Git 代码分支模型（1）：GitFlow、GitHub Flow、Trunk Based Development

在 2018年6月9日 上张贴 由 钱五哥 2条评论

Post Views: 4,406

最近和项目组几位同志（WBH、CLF、QEN）讨论了代码分支模型。重点讨论了3种模型，特别是讨论了GitFlow的适用性。

本系列一共3篇文章：

（1）GitFlow、GitHubFlow 和 TBD

（2）GitLab Flow

（3）Atlassian Simple Git Flow & Google Upstream

一、GitFlow模型。Vincent Driessen在2010年提出的GitFlow模型。这个模型的特点是只有2个主干分支，Master和Develop分支：Master分支上只有稳定的生产版本，Develop分支用于集成。其中还涉及到HotFix分支。而其他还有三类分支：Feature分支用于开发人员各自开发；Release用于代码合并和集成；HotFix用于产品版本代码的紧急修订。下图是经典的模型图

订阅全站

开源软件的安全和质量问题调研

VMware Workstation Player开源了？

拆：倒车雷达，工作原理很简单

GoogleEarth：巨大的北京大兴国际机场

开始使用Open Live Writer

友情链接

PostgreSQL中文网

百度技术运维团队

High Scalability

James Hamilton

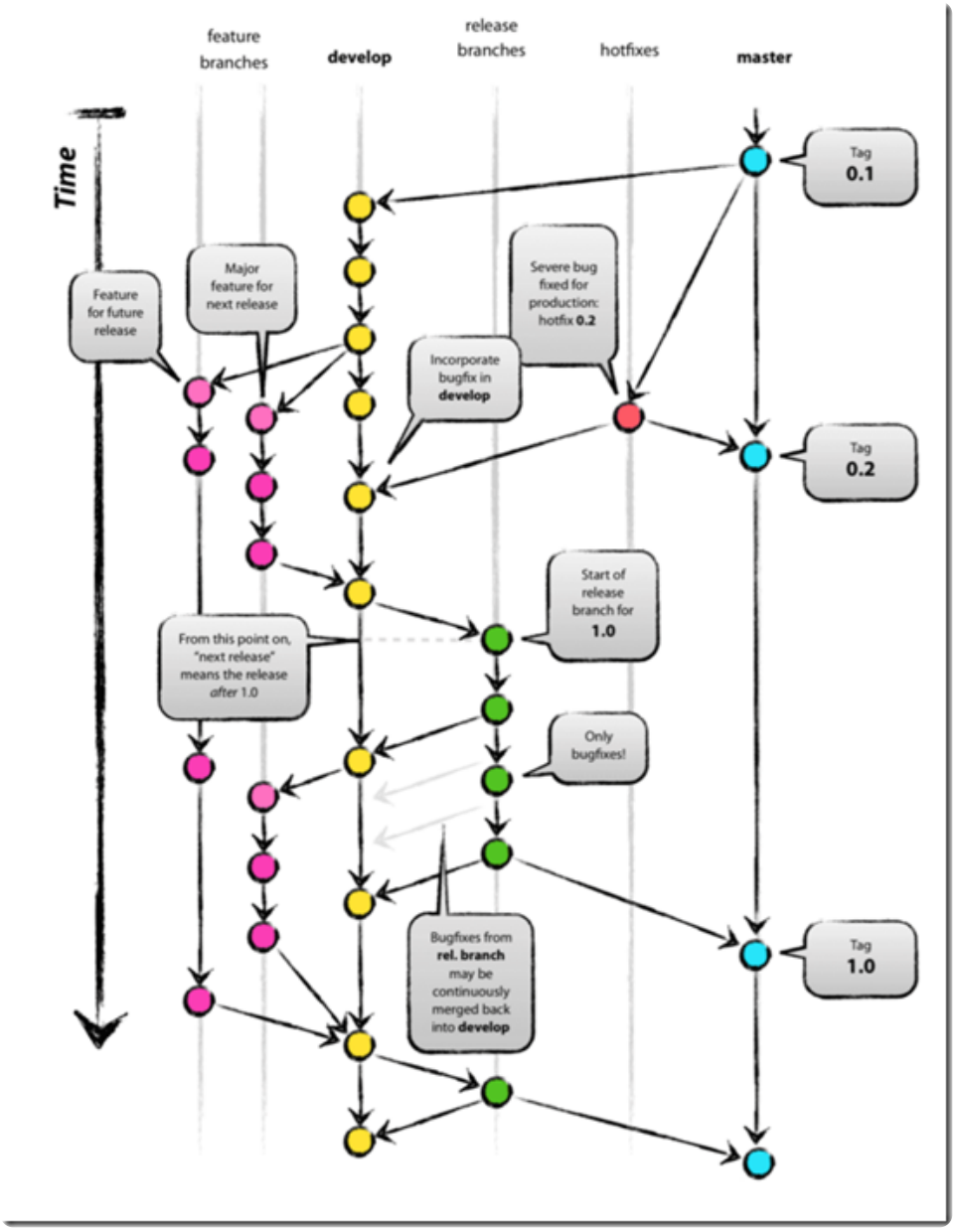
空军之翼

分类目录

IT产业

工业互联网

开源社区



不知道什么原因，ThoughtWorks这些年一直在抨击GitFlow模型，一年比一年严肃，后面我们会分析一下如何优化；

2011年：停用GitFlow的Feature分支

Disappointingly, we continue to see development teams embrace the practice of feature branching to isolate work and defer integration. Feature branches commonly cause significant pain and unpredictability during late merges, but more importantly prevent the continual design improvement necessary to maintain high quality software. We recommend continuous integration and branch by abstraction as an alternative to feature branching.

IT技术和产品
5G技术和业务
CAD
GIS
VOIP
WIFI技术和产品
中台
云计算和存储（IAAS）
人工智能
信息化应用（SAAS）
博客CMS
即时通信
大数据和数据库
安全
工业互联网
平台业务（PAAS）
广告业务
搜索引擎
机器翻译
桌面产品
研发工具链
个人兴趣
亲朋好友
军事政治

2015年：停用GitFlow的长时分支

Gitflow is a strict branching pattern for releases using Git. Although not an inherently bad pattern, we often see it misused. If the feature and develop branches are short lived and merged often, you are really using the power of Git, which makes these activities easy. However, a problem we often see is that these become long lived branches, which results in the dreaded merge conflicts many people began using Git to escape. A merge is a merge. Regardless of the source control tool or pattern you use. If you wait more than a day or two to merge, you could hit a big merge conflict. This becomes a real issue if you have a larger team. If you have more than a few people waiting to merge, you can have a serious a bottleneck. Introducing patterns like Gitflow require the discipline to merge often to be successful. So by all means use the pattern, but only if you have the discipline to prevent long lived branches

2016年：停用GitFlow

We firmly believe that long-lived version-control branches harm valuable engineering practices such as continuous integration, and this belief underlies our dislike for Gitflow. We love the flexibility of Git underneath but abhor tools that encourage bad engineering practices. Very short-lived branches hurt less, but most teams we see using Gitflow feel empowered to abuse its branch-heavy workflow, which encourages late integration (therefore discouraging true continuous integration).

这些内容的核心在于GitFlow重视管理，强调多分支开发，会造成集成滞后，合并困难，影响持续集成。在2016年发布的文章中，TW给出了几个其他的建议，包括推荐GitHub Flow和Trunk-Based Developmeng

二、GitHub Flow。这个是GitHub的Scott Chacon于2011年提出的，针对的就是GitFlow。但是其中主要问题并不是针对GitFlow，而是针对有人开发的GitFlow Script，这些Script用于相对强制性地执行GitFlow规则，但是仅支持命令行，因此被很多人吐槽。Scott给出了GitHub的模型，即GitHub Flow。GitHub Flow希望针对的场景是每天发布几次产品代码的场景（个人认为这是DevOps场景，而

不是产品交付型场景），希望采用敏捷的场景每天多次解决小问题。

GitHub Flow只有Master和Feature分支，相当于把GitFlow的Master和Develop分支合并为一个分支。有如下6个环节。

1. Anything in the master branch is deployable
2. To work on something new, create a descriptively named branch off of master (ie: new-oauth2-scopes)
3. Commit to that branch locally and regularly push your work to the same named branch on the server
4. When you need feedback or help, or you think the branch is ready for merging, open a pull request
5. After someone else has reviewed and signed off on the feature, you can merge it into master
6. Once it is merged and pushed to 'master', you can and should deploy immediately

GitHub Flow的使用场景：

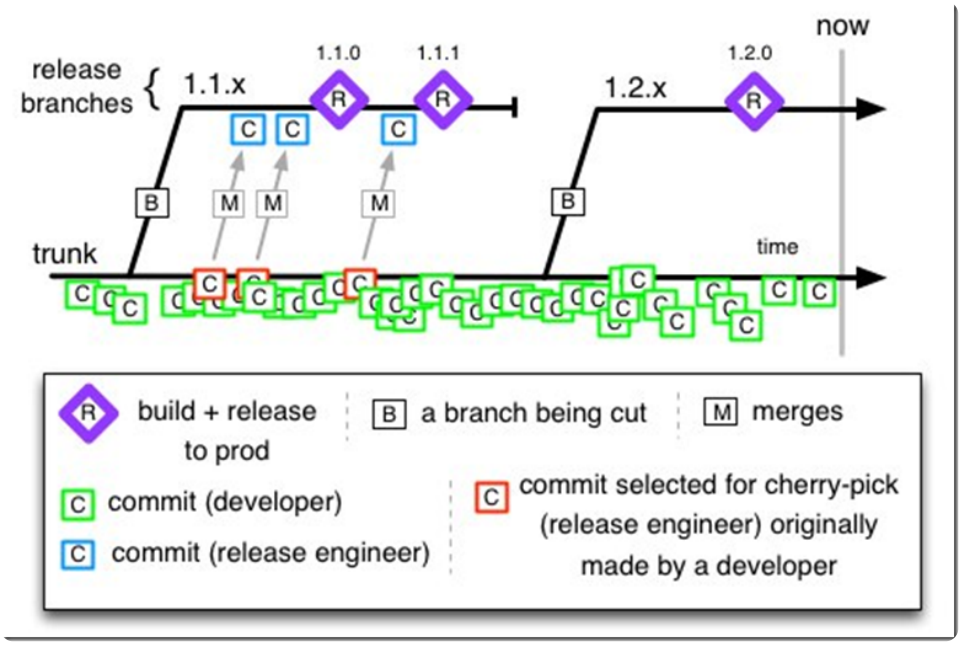
GitHub is 35 employees now, maybe 15-20 of whom work on the same project (github.com) at the same time. I think that most development teams – groups that work on the same logical code at the same time which could produce conflicts – are around this size or smaller.

三、Trunk Based development。这是Paul Hammant 2013年提出的模型，这个模型不用多解释，基本就是SVN的模型，在TBD模型中，所有开发都在主干，但是拉出新的分支交付。这个场景下，假设所有的特性开发都可以快速完成，这样就不会影响CI。

未分类
软件开发
APP开发
UIUE
产品经理
开源知识产权
文档管理
语言和工具

文章的标签

360 (4) AWS (10) BareMetal (4) Eclipse (3) Google (18) Hadoop (6) J2ME (5) KasperSky (3) Microsoft (19) MPx220 (4) MsnSpace (3) Nokia (8) SEO (4) Skype (7) Sony (3) StarCraft (4) Windows (5) Wordpress (3) 三体 (3) 中国移动 (8) 中国移动苏州研发中心 (4) 产品立项 (3) 刘慈欣 (3) 创新 (14) 华为 (11) 卡脖子技术 (3) 双城记 (22) 大云 (6) 家猫 (3) 开元通宝 (3) 开心网 (7) 搜狗 (3) 数博会 (4) 新浪 (3) 流浪猫 (7) 百度 (3) 盗墓笔记 (6) 科幻小说 (5) 苏州研发中心 (7) 裸金属 (4) 运20 (3) 野狼獾 (4) 项目管理 (3) 颐和园 (3) 黑暗料理 (3)



理论上，TBD单主干开发和敏捷核心要求比较匹配：

1. You've broken your application into multiple components.
2. Each component into a directory inside the trunk (possibly hierarchical).
3. Each directory its own source self-contained and its own build (possibly hierarchical).
4. You have a good set of unit tests and consider them important enough to illustrate snippets of example usage.
5. Continuous Integration drives things, even for hundreds of components, that drops items into a Maven-like repository. CruiseControl has a nice <httpfile/> directive that with the <include-projects/> directive that allows you to set up the killer CI installation that is branch-ready meaning you can run without a dedicated CI administrator.
6. Your management are good at release planning.
7. Developers are in the habit of never breaking the build 🙂

但是针对大型变更的时候（如替换某中间件），上述流程被打乱了，因此提出了BBA（Branch By Abstraction）

1. Introduce an abstraction over the core bits of the big thing you're going to change and commit
2. Update all the bits of code that were formerly using the thing directly to use it via the new abstraction and commit

3. Make a second implementation of the abstraction, with unit tests that specifically test its core functionality and commit
4. Update all the code from (2) to use the new implementation (still via the abstraction)* and commit
5. Deprecate the first implementation (or skip to 6 if you don't want a respectful grace period).
6. Delete the first implementation (its proven there is no need for you to go back).
7. Remove the abstraction (if it is inelegant).

这个是一种架构上的抽象层设计，可以让原有不相关模块继续开发，而仅变化直接相关的模块

简要比较一下上面3种模型：

分支模型	主干数	分支数	分支存活周期	使用场景	优化方向
GitFlow	2个	5类	长短不一，但是更适用于长周期	企业级开发，大型团队，重管理，敏捷要求相对较低。传统企业的文化，新手多	允许Develop分支开发，降低对Feature分支的管理要求
GitHub Flow	1	2类	短周期为主体	DevOps开发，中小型团队，敏捷要求高，全能团队，互联网企业的开发。开发者文化	优化长周期任务，类似TBD
TBD	1	2类	短周期为主体	DevOps开发，敏捷要求高，全能团队，互联网企业的开发。开发者文化	用BBA优化大型架构优化

相关信息：

1. GitFlow：A successful Git branching model 2010
2. 支持GitFlow模型的脚本
3. ThoughtWorks 2016：GitFlow有害论
4. ThoughtWorks 2011：停用Feature 分支
5. ThoughtWorks 2015：停用长时分支
6. ThoughtWorks 2016：停用GitFlow
7. GitHub Flow 2012
8. Trunkbased Development 2013
9. Branch by Abstraction, etc

张贴在研发管理、配置管理、项目管理　标签：Git、Git Flow、Github、GitHub Flow

← GoogleEarth：意外找到大唐大型无人机基地，翔龙和双头雕比翼！

《中华人民共和国促进科技成果转化法》的3个解读 →

《GIT 代码分支模型（1）：GITFLOW、GITHUB FLOW、TRUNK BASED DEVELOPMENT》有2个想法

Pingback： Git 代码分支模型（3）：Atlassian Simple Git Flow & Google Upstream – 钱五哥の自由空间

Pingback： Git 代码分支模型（2）：GitLab Flow介于GitFlow和Github Flow之间！ – 钱五哥の自由空间

发表评论

电子邮件地址不会被公开。 必填项已用*标注

评论

姓名 *

电子邮件 *

站点

发表评论