从0开始学习 GitHub 系列之「团队合作利器 Branch」

原创 2016-07-04 stormzhang AndroidDeveloper

阅读本文大概需要 8 分钟。

Git 相比于 SVN 最强大的一个地方就在于「分支」，Git 的分支操作简直不要太方便，而实际项目开发中团队合作最依赖的莫过于分支了，关于分支前面的系列也提到过，但是本篇会详细讲述什么是分支、分支的具体操作以及实际项目开发中到底是怎么依赖分支来进行团队合作的。

1 什么是分支？

我知道读者中肯定有些人对分支这个概念比较模糊，其实你们可以这么理解，你们几个人一起去旅行，中间走到一个三岔口，每条路可能有不同的风景，你们约定 3 天之后在某地汇聚，然后各自出发了。而这三条分叉路就可以理解成你们各自的分支，而等你们汇聚的时候相当于把你们的分支进行了合并。

2 分支的常用操作

通常我们默认都会有一个主分支叫 master ，下面我们先来看下关于分支的一些基本操作：

新建一个叫 develop 的分支

git branch develop

这里稍微提醒下大家，新建分支的命令是基于当前所在分支的基础上进行的，即以上是基于 mater 分支新建了一个叫做 develop 的分支，此时 develop 分支跟 master 分支的内容完全一样。如果你有 A、B、C三个分支，三个分支是三位同学的，各分支内容不一样，如果你当前是在 B 分支，如果执行新建分支命令，则新建的分支内容跟 B 分支是一样的，同理如果当前所在是 C 分支，那就是基于 C 分支基础上新建的分支。

切换到 develop 分支

git checkout develop

如果把以上两步合并，即新建并且自动切换到 develop 分支：

git checkout -b develop

把 develop 分支推送到远程仓库

git push origin develop

如果你远程的分支想取名叫 develop2 ，那执行以下代码：

git push origin develop:develop2

但是强烈不建议这样，这会导致很混乱，很难管理，所以建议本地分支跟远程分支名要保持一致。

查看本地分支列表

git branch

查看远程分支列表

git branch -r

删除本地分支

git branch -d develop

git branch -D develop (强制删除)

删除远程分支

git push origin :develop

如果远程分支有个 develop ，而本地没有，你想把远程的 develop 分支迁到本地：

git checkout develop origin/develop

同样的把远程分支迁到本地顺便切换到该分支：

git checkout -b develop origin/develop

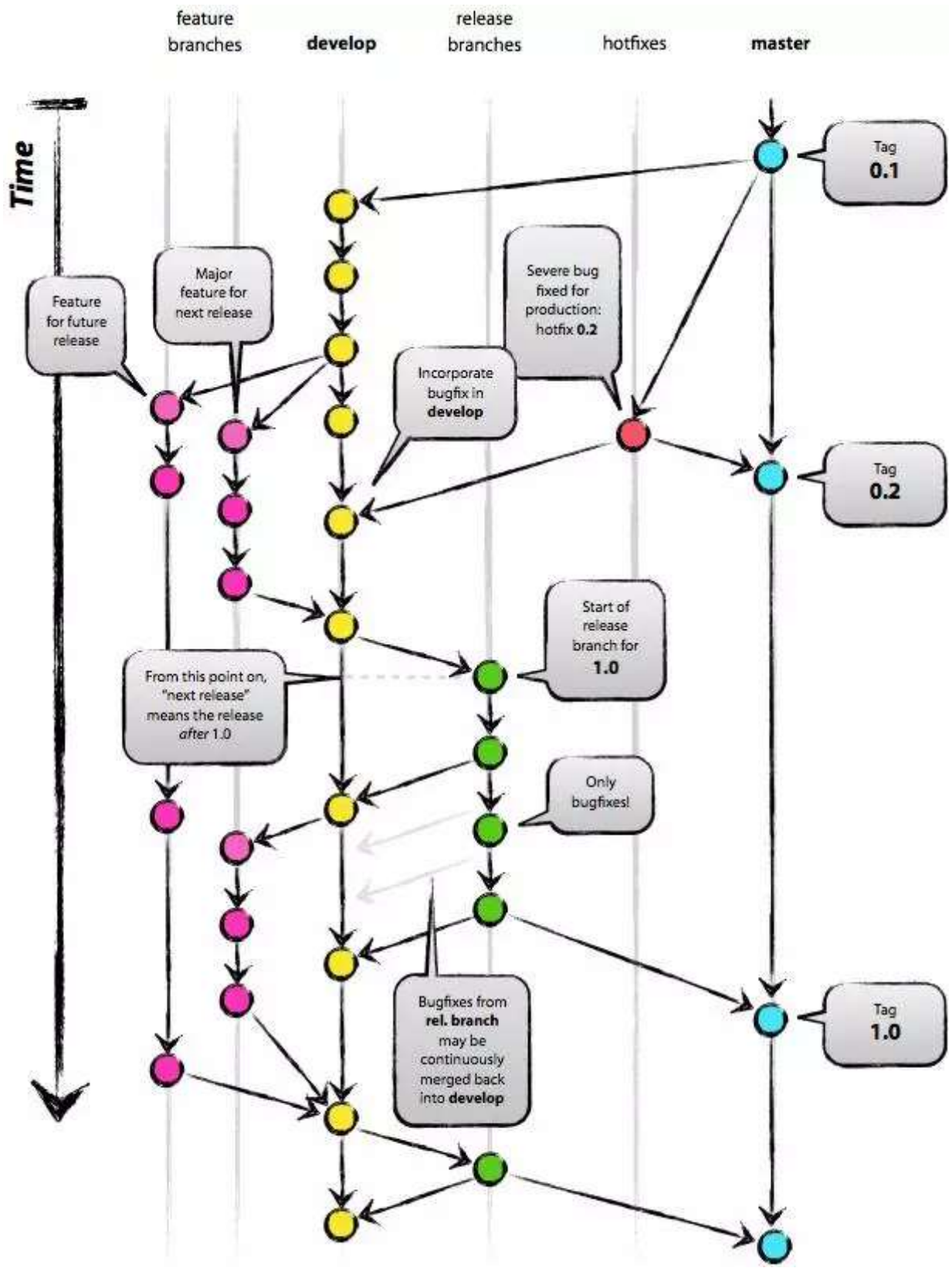
3 基本的团队协作流程

一般来说，如果你是一个人开发，可能只需要 master、develop 两个分支就 ok 了，平时开发在 develop 分支进行，开发完成之后，发布之前合并到 master 分支，这个流程没啥大问题。

如果你是 3、5 个人，那就不一样了，有人说也没多大问题啊，直接可以新建 A、B、C 三个人的分支啊，每人各自开发各自的分支，然后开发完成之后再逐步合并到 master 分支。然而现实却是，你正在某个分支开发某个功能呢，这时候突然发现线上有一个很严重的 bug ，不得不停下手头的工作优先处理 bug ，而且很多时候多人协作下如果没有一个规范，很容易产生问题，所以多人协作下的分支管理规范很重要，就跟代码规范一样重要，以下就跟大家推荐一种我们内部在使用的一种分支管理流程 Git Flow。

4 Git Flow

准确的说 Git Flow 是一种比较成熟的分支管理流程，我们先看一张图能清晰的描述他整个的工作流程：



第一次看上面那个图是不是一脸懵逼？跟我当时一样，不急，我来用简单的话给你们解释下。

一般开发来说，大部分情况下都会拥有两个分支 master 和 develop，他们的职责分别是：

- **master**：永远处在即将发布(production-ready)状态
- **develop**：最新的开发状态

确切的说 master、develop 分支大部分情况下都会保持一致，只有在上线前的测试阶段 develop 比 master 的代码要多，一旦测试没问题，准备发布了，这时候会将 develop 合并到 master 上。

但是我们发布之后又会进行下一版本的功能开发，开发中间可能又会遇到需要紧急修复 bug，一个功能开发完成之后突然需求变动了等情况，所以 Git Flow 除了以上 master 和 develop 两个主要分支以外，还提出了以下三个辅助分支：

- **feature**: 开发新功能的分支, 基于 develop, 完成后 merge 回 develop
- **release**: 准备要发布版本的分支, 用来修复 bug，基于 develop，完成后 merge 回 develop 和 master
- **hotfix**: 修复 master 上的问题, 等不及 release 版本就必须马上上线. 基于 master, 完成后 merge 回 master 和 develop

什么意思呢？

举个例子，假设我们已经有 master 和 develop 两个分支了，这个时候我们准备做一个功能 A，第一步我们要做的，就是基于 develop 分支新建个分支：

git branch feature/A

看到了吧，其实就是一个规范，规定了所有开发的功能分支都以 feature 为前缀。

但是这个时候做着做着发现线上有一个紧急的 bug 需要修复，那赶紧停下手头的工作，立刻切换到 master 分支，然后再此基础上新建一个分支：

git branch hotfix/B

代表新建了一个紧急修复分支，修复完成之后直接合并到 develop 和 master ，然后发布。

然后再切回我们的 feature/A 分支继续着我们的开发，如果开发完了，那么合并回 develop 分支，然后在 develop 分支属于测试环境，跟后端对接并且测试的差不多了，感觉可以发布到正式环境了，这个时候再新建一个 release 分支：

git branch release/1.0

这个时候所有的 api、数据等都是正式环境，然后在这个分支上进行最后的测试，发现 bug 直接进行修改，直到测试 ok 达到了发布的标准，最后把该分支合并到 develop 和 master 然后进行发布。

以上就是 Git Flow 的概念与大概流程，看起来很复杂，但是对于人数比较多的团队协作现实开发中确实会遇到这么复杂的情况，是目前很流行的一套分支管理流程，但是有人会问每次都要各种操作，合并来合并去，有点麻烦，哈哈，这点 Git Flow 早就想到了，为此还专门推出了一个 Git Flow 的工具，并且是开源的：

GitHub 开源地址：https://github.com/nvie/gitflow

简单点来说，就是这个工具帮我们省下了很多步骤，比如我们当前处于 master 分支，如果想要开发一个新的功能，第一步切换到 develop 分支，第二步新建一个以 feature 开头的分支名，有了 Git Flow 直接如下操作完成了：

git flow feature start A

这个分支完成之后，需要合并到 develop 分支，然而直接进行如下操作就行：

git flow feature finish A

如果是 hotfix 或者 release 分支甚至会自动帮你合并到 develop、master 两个分支。

想必大家已经了解了这个工具的具体作用，具体安装与用法我就不多提了，感兴趣的可以看我下我之前写过的一篇博客：

http://stormzhang.com/git/2014/01/29/git-flow/

5 总结

以上就是我分享给你们的关于分支的所有知识，一个人你也许感受不到什么，但是实际工作中大都以团队协作为主，而分支是团队协作必备技能，而 Git Flow 是我推荐给你们的一个很流行的分支管理流程，也是我们薄荷团队内部一直在实践的一套流程，希望对你们有借鉴意义。

相关文章：

从0开始学习 GitHub 系列之「初识 GitHub」

从0开始学习 GitHub 系列之「加入 GitHub」

从0开始学习 GitHub 系列之「Git速成」

从0开始学习 GitHub 系列之「向GitHub 提交代码」

从0开始学习 GitHub 系列之「Git 进阶」

阅读原文