Git 的研发流程 | 青训营笔记这是我参与「第三届青训营 - 后端场」笔记创作活动的的第3篇笔记 这里一下总结工作流的类

原文链接

工作流

类型	代表平台	特点	合入方式
集中式工作流	Gerrit / SVN	只依托于主干进行开发，不存在其他分支	Fast-forward
分支管理工作流	Github/Gitlab	可以定义不同特性的开发分支，在开发分支完成开发后再通过MR/PR合入主干分支	自定义，Fast-forword or Three-Way Merge都可以

集中式工作流

指只依托master分支进行研发活动

工作方式

1. 获取远端master代码
2. 直接在master分支完成修改
3. 提前拉取最新的master代码和本地代码和本地代码进行合并（使用rebase），如果有冲突需要解决冲突
4. 提交本地代码到master

集中式工作流-Gerrit

优点

1. 提供强制的代码评审机制，保证代码质量
2. 提供更丰富的权限，可以针对分支做细粒度的权限管控
3. 保证master的历史整活性
4. Aosp多仓的场景支持更好

缺点

1. 开发人员较多的情况下，更容易出现冲突
2. 对于多分支的支持较差，想要区分多个版本的线上代码时，更容易出现问题
3. 一般只有管理员才能创建仓库，比较难以在项目之间形成代码复用，比如类似的fork操作就不支持。

分支管理工作流

分支管理工作流	特点
Git Flow	分支类型丰富，规范严格
Github Flow	只有主干分支和开发分支，规则简单
Gitlab Flow	在主干分支和开发分支之上构建环境分支、版本分支，满足不同发布or环境的需要

分支管理工作流-Git Flow

Git Flow 对比较早期出现的分支管理策略。

- 包含五种类型的分支
    - Master ----- 主干分支
    - Develop ---- 开发分支
    - Feature ----- 特性分支
    - Release ----- 发布分支
    - Hotfix ---- 热修复分支
- 优点

  按照定义的标准严格执行， 代码会很清晰，并且很难出现混乱
- 缺点

  流程过于复杂，上线的节奏会比较慢

  由于太复杂研发容易不按照标准执行，从而导致代码出现混乱

分支管理工作流-Github Flow

github的工作流，只有一个主干分支，基于Pull Request（即常说的pr）往主干分支中提交代码。

团队合作的方式：

1. owner创建好仓库后，其他用户通过fork的方式来创建自己的仓库，并在fork的仓库上进行开发
2. owner创建好仓库后，统一给团队内成员分配权限，直接在同一个仓库进行开发

创建一个Pull Request

1. 创建一个main主分支（以前默认master为主分支）
2. 创建一个feature分支
3. 创建一个feature到main的Pull Request

可以在Pull Request页面执行CI / CA / CR等操作，都检查通过后，执行合入。

分支管理工作流-Gitlab Flow

Gitlab推荐的工作流实在GitFlow和Github Flow上做出优化，既保持了单一主分支的简便，又可以适应不同的开发环境。

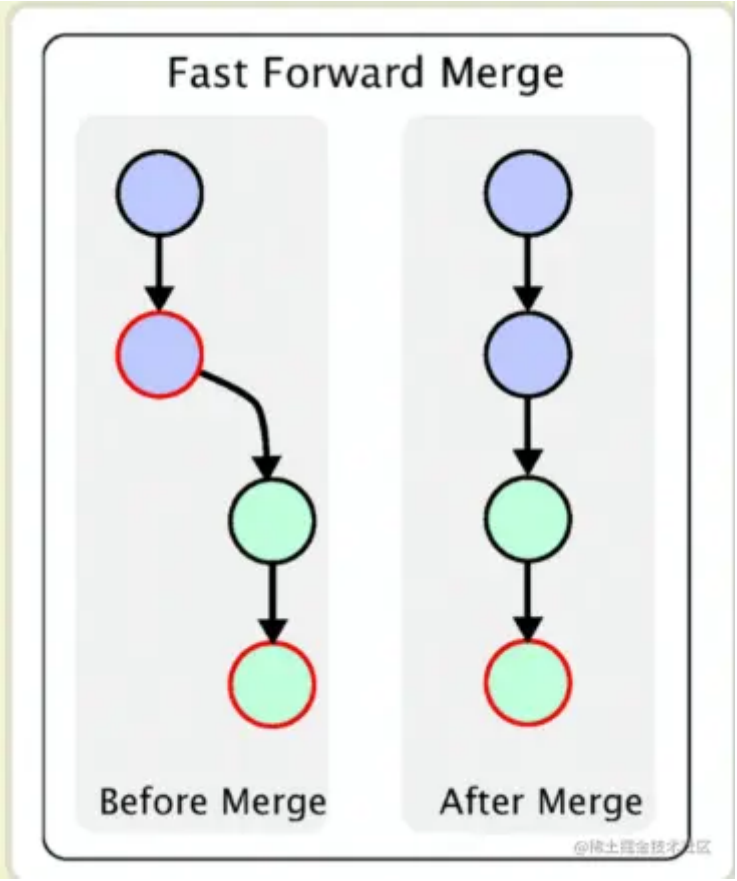
原则：upstream first上游优先

只有在上游分支采纳的代码才可以进入到下游分支，一般上游分支就是master

代码合并

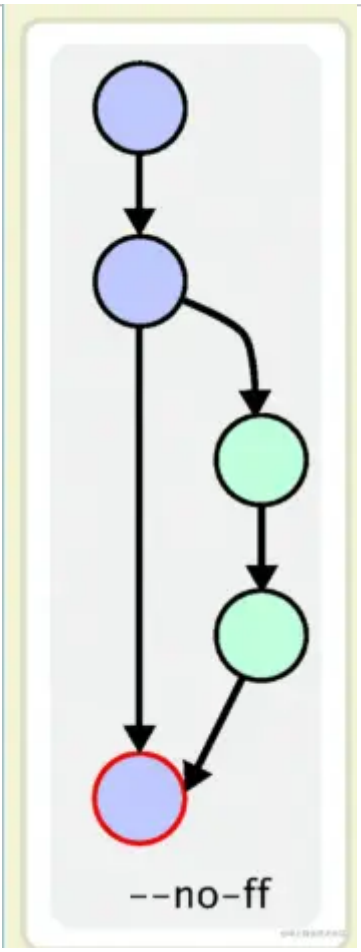
Fast-Forward

不会产生一个merge节点，合并后保持一个线性历史，如果target分支有了更新，则需要通过rebase操作更新source branch后才可以合入。



Three-Way Merge

三方合并，会产生一个新的merge节点



如何选择合适的工作流

选择原则

- 没有最好，只有最合适的
- 针对小型团队合作，推荐使用Github工作流即可

1. 尽量保证少量多次，最好不要一次性提交上千行代码
2. 提交Pull Request后最少需要保证有CR后再合入
3. 主干分支尽量保证整洁，使用fast-forward合入方式，合入前进行rebase

- 大型团队合作，根据自己的需要指定不同工作流，不需要局限在某种流程中。

常见问题

1. 在Gerrit平台上使用Merge的方式合入代码 Gerrit 是集中式工作流，不推荐使用Merge方式合入代码，应该是在主干分支开发后，直接Push

2. 不了解保护分支，Code Review，CI等概念，研发不规范保护分支：防止用户直接向主干分支提交代码，必须通过PR来进行合入。

Code Review，CI：都是在合入前的检查策略，Code Review是人工进行检查，CI则是通过一些定制化的脚本来进行检验。

3. 代码历史混乱，代码合并方式不清晰

不理解Fast-Forward和Three-Way Merge的区别，本地代码更新频繁的使用Three-Way的方式，导致生成过多的Merge节点，使提交历史变得复杂不清晰。