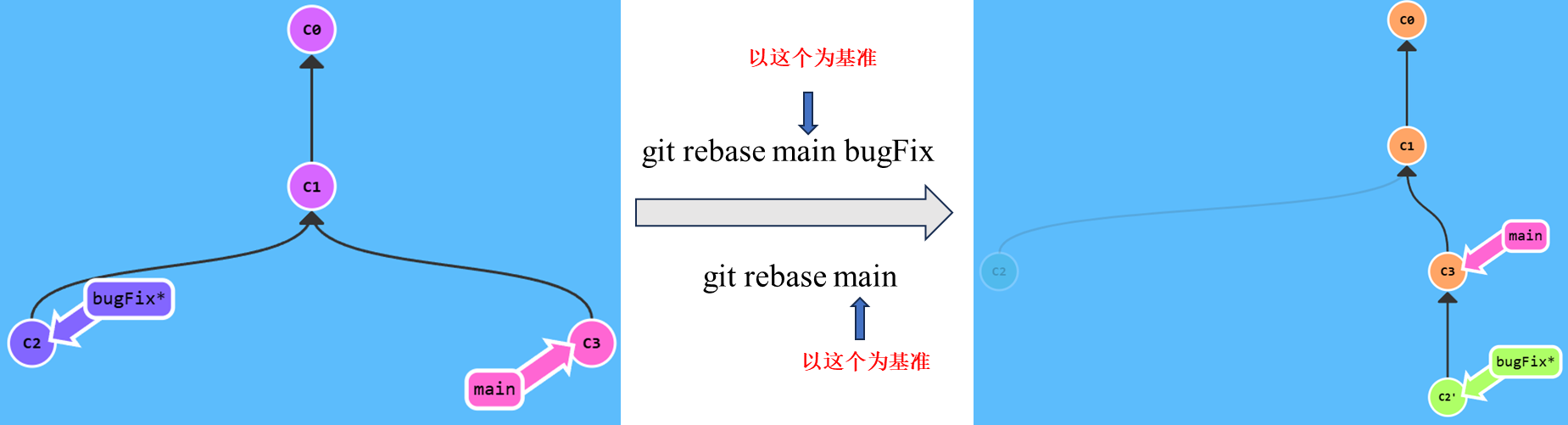
此操作舍弃了一些操作记录，让提交记录和一条直线一样。

# 这个是将以**branchA**为基准，然后将**branchB**分支上的操作变基到**branchA**

**git rebase branchA branchB**

**#** 假设当前是在bugFix上，可以使用rebase变基到main上，操作时记得换分支

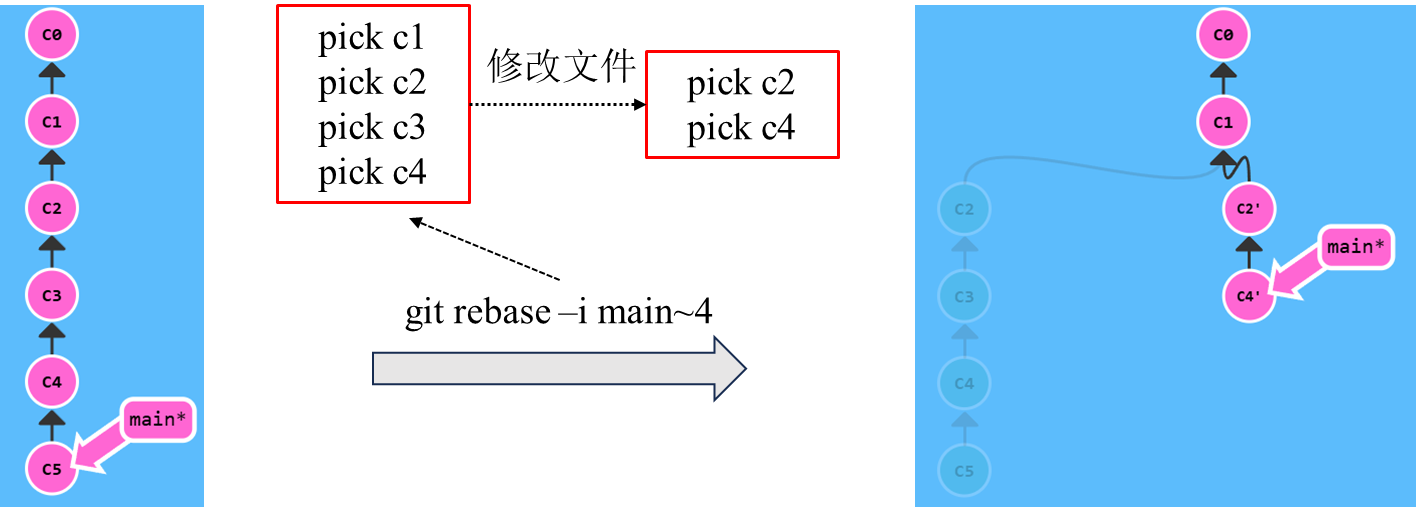
**git rebase main**



# git rebase -i 交互式变基

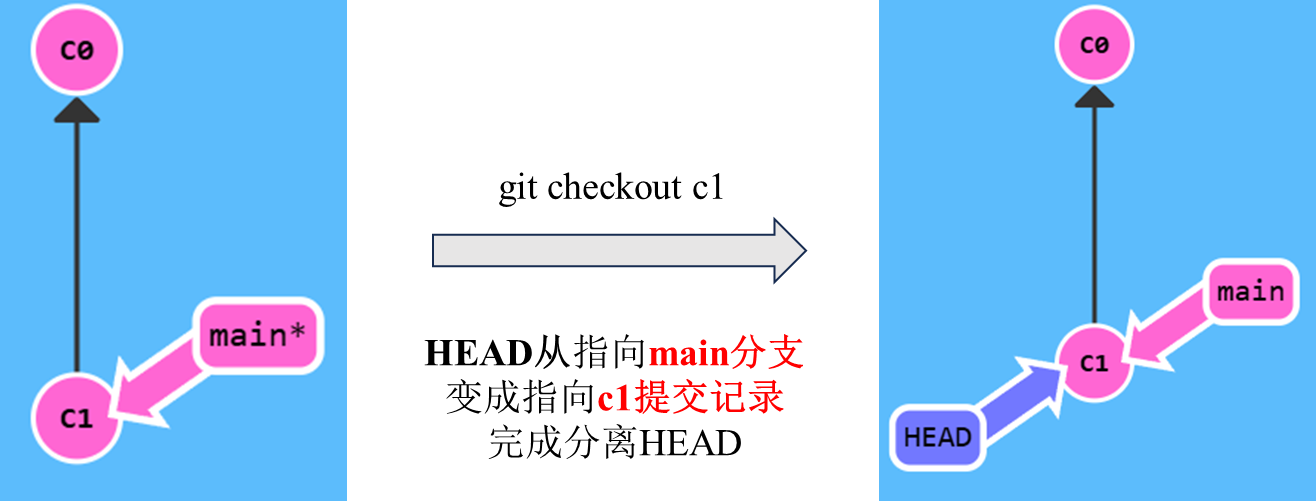
* git rebase -i <base\_commit>

以base\_commit为基准，将当前HEAD所指向的分支与其做交互式变基。



# 分离HEAD

当分支带\*的时候，表明**HEAD**指向当前分支，**需要区分提交记录与分支**。



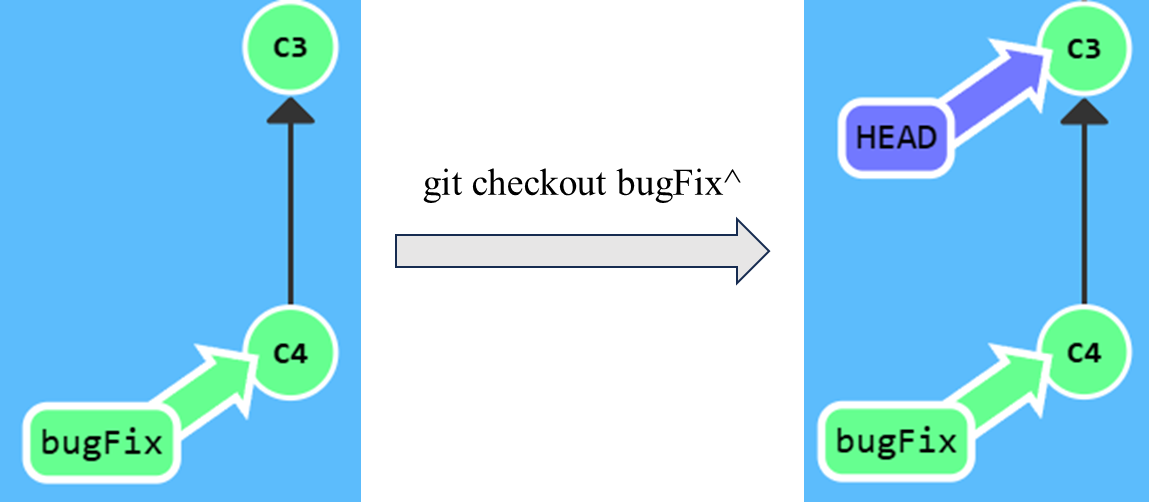
# 相对引用

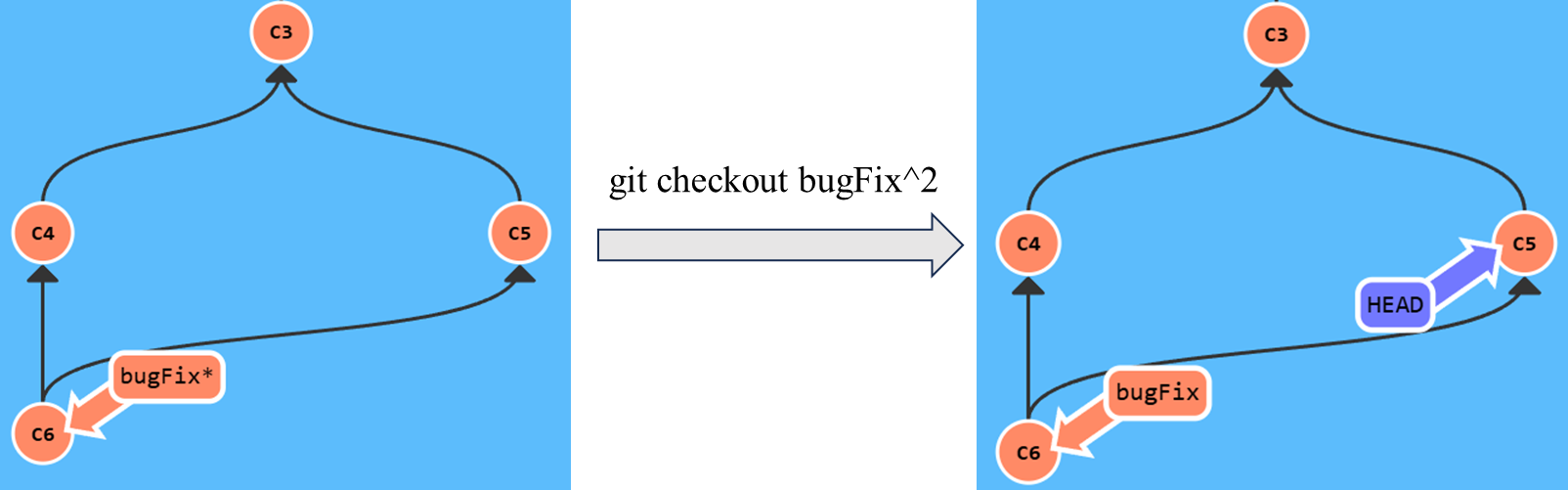
## **^**

表示指向父提交。

1.分支A后面加上“**^**”，表示基于分支A的父提交记录。

2.如果“**^**”后面加上数字x，则表示第x次的父提交。

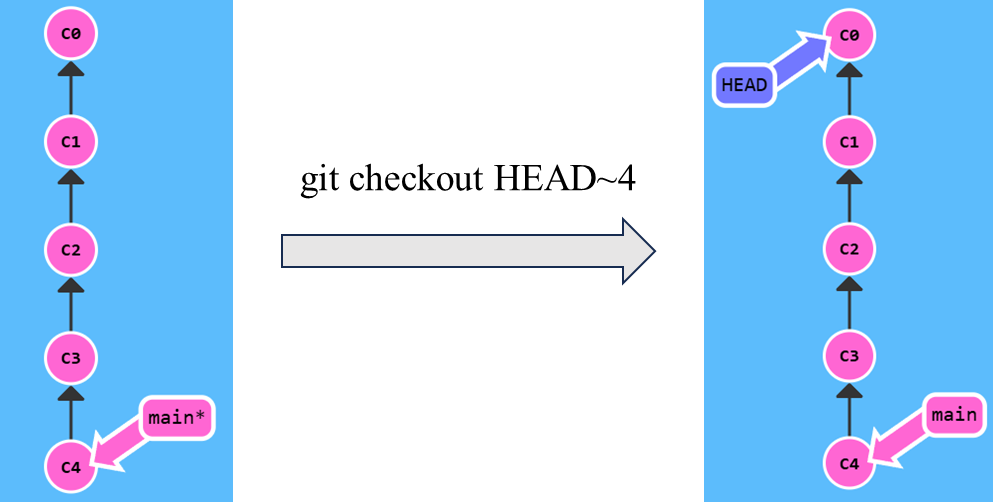




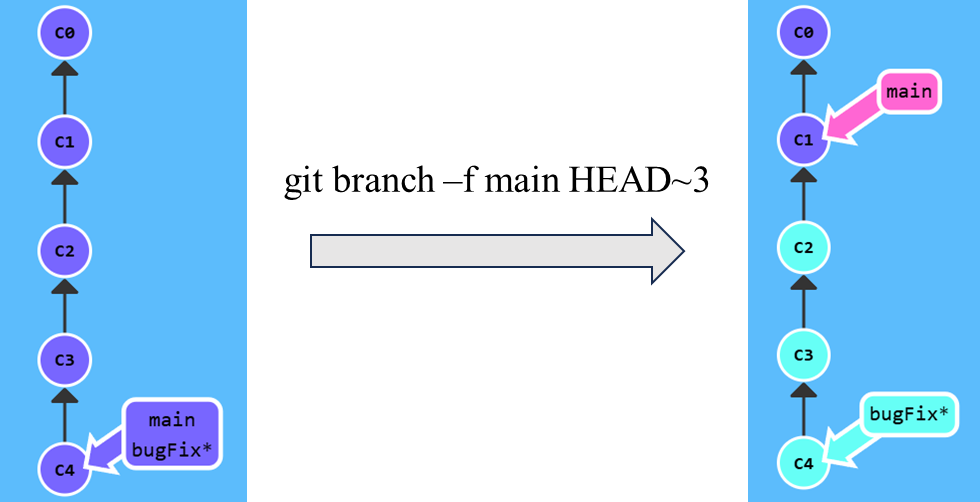
## **~**

1. branchA~x：

表示指向分支“branchA”的第前x次的父提交。



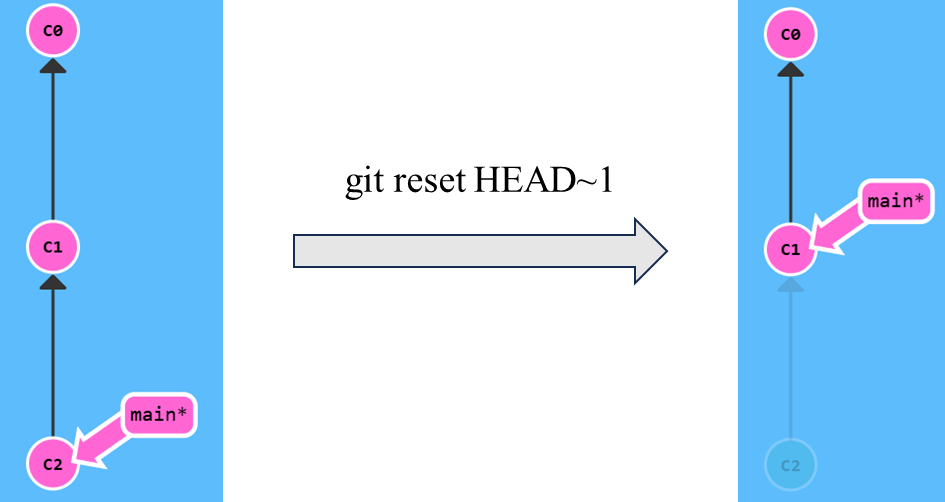
2.有时候会结合`**git branch -f originalBranch desertinationBranch~x`，**强制改变分支的指向。



# git reset和git revert

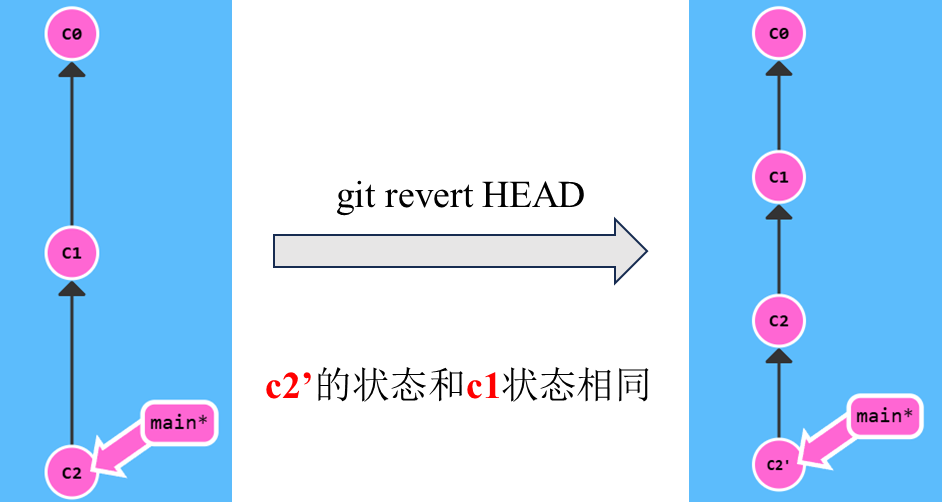
## git reset

可以“改写历史”，抹去过去的操作，通常是本地操作居多。



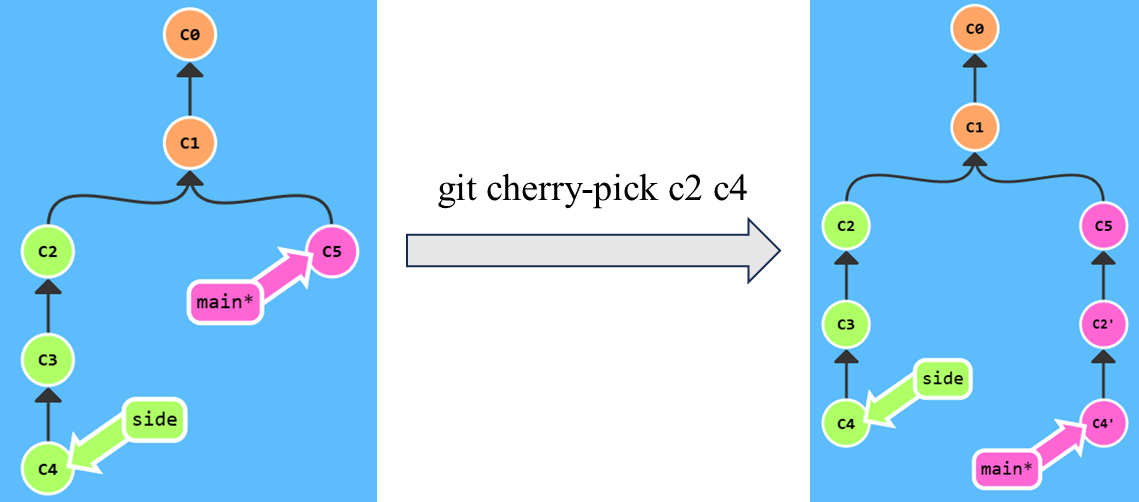
## git revert

可以显示 “撤销操作”，通常团队开发居多。



# git cherry-pick commit-x commit-y …

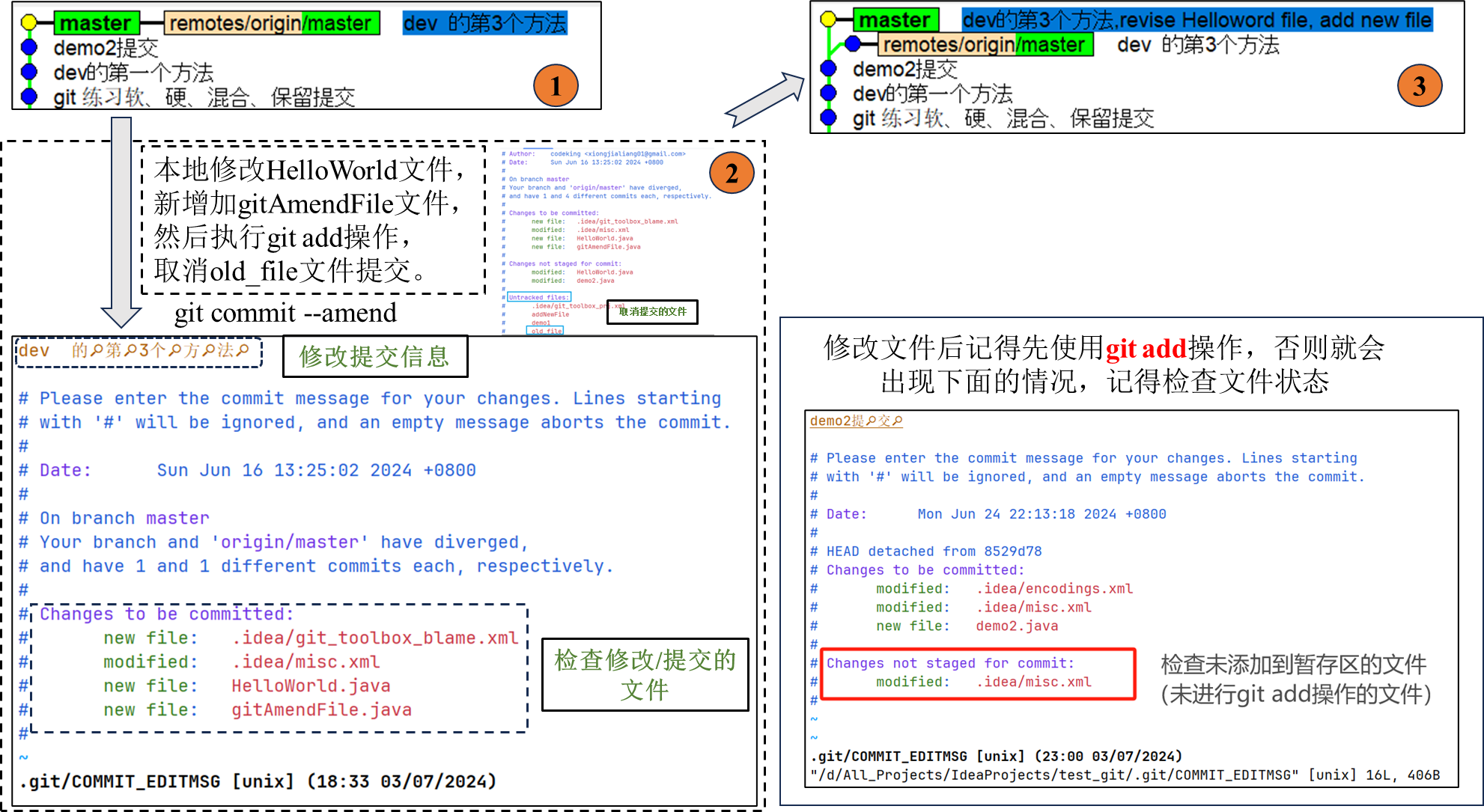
将commit-x commit-y …这些提交的内容复制到**当前HEAD所指向的分支X**（但是这些cherry-pick的提交**不能是分支X**的上游提交）。



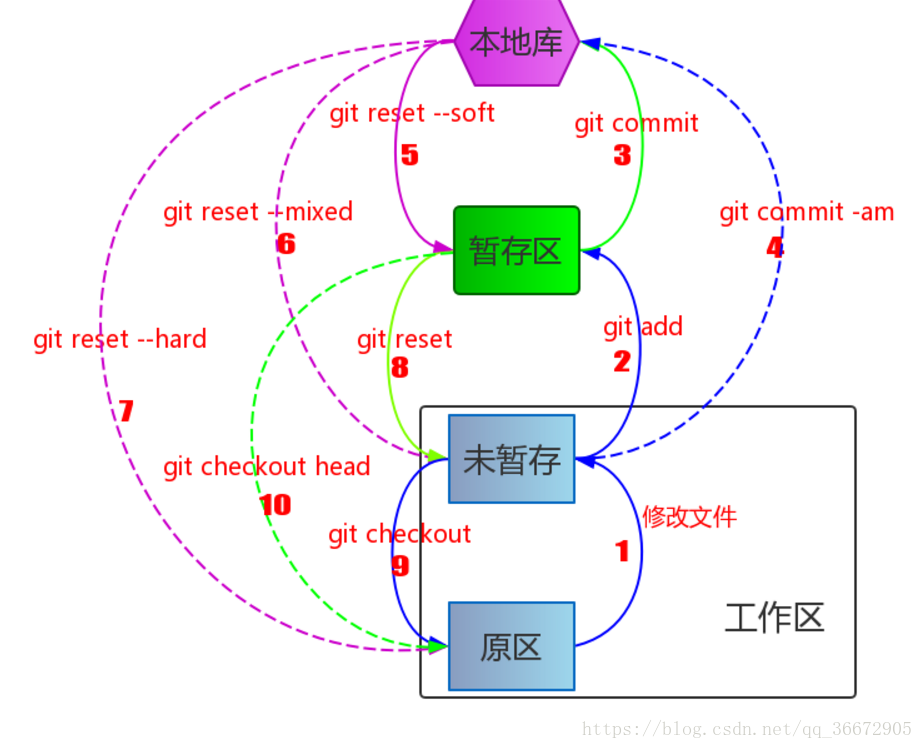
# git commit –amend

可实现的功能（**修改上一次的提交的内容**）：

1. 修改文件（需要结合`git add . `或者`git add 修改文件`）
2. 提交新文件（需要结合`git add . `或者`git add 新增文件`）
3. 取消旧文件提交（取消追踪文件`git rm -rf xx-file`）
4. 修改提交信息



# git reset 的四种模式



## git reset –soft

完全保留工作区和暂存区，会撤销git commit提交，保留git add操作的内容（所有reset都会执行这个操作）

## git reset --mixed

完全保留工作区，会撤销commit提交和git add操作（重置了暂存区，本地更改保留，但是不会被git追踪了）

## git reset --hard

彻底清除工作区和暂存区，**\*\*\*慎用\*\*\***

## git reset --keep

保留工作区和HEAD之间的差异

# 巧用git reset 撤销某一个文件的提交

## 通用方法

1. 取消文件追踪

`git rm -rf --cached xx-file`

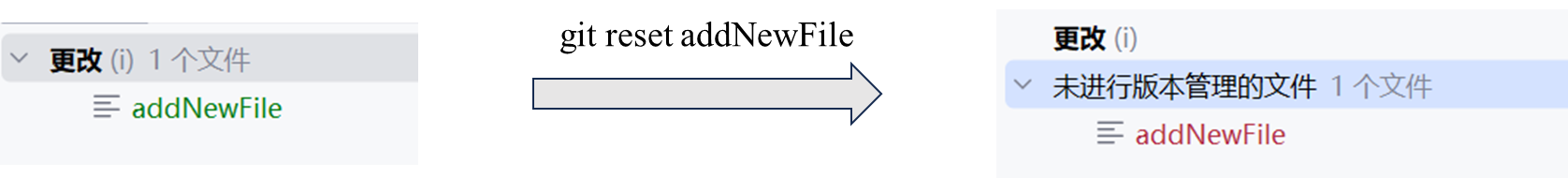
1. 修正提交

`git commit --amend`

**`:wq`**退出即可。

## 未提交

直接用`git reset 对应的文件`就行



## 已提交

1. 重置或者删除提交

重置` git reset --soft HEAD~1 `（推荐软重置，原来的提交到暂存区的内容都在）

删除提交`git reset --hard HEAD~1`

1. 然后用`git reset 对应的文件`
2. **暴力解决方式：**

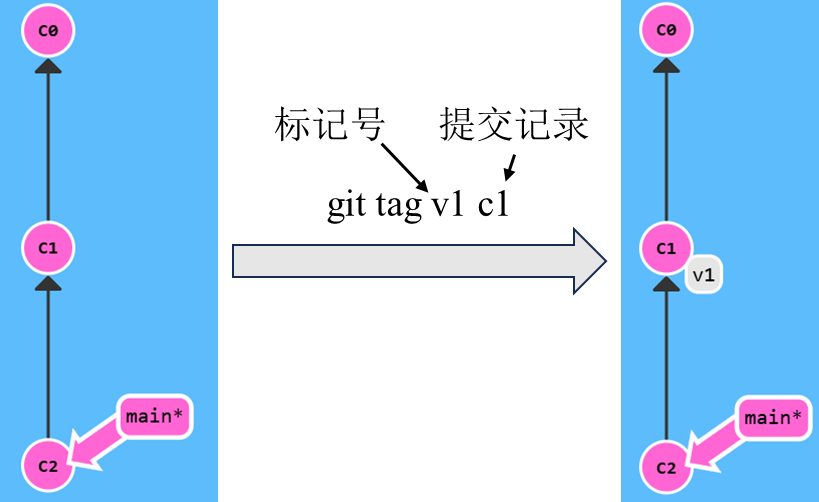
执行` git reset --mixed HEAD~1 `，然后选择文件重新提交。

# git tag

打标记的作用，记录特定版本。

`git tag Vxx（标记号） Cxx（提交记录）`

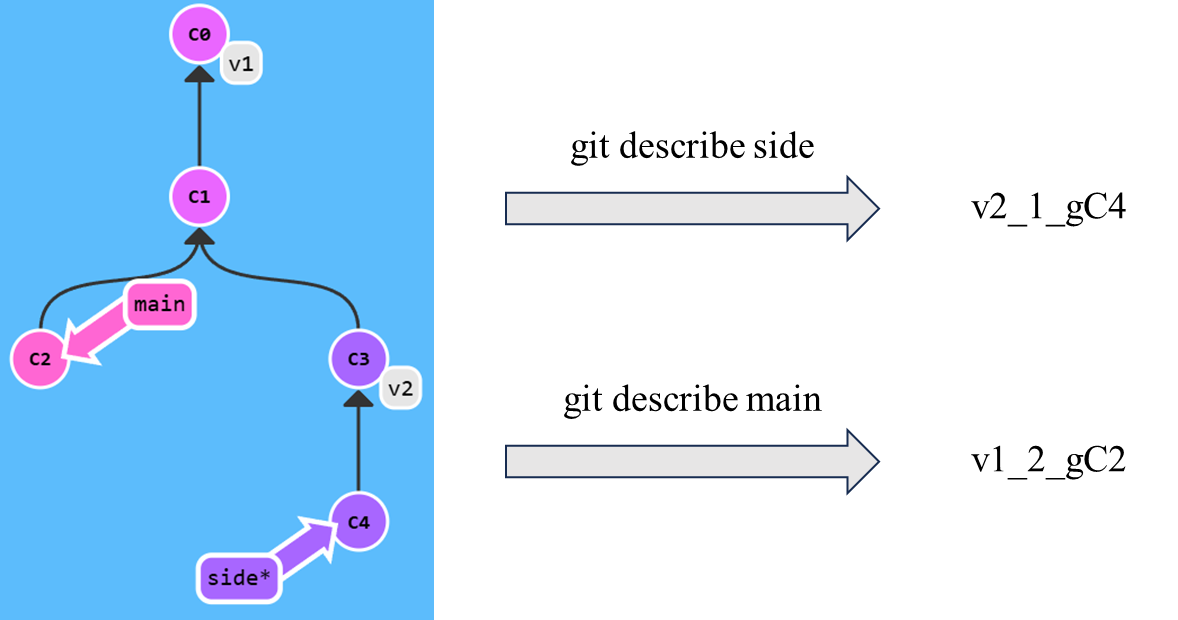
打了标记以后可以直接用`git checkout Vxx`更方便切换到指定的版本。



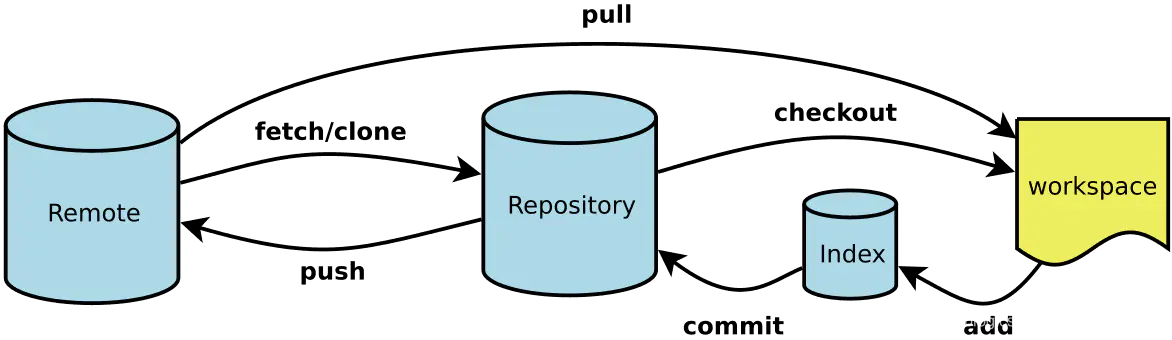
# git describe 分支（或者提交记录）

* **结果展示：<tag>\_<numCommits>\_g<hash>**

tag 表示的是离 ref 最近的标签， numCommits 是表示这个 ref 与 tag 相差有多少个提交记录， hash 表示的是给定的 ref 所表示的提交记录哈希值的前几位。

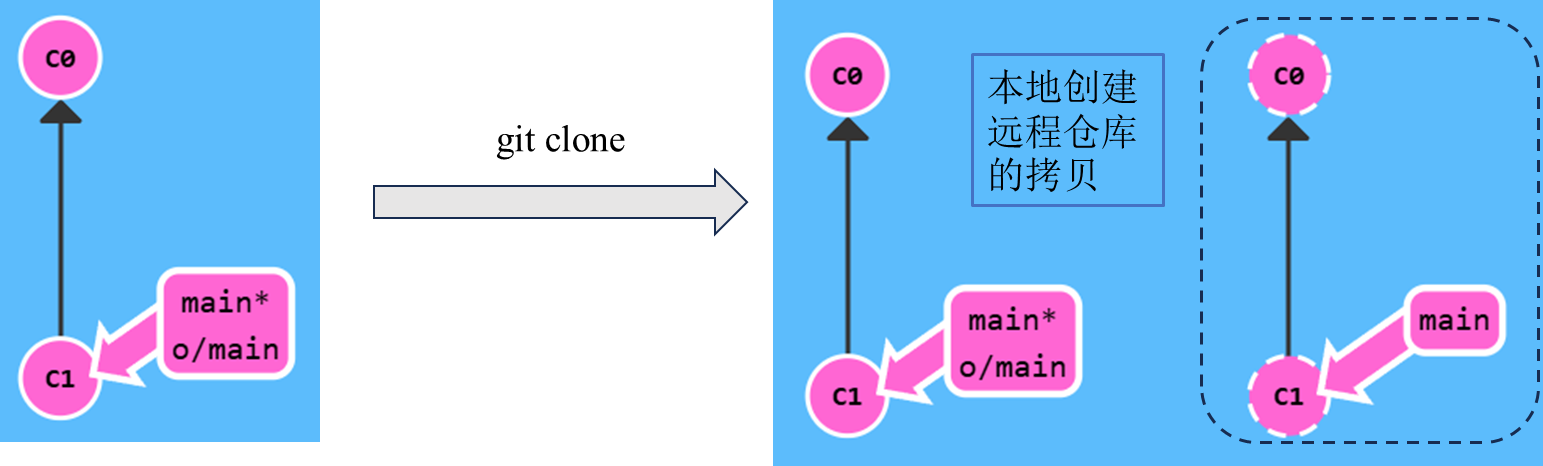


# 远程相关的内容



# git clone

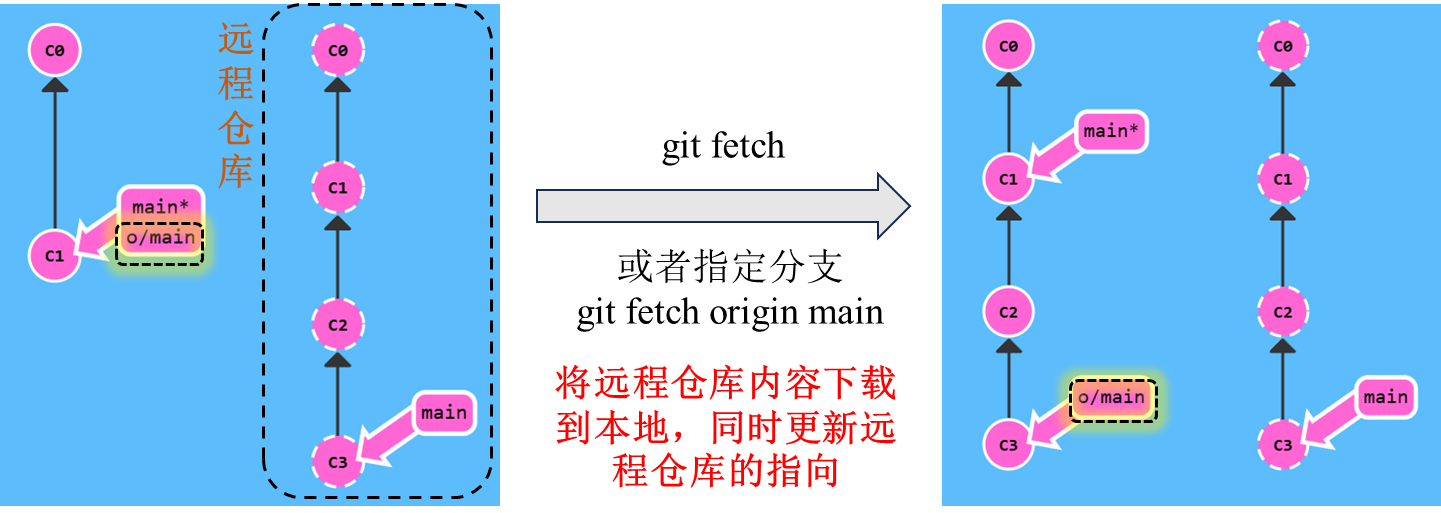
本地仓库会创建远程的拷贝，这个origin/main 是只有更新远程的时候才会改变，本地是无法操作这个的。



远程分支格式：<remote name>/<branch name>

# git fetch

1. 从远程仓库下载本地仓库中缺失的提交记录
2. 更新远程分支指针(如 o/main)



## git fetch

不跟参数，默认下载远程的所有分支。git fetch –all 也是下载所有分支。

## git fetch <remote> <place >

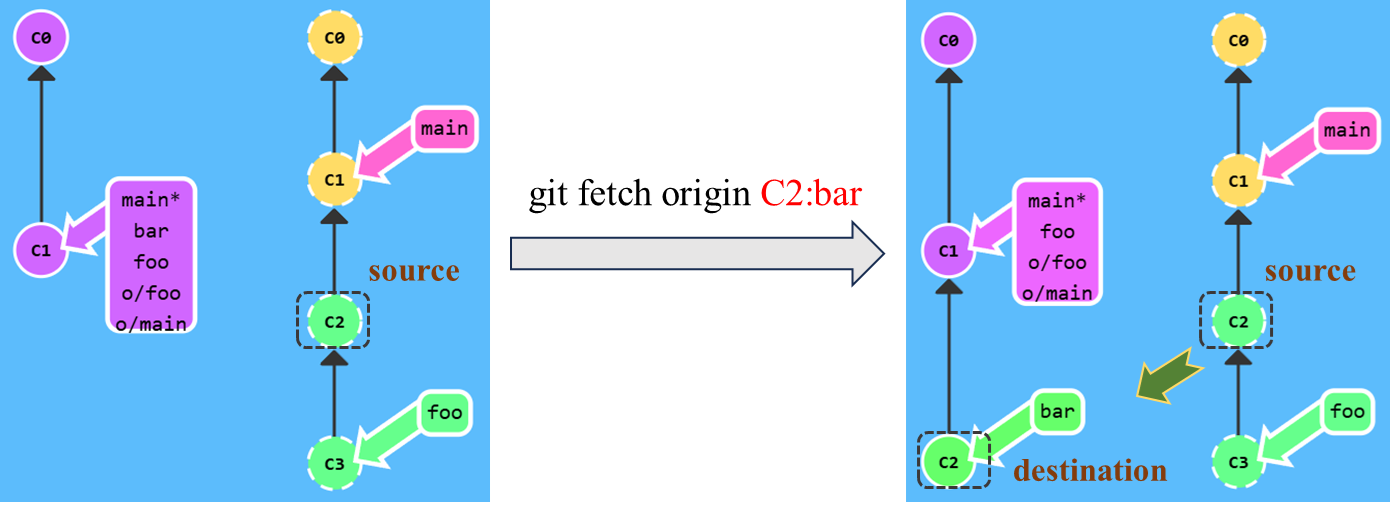
下载指定分支“place”。

**注意：**git fetch 并不会改变你本地仓库的状态。它**不会更新你的 main 分支**，也**不会修改本地磁盘的文件**。理解这一点很重要，因为许多开发人员误以为执行了 git fetch 以后，他们本地仓库就与远程仓库同步了。它可能已经将进行这一操作所需的所有数据都下载了下来，但是并没有修改你本地的文件。

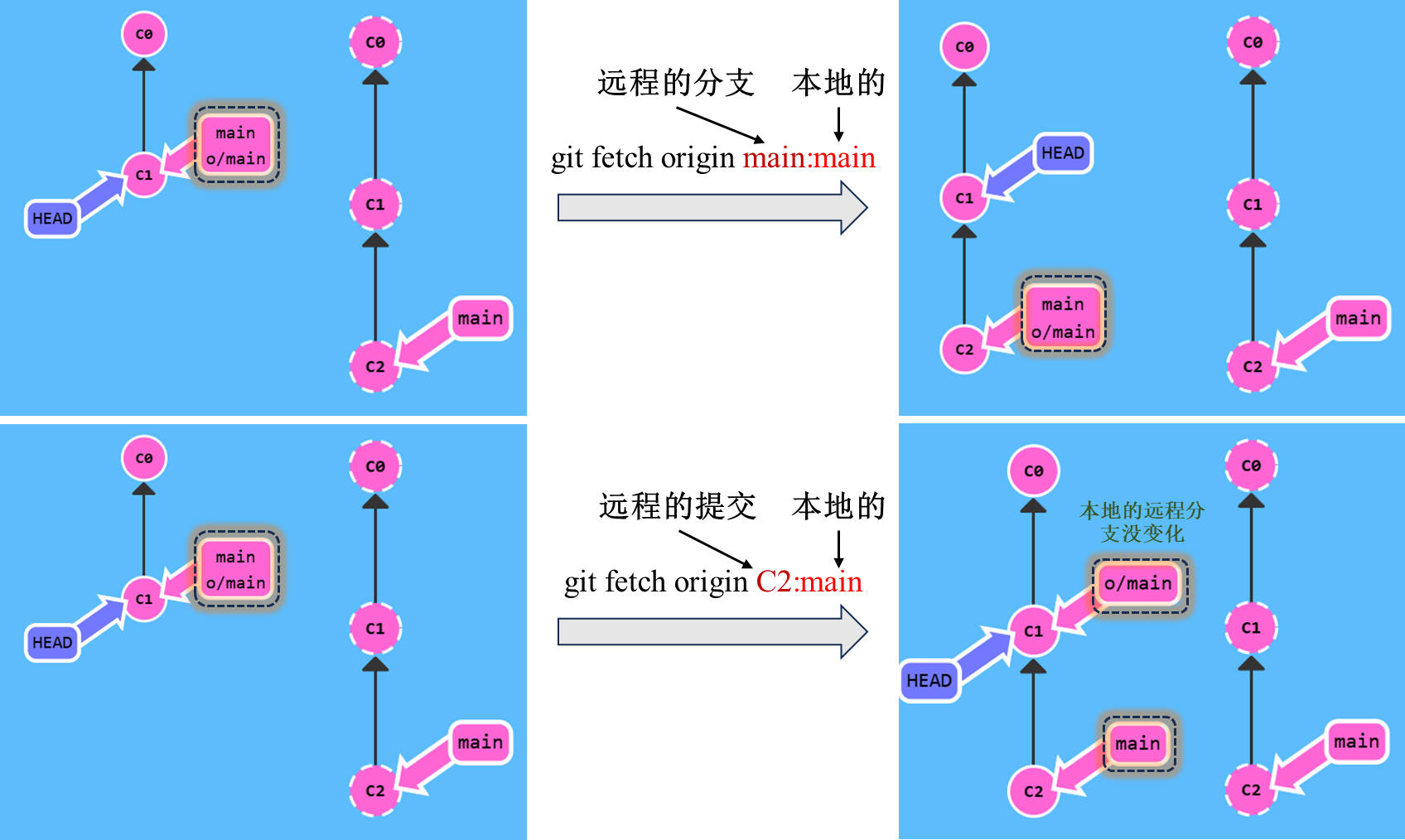
## git fetch <remote> <source>:<destination>

注意记得先分离HEAD!

source 现在指的是远程仓库中的位置，而 <destination> 才是要放置提交的本地仓库的位置，它与 git push 刚好相反。



如果不想改变本地origin/分支指向，source的内容要写提交记录而不是具体分支。



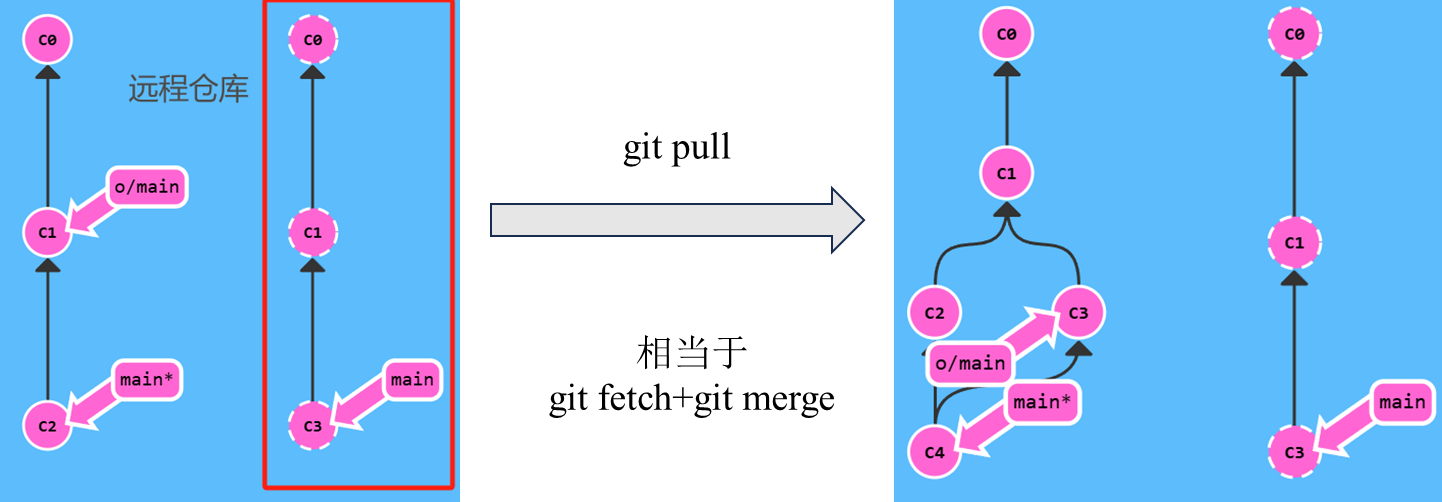
## git fetch <remote> :<fetchBranch>

本地新建“fetchBranch”分支，与远程分支的位置无关，就**新建分支的作用**。

# git pull

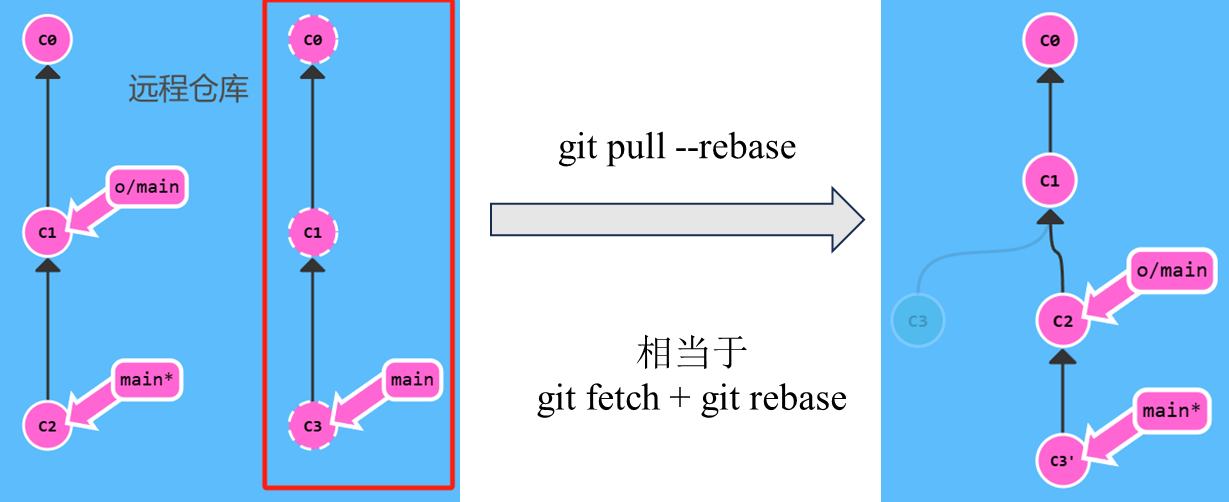
## git pull =git fetch + git merge

拉取远程分支（X），并将X分支merge到本地的分支（HEAD所指向的）上。



## git pull --rebase = git pull + git rebase

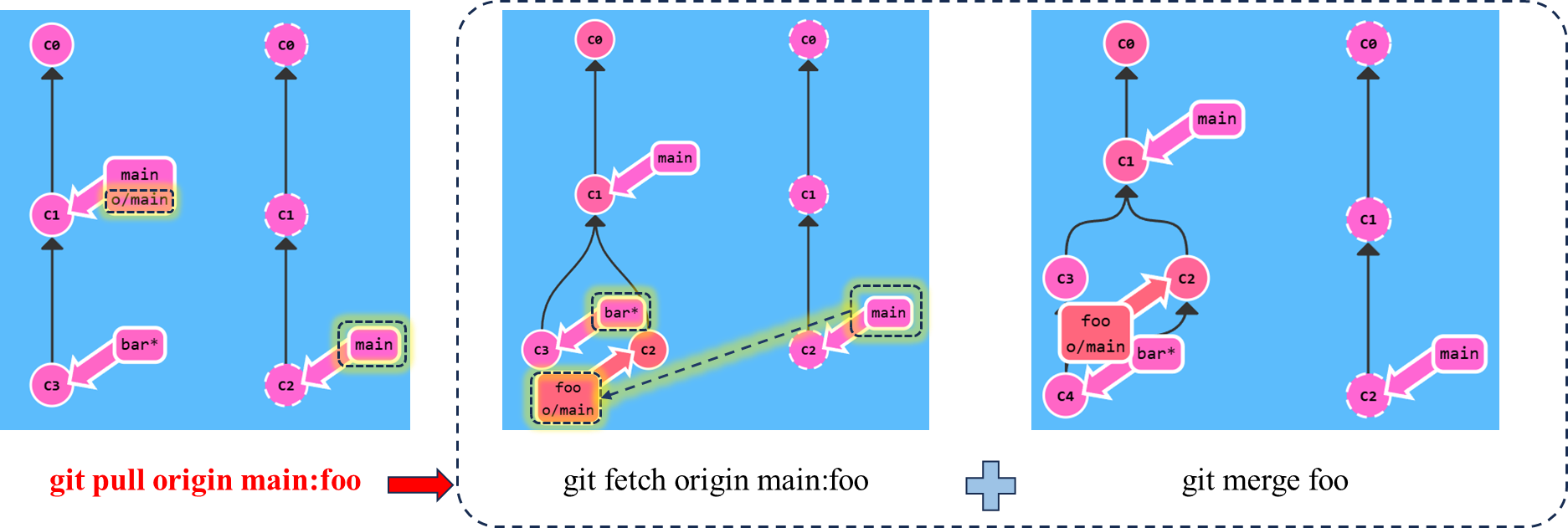
拉取远程的提交记录（X），并变基到X上。



## git pull <remote> <source>:<destination>

git pull <remote> <source>:<destination>

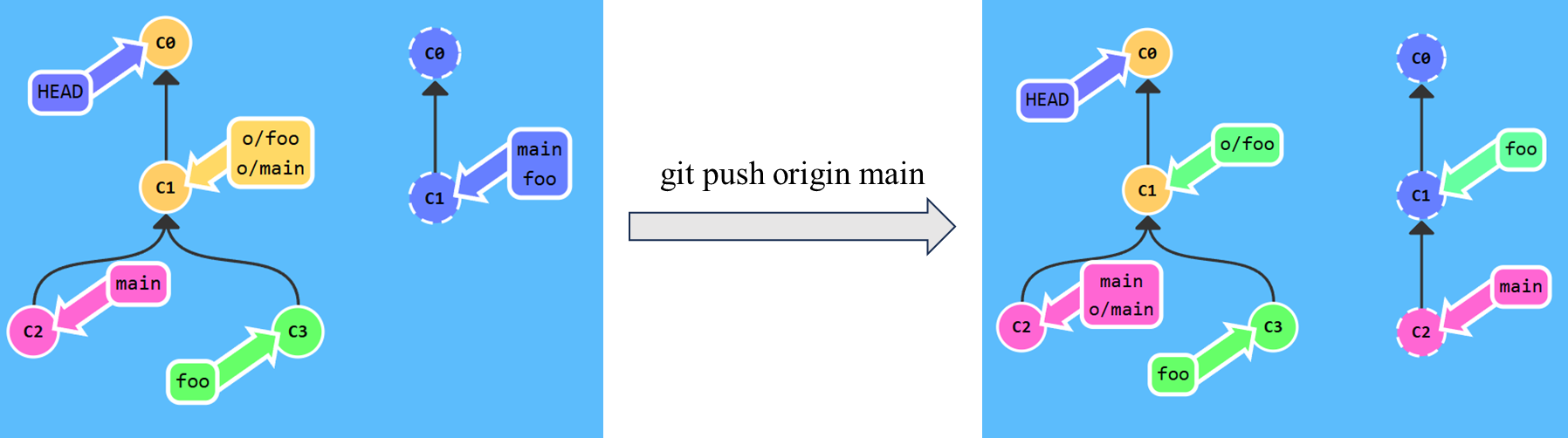
= git fetch <remote> <source>:<destination>; git merge destination

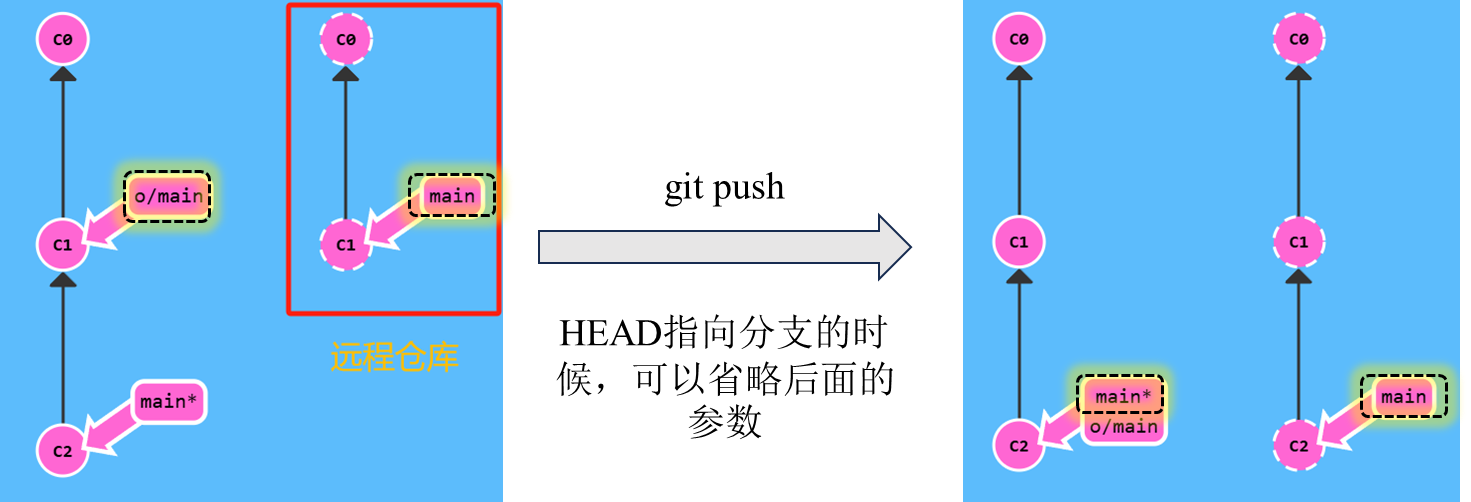


# git push

## git push <remote> <place>

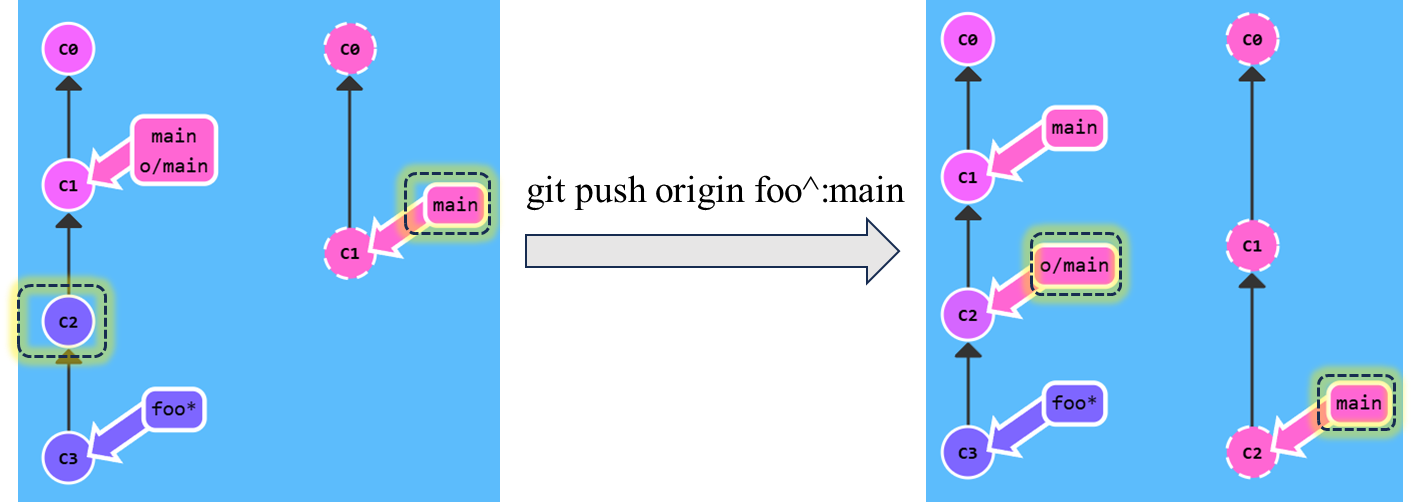
推送本地place分支到远程仓库remote的place分支上。





## git push <remote> <source>:<destination>

推送本地的 source的提交内容到远程的destination分支。



## git push <remote> :<delBranch>

删除远程分支“delBranch”

# 其他内容

# git stash

隐藏当前正在工作的时候做的记录修改，相当于快照。