

Git 各指令的本质，真是通俗易懂啊

良许Linux 2022-04-15 21:03



良许Linux

技术分享 | 资料共享 | 英语交流

后台回复【进群】，带你进入高手如云交流群



来源: juejin.cn/post/6895246702614806542

0 前言

作为当前世界上最强大的代码管理工具Git相信大家都很熟悉，但据我所知有很大一批人停留在clone、commit、pull、push...的阶段，是不是对rebase心里没底只敢用merge？

碰见版本回退就抓瞎？别问我怎么知道的，问就是：“我曾经就是这样啊~~”。

针对这些问题，今天我就将这几年对Git的认知和理解分享出来，尽可能的从本质去讲解Git，帮助你一步一步去了解Git的底层原理，相信读完本篇文章你便可以换种姿态，更加风骚得使用Git各种指令。

目录

- 1. 基本概念



- 1.1 Git的优势
- 1.2 文件状态
- 1.3 commit 节点
- 1.4 HEAD
- 1.5 远程仓库
- 2. 分支
 - 2.1 什么是分支?
- 3. 命令详解
 - 3.1 提交相关
 - 3.2 分支相关
 - 3.3 合并相关
 - 3.4 回退相关
 - 3.5 远程相关

1

基本概念

1.1 Git的优势

Git是一个分布式代码管理工具，在讨论分布式之前避免不了提及一下什么是中央式代码管理仓库

- 中央式：所有的代码保存在中央服务器，所以提交必须依赖网络，并且每次提交都会带入到中央仓库，如果是协同开发可能频繁触发代码合并，进而增加提交的成本和代价。最典型的就是svn
- 分布式：可以在本地提交，不需要依赖网络，并且会将每次提交自动备份到本地。每个开发者都可以把远程仓库clone一份到本地，并会把提交历史一并拿过来。代表就是Git

那Git相比于svn有什么优势呢？

打个比方："巴拉巴拉写了一大堆代码，突然发现写的有问题，我想回到一个小时之前"，对于这种情况Git的优势就很明显了，因为commit的成本比较小并且本地会保存所有的提交记录，随时随刻可以进行回退。

在这并不是说svn的不能完成这种操作，只是Git的回退会显得更加优雅。Git相比于中央式工具还有很多优点，就不一一列举了，感兴趣的可自行了解。

1.2 文件状态

在Git中文件大概分为三种状态：已修改（modified）、已暂存（staged）、已提交（committed）

- **修改**：Git可以感知到工作目录中哪些文件被修改了，然后把修改的文件加入到modified区域
- **暂存**：通过add命令将工作目录中修改的文件提交到暂存区，等候被commit

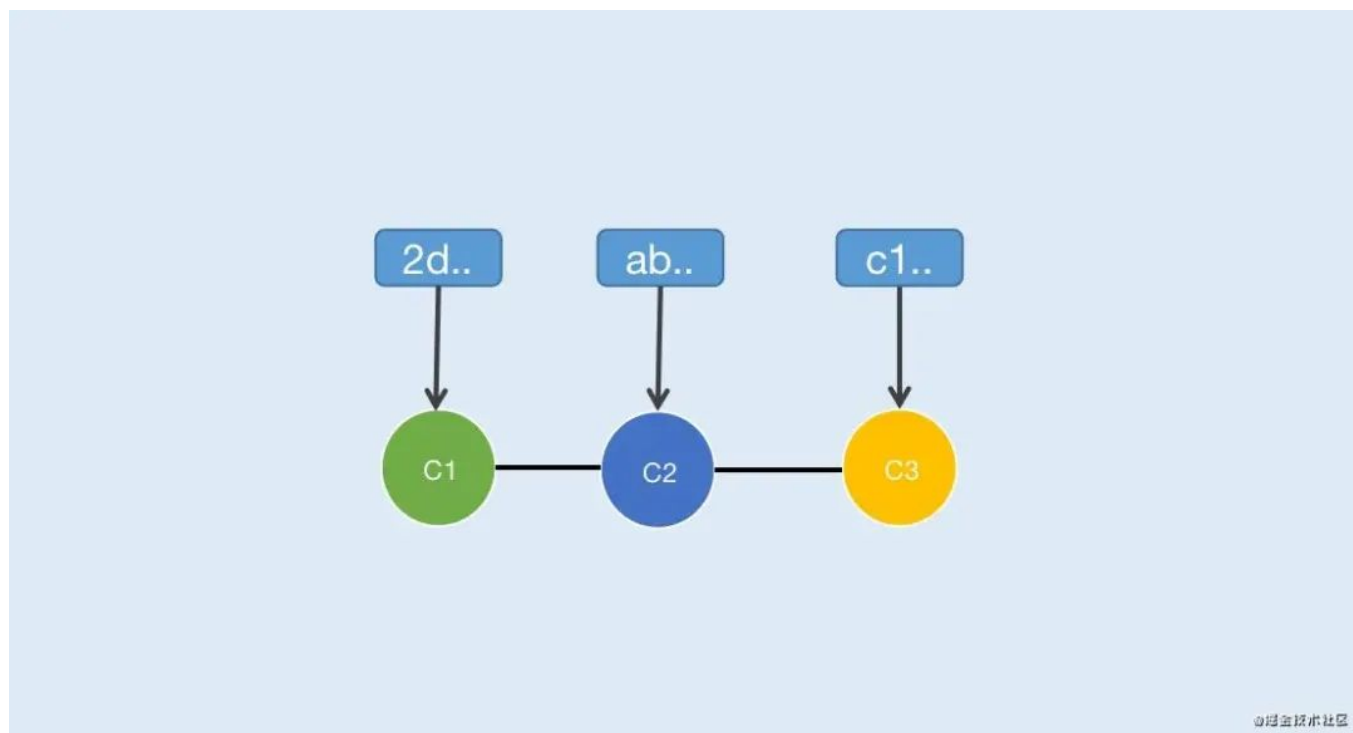
- **提交**：将暂存区文件commit至Git目录中永久保存

1.3 commit节点



为了便于表述，本篇文章我会通过节点代称commit提交

在Git中每次提交都会生成一个节点,而每个节点都会有一个哈希值作为唯一标示，多次提交会形成一个线性节点链（不考虑merge的情况），如图1-1



节点上方是通过 SHA1计算的哈希值

C2节点包含C1提交内容,同样C3节点包含C1、C2提交内容

1.4 HEAD

HEAD是Git中非常重要的一个概念，你可以称它为指针或者引用，它可以指向任意一个节点，并且指向的节点始终为当前工作目录，换句话说就是当前工作目录(也就是你所看到的代码)就是HEAD指向的节点。

还以图1-1举例，如果HEAD指向C2那工作目录对应的就是C2节点。具体如何移动HEAD指向后面会讲到，此处不要纠结。

同时HEAD也可以指向一个分支，间接指向分支所指向的节点。

1.5 远程仓库



虽然Git会把代码以及历史保存在本地，但最终还是要提交到服务器上的远程仓库。通过clone命令可以把远程仓库的代码下载到本地，同时也会将提交历史、分支、HEAD等状态一并同步到本地，但这些状态并不会实时更新，需要手动从远程仓库去拉取，至于何时拉、怎么拉后面章节会讲到。

通过远程仓库为中介，你可以和你的同事进行协同开发，开发完新功能后可以申请提交至远程仓库，同时也可以从远程仓库拉取你同事的代码。

注意点

因为你和你的同事都会以远程仓库的代码为基准，所以要时刻保证远程仓库的代码质量，切记不要将未经检验测试的代码提交至远程仓库

2 分支

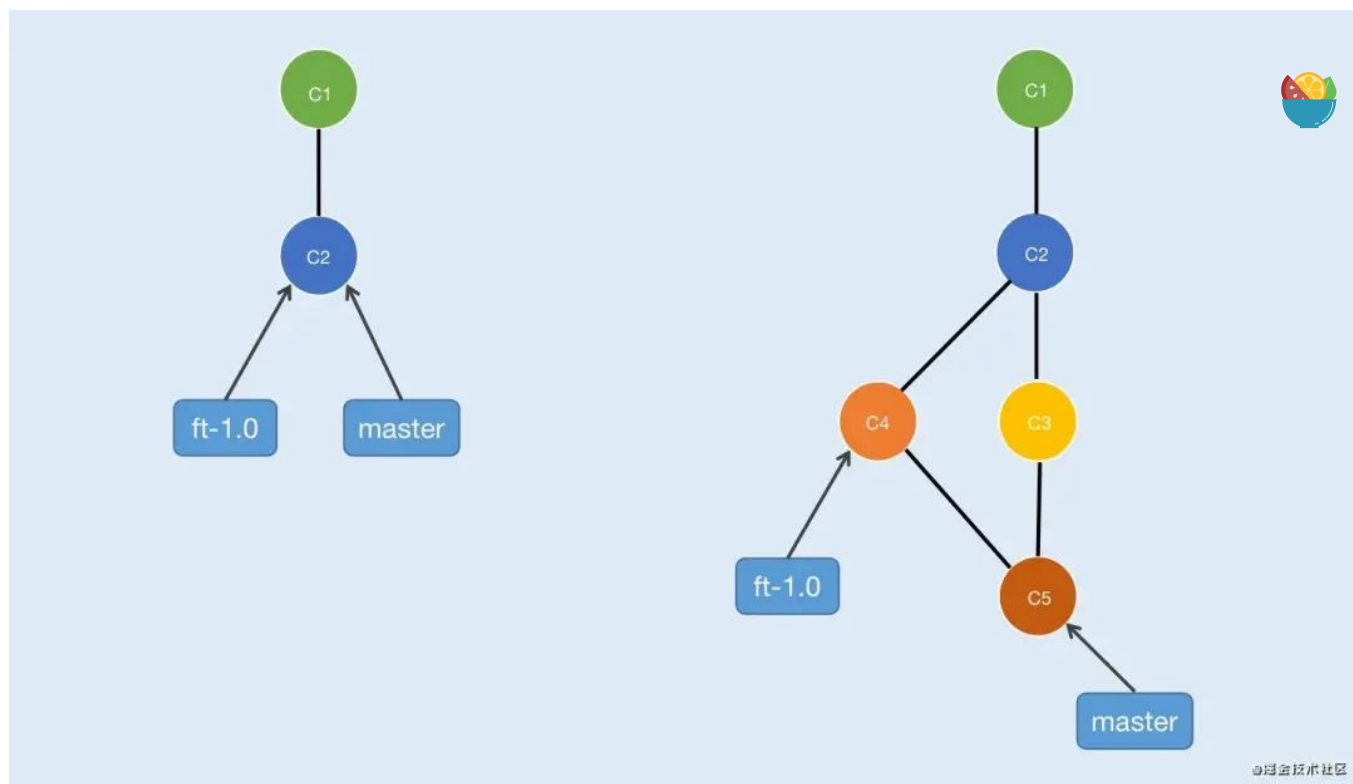
2.1 什么是分支？

分支也是Git中相当重要的一个概念，当一个分支指向一个节点时，当前节点的内容即是该分支的内容，它的概念和HEAD非常接近同样也可以视为指针或引用，不同的是分支可以存在多个，而HEAD只有一个。通常会根据功能或版本建立不同的分支。

那分支有什么用呢？

- 举个例子：你们的 App 经历了千辛万苦终于发布了v1.0版本，由于需求紧急v1.0上线之后便马不停蹄的开始v1.1，正当你开发的兴起时，QA同学说用户反馈了一些bug，需要修复然后重新发版，修复v1.0肯定要基于v1.0的代码，可是你已经开发了一部分v1.1了，此时怎么搞？

面对上面的问题通过引入分支概念便可优雅的解决，如图2-1



- 先看左边示意图，假设C2节点既是v1.0版本代码，上线后在C2的基础上新建一个分支ft-1.0
- 再看右边示意图，在v1.0上线后可在master分支开发v1.1内容，收到QA同学反馈后提交v1.1代码生成节点C3，随后切换到ft-1.0分支做bug修复，修复完成后提交代码生成节点C4，然后再切换到master分支并合并ft-1.0分支，到此我们就解决了上面提出的问题

除此之外利用分支还可以做很多事情，比如现在有一个需求不确定要不要上线，但是得先做，此时可以单独创建一个分支开发该功能，等到啥时候需要上线直接合并到主分支即可。分支适用的场景很多就不一一列举了。

注意点

当在某个节点创建一个分支后，并不会把该节点对应的代码复制一份出来，只是将新分支指向该节点，因此可以很大程度减少空间上的开销。一定要记着不管是HEAD还是分支它们都只是引用而已，量级非常轻

3 命令详解

3.1 提交相关

前面我们提到过，想要对代码进行提交必须得先加入到暂存区，Git中是通过命令 `add` 实现添加某个文件到暂存区：



```
git add 文件路径
```

添加所有文件到暂存区：

```
git add .
```

同时Git也提供了撤销工作区和暂存区命令

撤销工作区改动：

```
git checkout -- 文件名
```

清空暂存区：

```
git reset HEAD 文件名
```

提交：

将改动文件加入到暂存区后就可以进行提交了，提交后会生成一个新的提交节点，具体命令如下：

```
git commit -m "该节点的描述信息"
```

3.2 分支相关

创建分支

创建一个分支后该分支会与HEAD指向同一节点，说通俗点就是HEAD指向哪创建的新分支就指向哪，命令如下：

```
git branch 分支名
```

切换分支

当切换分支后，默认情况下HEAD会指向当前分支，即HEAD间接指向当前分支指向的节点

```
git checkout 分支名
```



同时也可以创建一个分支后立即切换，命令如下：

```
git checkout -b 分支名
```

删除分支

为了保证仓库分支的简洁，当某个分支完成了它的使命后应该被删除。比如前面所说的单独开一个分支完成某个功能，当这个功能被合并到主分支后应该将这个分支及时删除。

删除命令如下：

```
git branch -d 分支名
```

3.3 合并相关

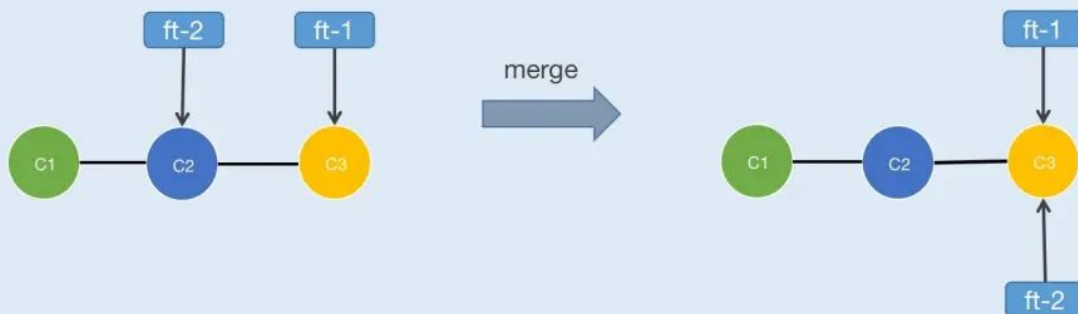
关于合并的命令是最难掌握同时也是最重要的。我们常用的合并命令大概有三个merge、rebase、cherry-pick

merge

merge是最常用的合并命令，它可以将某个分支或者某个节点的代码合并至当前分支。具体命令如下：

```
git merge 分支名/节点哈希值
```

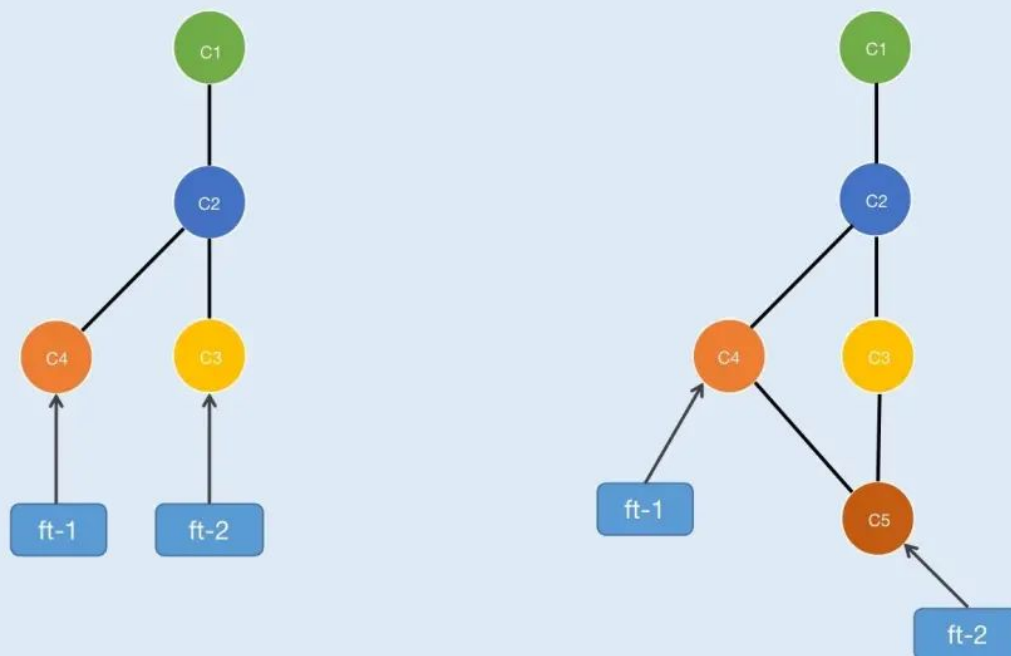
如果需要合并的分支完全领先于当前分支，如图3-1所示



©掘金技术社区

由于分支ft-1完全领先分支ft-2即ft-1完全包含ft-2，所以ft-2执行了“git merge ft-1”后会触发fast forward(快速合并)，此时两个分支指向同一节点，这是最理想的状态。

但是实际开发中我们往往碰到的是下面这种情况：如图3-2(左)



©掘金技术社区

这种情况就不能直接合了，当ft-2执行了“git merge ft-1”后Git会将节点C3、C4合并随后生成一个新节点C5，最后将ft-2指向C5 如图3-2(右)



注意点：

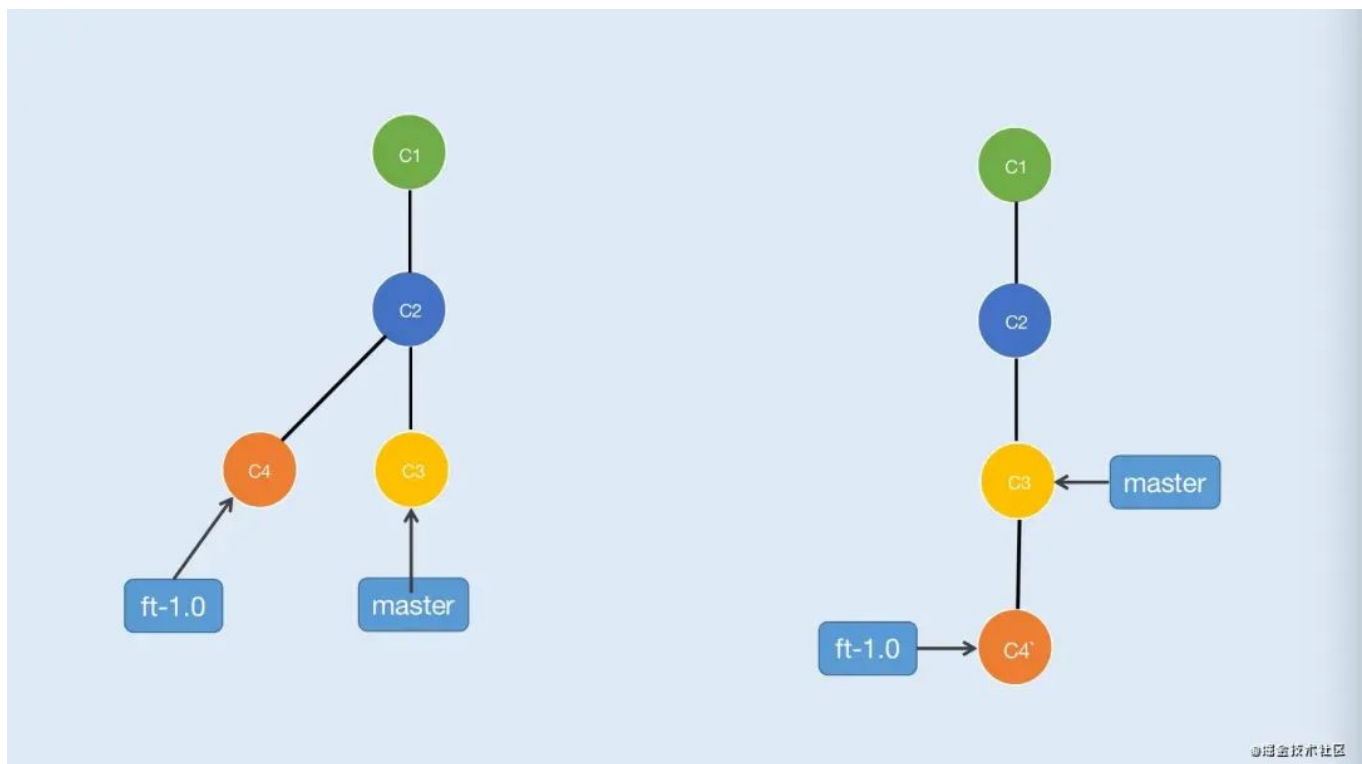
如果C3、C4同时修改了同一个文件中的同一句代码，这个时候合并会出错，因为Git不知道该以哪个节点为标准，所以这个时候需要我们自己手动合并代码

rebase


rebase也是一种合并指令，命令行如下：

```
git rebase 分支名/节点哈希值
```

与merge不同的是rebase合并看起来不会产生新的节点(实际上是会产生的，只是做了一次复制)，而是将需要合并的节点直接累加 如图3-3



当左边示意图的ft-1.0执行了git rebase master后会将C4节点复制一份到C3后面，也就是C4'，C4与C4'相对应，但是哈希值却不一样。

rebase相比于merge提交历史更加线性、干净，使并行的开发流程看起来像串行，更符合我们的直觉。既然rebase这么好用是不是可以抛弃merge了？其实也不是了，下面我罗列一些merge和rebase的优缺点 

merge优缺点：

- 优点：每个节点都是严格按照时间排列。当合并发生冲突时，只需要解决两个分支所指向的节点的冲突即可
- 缺点：合并两个分支时大概率会生成新的节点并分叉，久而久之提交历史会变成一团乱麻

rebase优缺点：

- 优点：会使提交历史看起来更加线性、干净
- 缺点：虽然提交看起来像是线性的，但并不是真正的按时间排序，比如图3-3中，不管C4早于或者晚于C3提交它最终都会放在C3后面。并且当合并发生冲突时，理论上来讲有几个节点rebase到目标分支就可能处理几次冲突

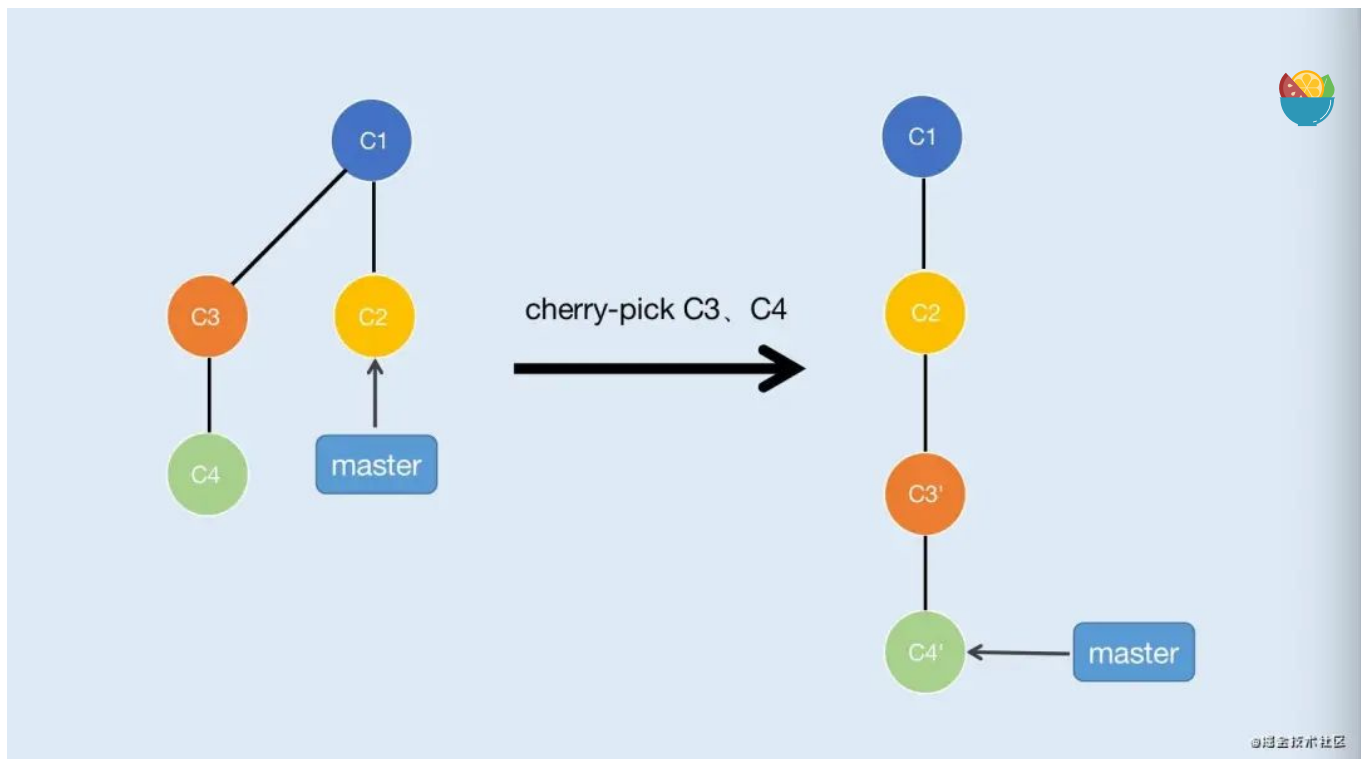
对于网络上一些只用rebase的观点，作者表示不太认同，如果不同分支的合并使用rebase可能需要重复解决冲突，这样就得不偿失了。但如果是本地推到远程并对应的是同一条分支可以优先考虑rebase。所以我的观点是 根据不同场景合理搭配使用merge和rebase，如果觉得都行那优先使用rebase

cherry-pick

cherry-pick的合并不同于merge和rebase，它可以选择某几个节点进行合并，如图3-4

命令行：

```
git cherry-pick 节点哈希值
```



假设当前分支是master，执行了git cherry-pick C3(哈希值)，C4(哈希值)命令后会直接将C3、C4节点抓过来放在后面，对应C3'和C4'

3.4 回退相关

分离HEAD


在默认情况下HEAD是指向分支的，但也可以将HEAD从分支上取下来直接指向某个节点，此过程就是分离HEAD，具体命令如下：

```
git checkout 节点哈希值
//也可以直接脱离分支指向当前节点
git checkout --detach
```

由于哈希值是一串很长很长的乱码，在实际操作中使用哈希值分离HEAD很麻烦，所以Git也提供了HEAD基于某一特殊位置(分支/HEAD)直接指向前一个或前N个节点的命令，也即相对引用，如下：

```
//HEAD分离并指向前一个节点
git checkout 分支名/HEAD^
```

```
//HEAD分离并指向前N个节点
git checkout 分支名~N
```

将HEAD分离出来指向节点有什么用呢？举个例子：如果开发过程发现之前的提交有问题，此时可以将HEAD指向对应的节点，修改完毕后再提交，此时你肯定不希望再生成一个新的节点，而你只需在提交时加 amend即可，具体命令如下：

```
git commit --amend
```

回退

回退场景在平时开发中还是比较常见的，比如你巴拉巴拉写了一大堆代码然后提交，后面发现写的有问题，于是你想将代码回到前一个提交，这种场景可以通过reset解决，具体命令如下：

```
//回退N个提交  
git reset HEAD~N
```

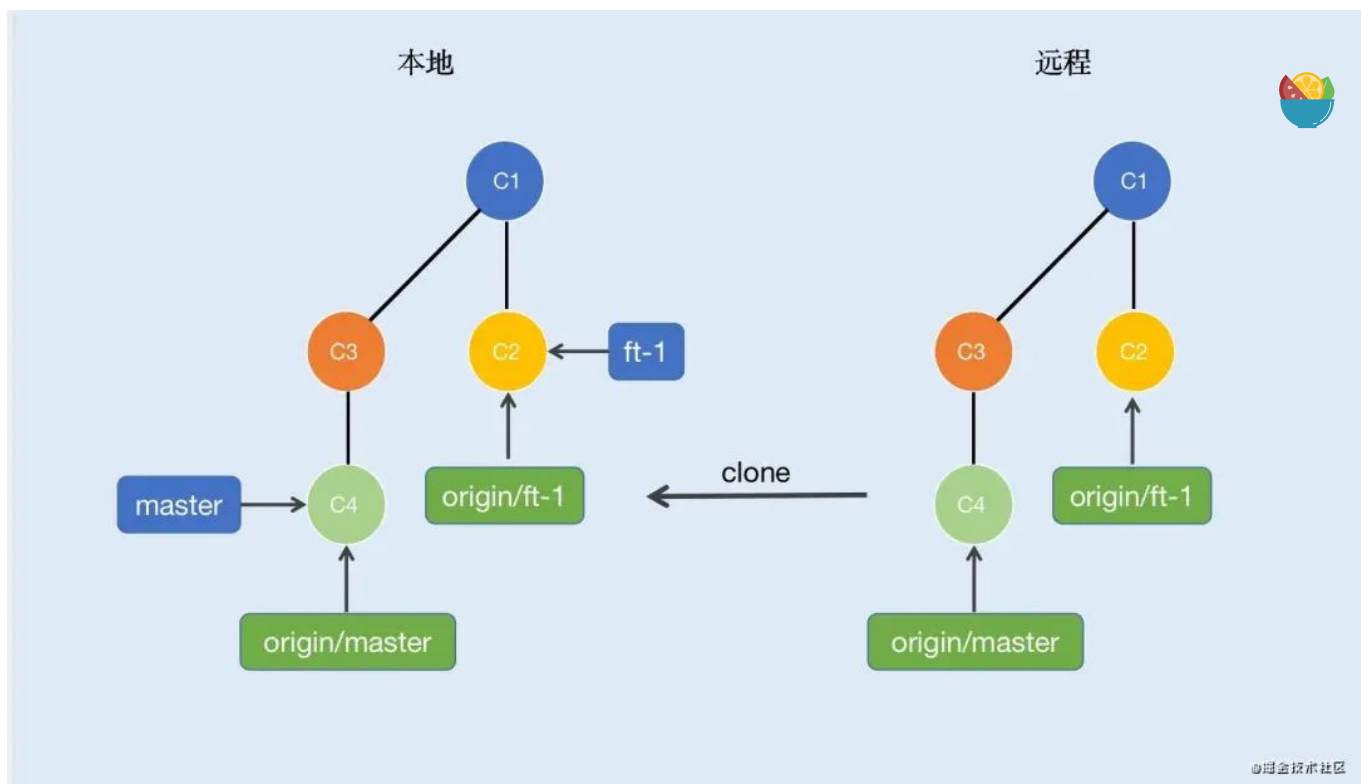
reset和相对引用很像，区别是reset会使分支和HEAD一并回退。

3.5 远程相关

当我们接触一个新项目时，第一件事情肯定是要把它的代码拿下来，在Git中可以通过clone从远程仓库复制一份代码到本地，具体命令如下：

```
git clone 仓库地址
```

前面的章节我也有提到过，clone不仅仅是复制代码，它还会把远程仓库的引用(分支/HEAD)一并取下保存在本地，如图3-5所示：



其中origin/master和origin/ft-1为远程仓库的分支，而远程的这些引用状态是不会实时更新到本地的，比如远程仓库origin/master分支增加了一次提交，此时本地是感知不到的，所以本地的origin/master分支依旧指向C4节点。我们可以通过fetch命令来手动更新远程仓库状态

小提示：

并不是存在服务器上的才能称作是远程仓库，你也可以clone本地仓库作为远程，当然实际开发中我们不可能把本地仓库当作公有仓库，说这个只是单纯的帮助你更清晰的理解分布式

fetch

说的通俗一点，fetch命令就是一次下载操作，它会将远程新增加的节点以及引用(分支/HEAD)的状态下载到本地，具体命令如下：

```
git fetch 远程仓库地址/分支名
```

pull

pull命令可以从远程仓库的某个引用拉取代码，具体命令如下：

```
git pull 远程分支名
```

其实pull的本质就是fetch+merge，首先更新远程仓库所有状态到本地，随后再进行合并。合并完成后本地分支会指向最新节点

另外pull命令也可以通过rebase进行合并，具体命令如下：

```
git pull --rebase 远程分支名
```

push

push命令可以将本地提交推送至远程，具体命令如下：

```
git push 远程分支名
```

如果直接push可能会失败，因为可能存在冲突，所以在push之前往往会先pull一下，如果存在冲突本地解决。push成功后本地的远程分支引用会更新，与本地分支指向同一节点。

综上所述

- 不管是HEAD还是分支，它们都只是引用而已，引用+节点是 Git 构成分布式的关键
- merge相比于rebase有更明确的时间历史，而rebase会使提交更加线性应当优先使用
- 通过移动HEAD可以查看每个提交对应的代码
- clone或fetch都会将远程仓库的所有提交、引用保存在本地一份
- pull的本质其实就是fetch+merge，也可以加入--rebase通过rebase方式合并

本公众号全部博文已整理成一个目录，请在公众号里回复「**m**」获取！

推荐阅读：

24 个常见的 Docker 疑难杂症处理技巧

在央企当程序员是一种怎样的体验？

Linux 进程编程入门

5T技术资源大放送！包括但不限于：C/C++，Linux，Python，Java，PHP，人工智能，单片机，树莓派，等等。在公众号内回复「1024」，即可免费获取

