<https://mp.weixin.qq.com/s/xXDTOBWJHBHPFs7-3oYEUg>

# 鹅厂是如何使用 Git 的

## 1. 前言

开发人员在日常开发过程中，不可避免地会使用到代码的版本控制工具，如 svn、git 等等，记得在刚刚入职的时候，部门使用的主要的 VCS 工具还是 svn，期间有着非常痛苦的 download 经历，下载一份仓库花了我 2-3 个小时，相比于 svn，git 有着非常多的优势，比如仓库 clone 速度非常快、核心的分支功能等等，后续公司也在推使用 git 来维护代码仓库，完全摒弃笨重的 svn。

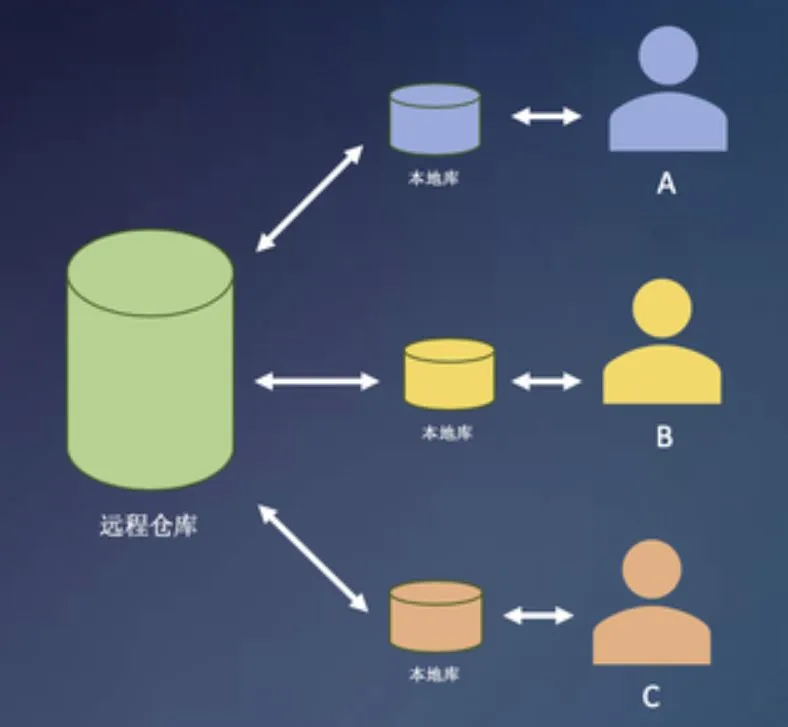
那么，切换到 git 来维护代码仓库，会对我们的日常开发造成影响吗？答应是显然的，首先我们需要学习 git 的基本概念与用法，然后就需要我们在具体的项目实践过程中打磨我们的 git 使用技巧，比如灵活的分支、子模块使用等等，关于 git 概念或技术上的介绍，本文不予展开，如果对 git 实现上的细节感兴趣的话，可以自行搜索学习。

接下来主要跟大家探讨的主题是 git 工作流，git 初学者可能对这个概念并不是很清晰，脑海中想到的可能是 git 的工作原理之类的，其实并不是的，git 工作流指的是多人协作过程中的 git 的使用流程，不涉及技术细节，是一种项目管理、开发约定的方式。有些同学可能觉得习得了 git 三板斧（clone、add commit、push）就算是完成了对 git 的开发认知，其实咱们可能还停留在最原始的想象之中。

## 2. 集中式工作流

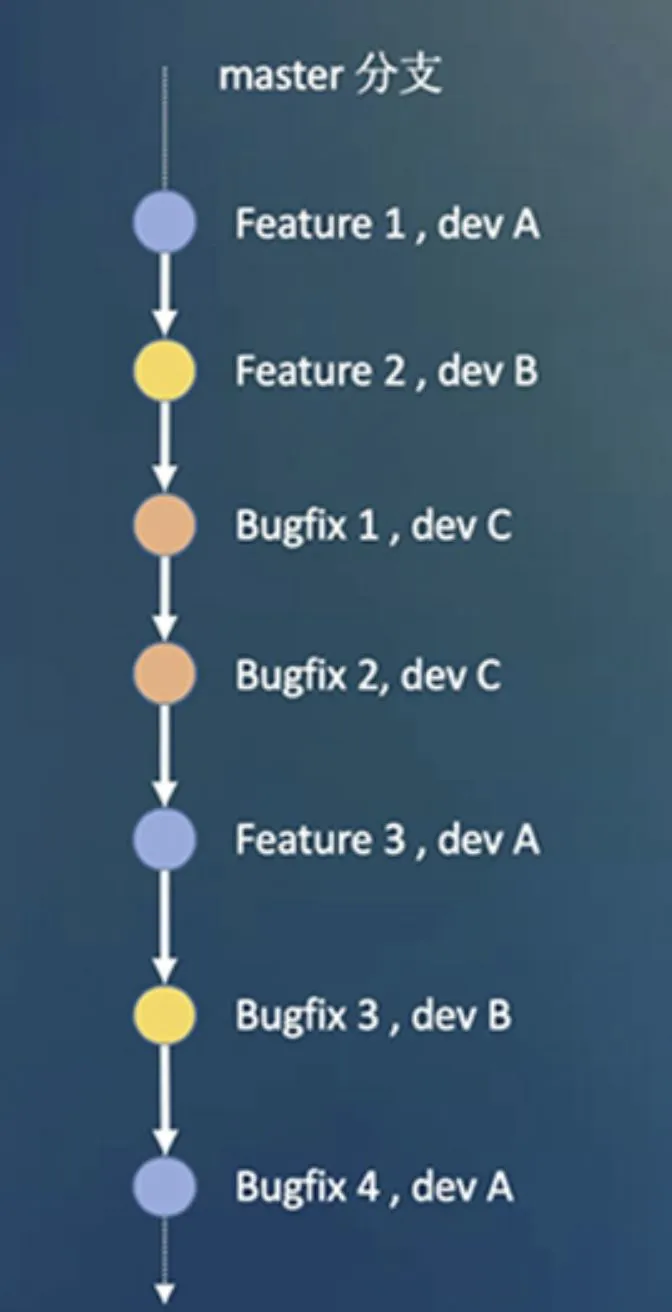
集中式工作流，这种工作方式对于使用过 svn 的同学想必会非常的熟悉，让我们思考下在 svn 下的协作体验，不同的开发同学需要依次将本地的修改提交到服务器，如果有冲突就先解决本地的冲突再提交，这个过程中远端的服务器就像是一个集中管理者，管理着所有人的代码提交，所以 svn 的开发协作流程就是典型的集中式工作流，那切换到 git 场景下，集中式工作流的工作方式又是什么样的呢？

首先我们看下 git 的基础操作框架，如图 2.1 所示：



这里有一份中央仓库，是存放项目代码的地方，三个开发人员 A、B、C 分别在本地持有一份中央仓库的拷贝 - 本地仓库，这里相比于 svn 的框架只是多了一个本地仓库；

接下来我们再来看在项目开发进行了一段时间之后的提交日志是什么样的，如图 2.2 所示：



这里是一条最简单的 master 分支上的提交日志记录，那相比于 svn 的框架有啥区别呢，只要把 master 分支字样改成 trunk 就变成了一条 svn 的提交记录。

最后，我们考虑以下几个条件：

1、有无本地仓库

2、默认分支是 master 还是 trunk

3、提交操作使用 git command 还是 svn command（细节忽略）

我们可以看出 svn 下的集中式工作流同样适用于 git，只要大家把 svn 相关的概念全部切换到 git 下即可：

1、认识本地仓库

2、认识默认分支 master

3、使用 git 的提交命令

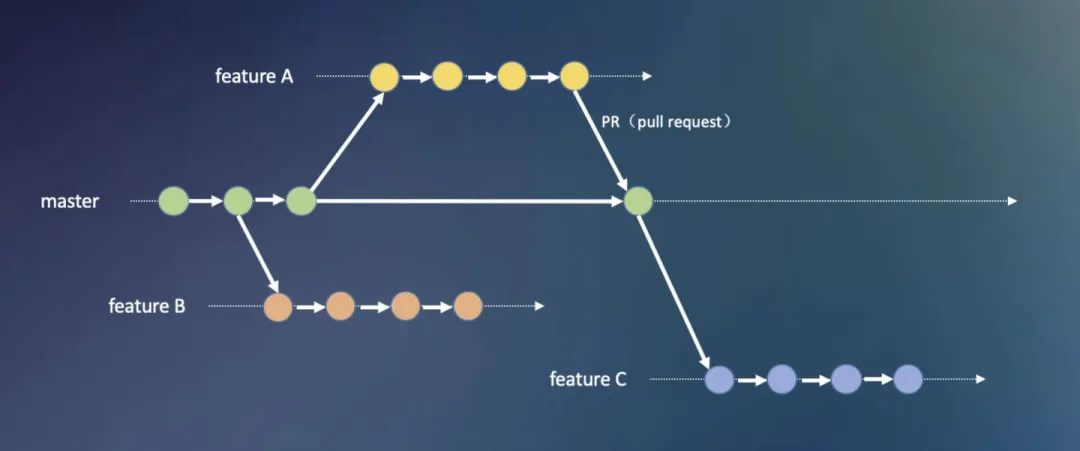
以上三点中的前两点对于集中式工作流下的开发者其实是透明的，开发者只需要将提交命令改成 git 就可以无缝衔接 svn 下的集中式工作流！

所以，svn 切换到 git 的成本其实还是很低的，只需要你掌握 git 的基础提交命令！

git 下的集中式工作流，是一种只使用 master 主分支的开发方式，这种方式简单明了，但是缺点是不同开发人员的提交日志混杂在一起，难以定位问题。

## 3. 功能分支工作流

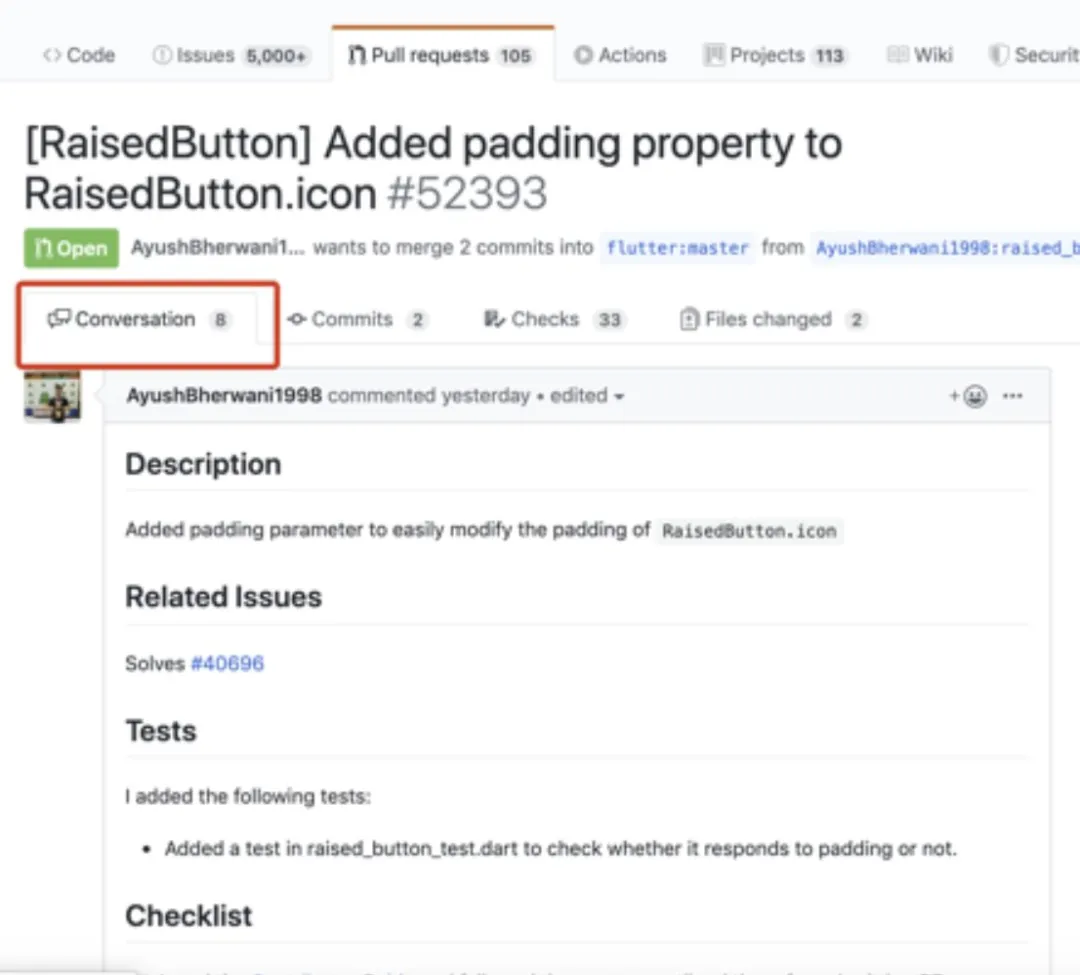
功能分支工作流，这种工作方式是以集中式工作流为基础，再为不同功能开发分配单独的功能分支来进行的；这种工作流的主干分支仍然是 master 分支，但是开发者在进行日常需求开发时不能将代码直接提交到 master 分支上，一般是为特定的需求新建一个功能分支，并且取一个具有描述性的名字，例如：feat-personal-page、issue-#1702，描述性的名称可以让其他开发者快速地明白这个功能分支的主要作用，提高不同开发者之间的协同效率；功能分支功能流的提交日志记录如图 3.1 所示：



从图中可以看出，相比于集中式工作流，分支历史看起来更加简洁、合理，让不同功能的开发进行隔离，避免不同功能代码之间产生不利的影响。

此外，在功能分支上的需求开发完成之后，我们需要将分支合并到主干分支 master 上，这时候需要进行的操作是 pull request，为什么要进行 PR 操作，而不是直接进行代码的 merge 呢，这里首先需要大家认识 PR 是什么操作，其次需要大家了解 PR 操作的意义；

功能需求开发完成之后，需要将本地功能分支推送到中央仓库的功能分支上，然后在中央仓库的功能分支上发起一个 pull request 请求去将功能分支上的修改合并到 master 分支上，这个过程一般是在 GIT 的项目主页上进行，公司内部就工蜂的项目主页，如图 3.2 所示，是 flutter 项目的某一次 PR 详情：



PR 操作给项目带来的益处有两点：1、code review2、讨论代码的公共平台

前者是每次 PR 操作发生时会通知相关者来检查待合并的代码，在检查过程中即完成了对代码的检视，这个过程保障了 master 分支上的已合并代码的健壮性；后者则是因为每次 PR 都会有一个 PR 详情主页，如图 3.2，每一个开发者都可以针对代码的实现提出自己的意见，使得讨论代码变成更加便捷高效，且为代码变更回顾提供了可能。

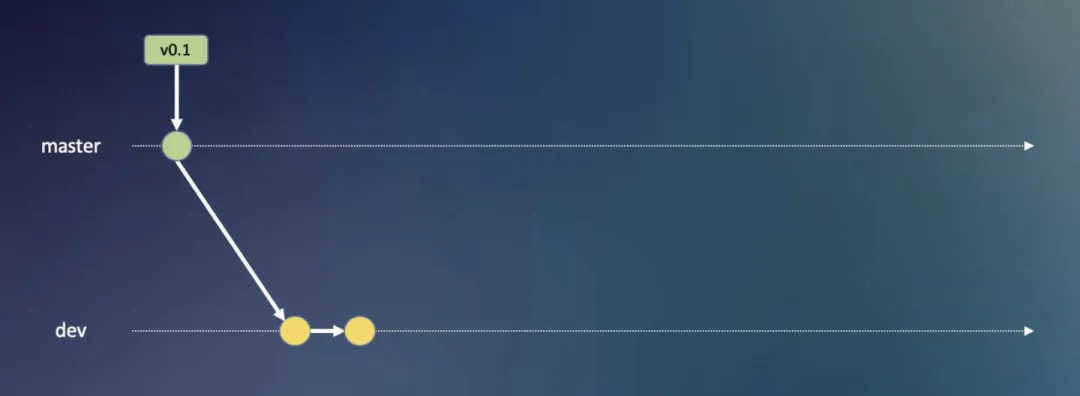
功能分支工作流是 git 项目开发非常灵活使用的一种方式，但是对于大型的项目而言，需要为不同的分支分配更加具体的角色。

## 4.Gitflow 工作流

Gitflow 工作流是目前非常成熟的一个方案，它定义了一个围绕项目发布的严格分支模型，通过为代码研发、项目发布以及维护分配独立的分支来让项目的迭代过程更加地顺畅，不同于之前的集中式工作流以及功能分支工作流，gitflow 工作流常驻的分支有两个：主干分支 master、开发分支 dev，此外针对项目研发的各个阶段，设定了特定的分支。

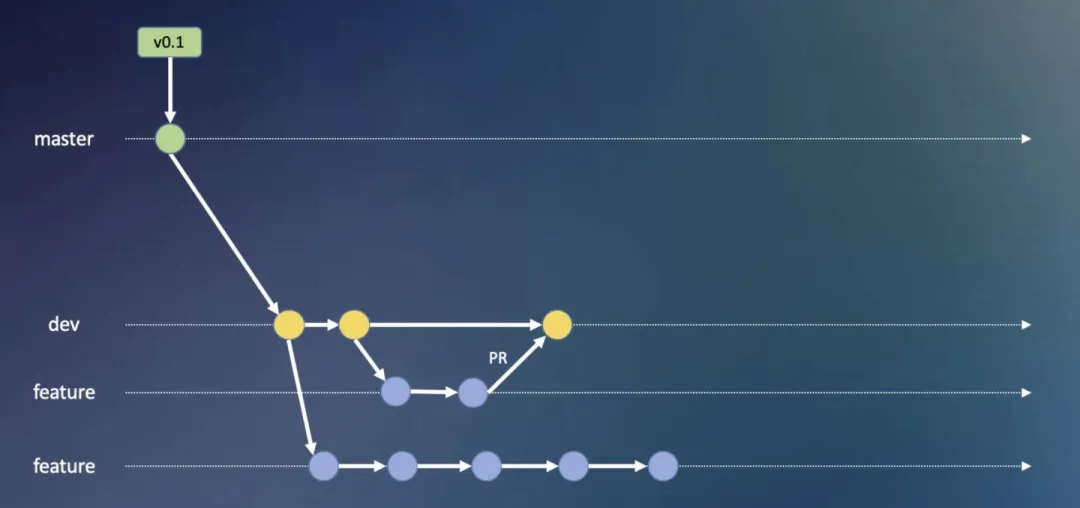
**阶段分支**常驻 master、dev 研发 feature 热修复 hotfix 发布 release

首先针对常驻分支，如图 4.1



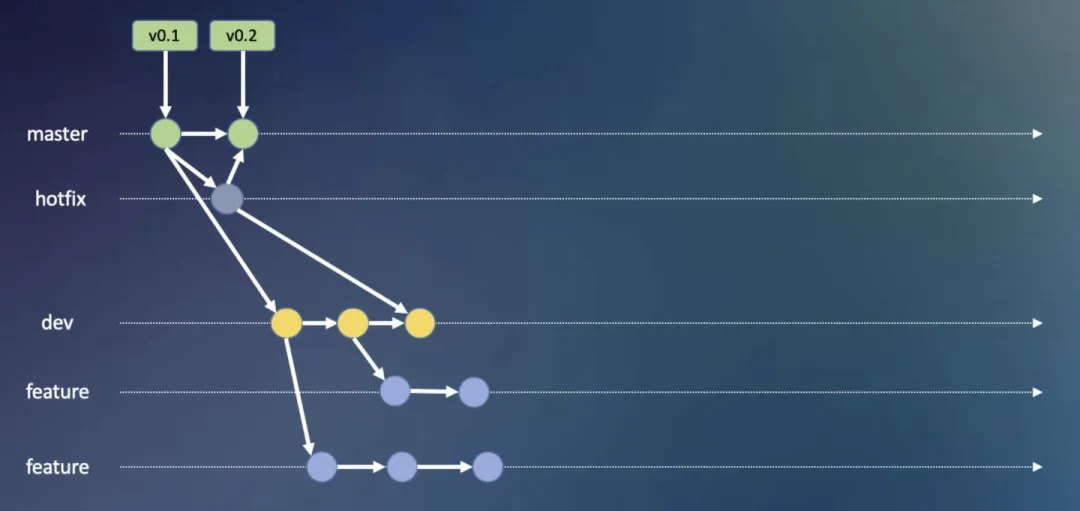
常驻分支表示在项目提交历史中一直存在的分支，这里 master 分支主要跟踪项目正式发布的代码历史，dev 分支主要跟踪项目代码研发的提交历史；此外在 master 分支上通常会为某次版本发布分配一个标签来记录版本号，这为以后项目排查定位提供便利。

接下来，我们来看 gitflow 工作流中，代码研发阶段的工作流程。



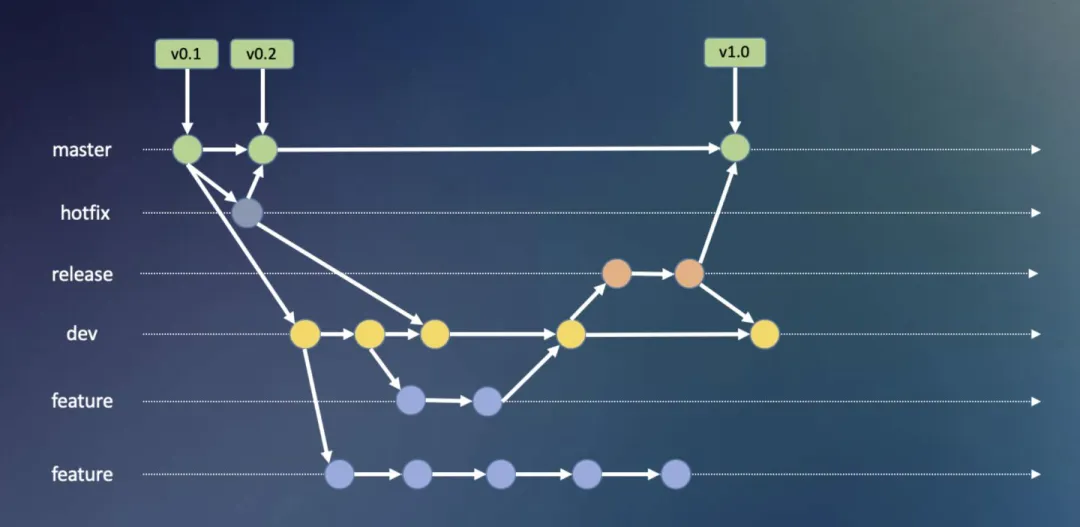
如图 4.2 所示，开发阶段开启某一个需求时需要从 dev 分支上新建功能分支 feature，图中所示为两个 feature 分支，代表同时有两个功能在开发中，这里的 feature 分支使用跟功能分支工作流中的使用方式是一样的，在需求开发完成之后需要提交 PR 请求合并进 dev 分支，完成之后即可删除对应的功能分支。

很多时候，在需求研发过程中，线上的代码可能会出现问题，这时候需要我们进行及时的修复，这就是项目迭代过程中的热修复阶段。



如图 4.3 所示，假设我们在开发的过程中线上出现了一个 bug，这时候我们需要从 master 的标签 v0.1 上检出一份分支代码 hotfix，修复并验证好了之后，需要将 hotfix 代码分别合并到 master /dev 分支上，并在 master 的提交上打上一个标签 v0.2，这里需要将热修复的代码分别合并进两个常驻分支是因为需要保障两边代码的一致性。

最后，我们来看下项目迭代的发布阶段，我们需要将之前功能开发完成的特性发布到线上去，如图 4.4 所示



首先在 dev 分支的提交处新建 release 分支，在这个分支上进行 bug 修复、面向发布的一些任务，这个分支不做任何功能上的任务，完成之后将 release 分支再分别合并进 master/dev 分支，并在 master 提交上打上标签 v1.0，这样一个发布阶段的代码操作就完成了

最后我们来看发布之后的目前的日志记录情况，如图 4.5 所示，这里可以将没有用的分支 hotfix、release、feature 均删除了，可以看出我们的常驻分支就 master/dev，最下面的 feature 表示仍在开发中。

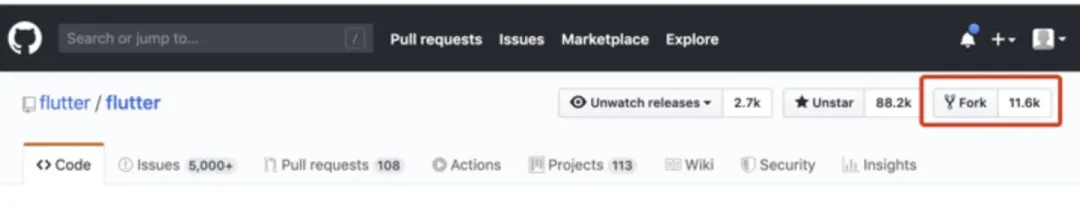
gitflow 工作流是目前比较很成熟的方案，它的优点有：

1、发布迭代流程更顺畅2、使得代码有了更加严谨的项目结构，方便定位排查问题

大型的项目 / 迭代速度快的推荐使用这种工作流程！

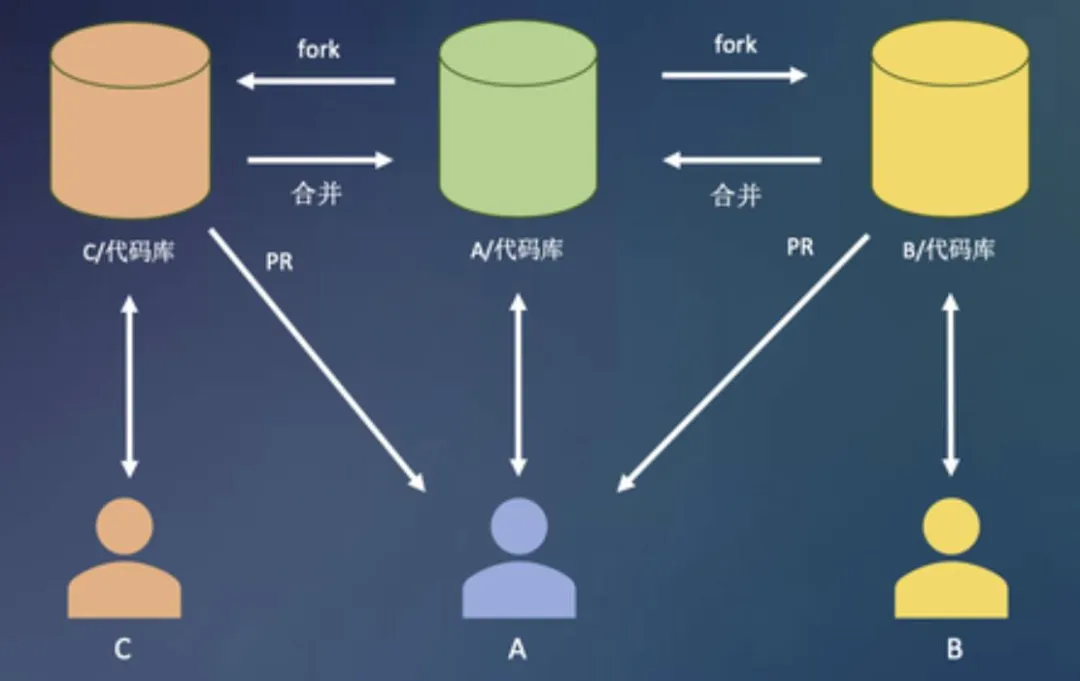
## 5.Forking 工作流

最后介绍一种开源项目常用的工作流 ——Forking 工作流，介绍之前首先需要了解什么是 fork 操作，如图 5.1 所示：



fork 操作是在个人远程仓库新建一份目标远程仓库的拷贝，操作很简单，比如 github 上在项目的主页点击 fork 按钮即可。

明白了 fork 操作之后，我们来看下 forking 工作流的流程，如图 5.2 所示：

首先开发者 A 拥有一个远端仓库，这时候有一个开发者 C 也想参与 A 的这个项目的开发工作，那他就可以 fork 一份 A 的这个仓库，之后在 c 的个人仓库里就有了这份代码库，后续开发者 C 就可以在自己的这个项目里进行开发工作，c 在完成了某个功能的实现之后，可以给 A 的仓库发一个 PR 请求，这时候会通知到开发者 A 有新的 PR，A 如果有问题可以直接在这个 PR 里提，开发者 C 可以进行进一步的修改，最后 A 通过了 C 的这份 PR 请求，就会将 C 的代码合并进 A 的仓库，这样就完成了 A / 代码库新特性的开发。同时如果有其他开发者对 A 的项目有兴趣也会进行相同的操作。

这里注意到 开发者 B/C 并不是 A 代码库的开发人员，而是第三方开发者，所以这种工作流主要用于开源项目！

## 6. 总结

最后回顾下这几种 git 工作流，集中式工作流可以说是 git 工作流的基础，初学者可以无缝地从 svn 的模式切换到 git 的模式；功能分支工作流在集中式的基础上又引入了功能分支，灵活地利用了 git 的分支特性，功能分离 / PR 优化了日常工作的效率；gitflow 工作流则是为大型项目的迭代过程服务的，指定了一个严格的分支模型，使得迭代流程更加顺畅；forking 工作流则是开源项目的首选，想要为开源项目做贡献就必须要懂得这种工作流！