04 | 一切的源头，代码分支策略的选择

2018-07-12 王潇俊

讲述：王潇俊
时长 13:30 大小 6.19M

记得大概是一年前吧，我与好友老吴喝茶聊天时，讨论到：高效的持续交付体系，必定需要一个合适的代码分支策略。

我告诉老吴：“采用不同的代码分支策略，意味着实施不同的代码集成与上线流程，这会影响整个研发团队每日的协作方式，因此研发团队通常会很认真地选择自己的策略。”

老吴是一名有多年开发经验的资深架构师，当时正好要接手一个框架团队，从个人贡献者向团队管理者转型。他个人对代码管理工具可谓熟之又熟，甚至连“老古董”的 CVS 都可以跟你聊半天。但他在为团队制定代码分支管理策略时，还是慎之又慎，足见其重要性。

最后我们发现，要确定选用哪种代码分支管理策略，需要先假设几个问题，这几个问题有了答案，也就代表你找到了适合的方向。

你需要思考的几个问题如下：

1. Google 和 Facebook 这两个互联网大咖都在用主干开发（Trunk Based Development，简称 TBD），我们是不是也参照它俩，采用主干开发分支策略？
2. 用 Google 搜索一下，会发现有个排名很靠前的分支策略，叫“A successful Git branching model”（简称 Git Flow），它真的好用吗？团队可以直接套用吗？
3. GitHub 和 GitLab 这两个当下最流行的代码管理平台，各自推出了 GitHub Flow 和 GitLab Flow，它们有什么区别？适合我使用吗？
4. 像阿里、携程和美团点评这样国内知名的互联网公司，都在用什么样的分支策略？

今天，我想再沿着当时的思考路径，和你一起回顾和总结一下，希望能够带你全面了解代码分支策略，帮助你做出合适的选择。

谈谈主干开发（TBD）

主干开发是一个源代码控制的分支模型，开发者在一个称为 “trunk” 的分支（Git 称 master） 中对代码进行协作，除了发布分支外没有其他开发分支。

Google 和 Facebook 都是采用“主干开发”的方式，代码一般直接提交到主干的头部，这样可以保证所有用户看到的都是同一份代码的最新版本。

“主干开发”确实避免了合并分支时的麻烦，因此像 Google 这样的公司一般就不采用分支开发，分支只用来发布。

大多数时候，发布分支是主干某个时点的快照。以后的改 Bug 和功能增强，都是提交到主干，必要时 cherry-pick （选择部分变更集合并到其他分支）到发布分支。与主干长期并行的特性分支极为少见。

由于不采用“特性分支开发”，所有提交的代码都被集成到了主干，为了保证主干上线后的有效性，一般会使用特性切换（feature toggle）。特性切换就像一个开关可以在运行期间隐藏、启用或禁用特定功能，项目团队可以借助这种方式加速开发过程。

特性切换在大型项目持续交付中变得越来越重要，因为它有助于将部署从发布中解耦出来。但据吉姆 · 伯德（Jim Bird）介绍，特性切换会导致代码更脆弱、更难测试、更难理解和维护、更难提供技术支持，而且更不安全。

他的主要论据是，将未经测试的代码引入生产环境是一个糟糕的主意，它们引发的问题可能会在无意间暴露出来。另外，越来越多的特性切换会使得逻辑越来越混乱。

特性切换需要健壮的工程过程、可靠的技术设计和成熟的特性切换生命周期管理，如果不具备这三个关键的条件，使用特性切换反而会降低生产力。

根据上面的分析，主干开发的分支策略虽然有利于开展持续交付，但是它对开发团队的能力要求也更高。

主干开发的优缺点如表 1 所示。

优点	缺点
1. 频繁集成，每次集成冲突少，集成效率高。 2. 能享受持续交付带来所有的好处。 3. 无需在分支之间做切换。	1. 太多的团队成员同时工作在主干上，到发布的时候就可能出现“一粒老鼠屎坏了一锅粥”这样的灾难。 2. 要借助特性切换等机制来保证线上运行的正确性，这会引入新的问题。

表 1 主干开发的优缺点

谈谈特性分支开发

和主干开发相对的是 “特性分支开发” 。在这个大类里面，我会给你分析 Git Flow、GitHub Flow 和 GitLab Flow 这三个常用的模型。

第一，Git Flow

我们在 Google 上查关键词“branch model”（也就是“分支模型”），有一篇排名比较靠前的文章“A successful Git branching model”，它介绍了 Git Flow 模型。

Git 刚出来的那些年，可参考的模型不多，所以 Git Flow 模型在 2011 年左右被大家当作了推荐的分支模型，至今也还有项目团队在使用。然而，Git Flow 烦琐的流程也被许多研发团队吐槽，大家普遍认为 hotfix 和 release 分支显得多余，平时都不会去用。

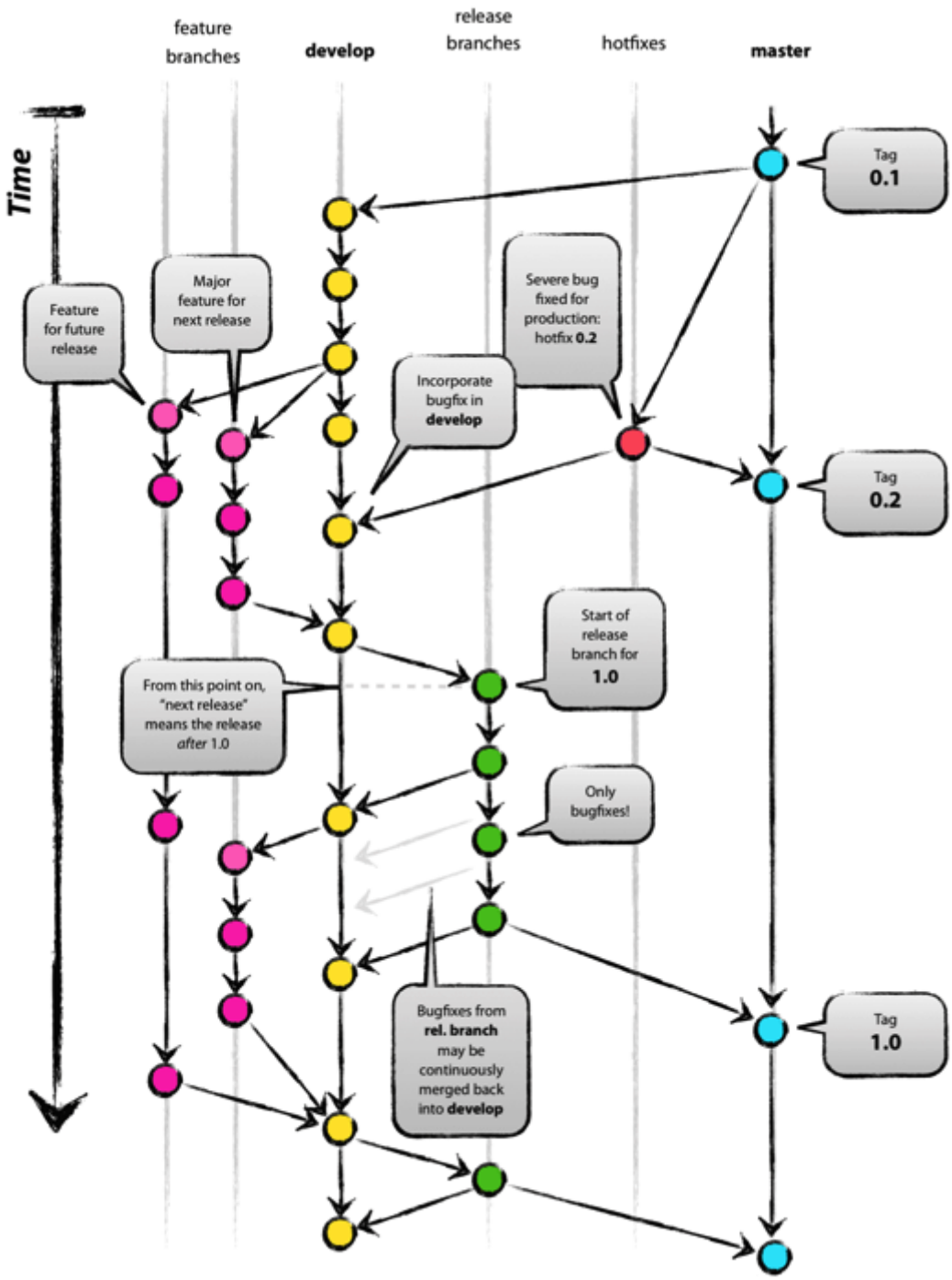

图 1 Git Flow 示意图

第二，GitHub Flow

GitHub Flow 是 GitHub 所使用的一种简单流程。该流程只使用 master 和特性分支，并借助 GitHub 的 pull request 功能。

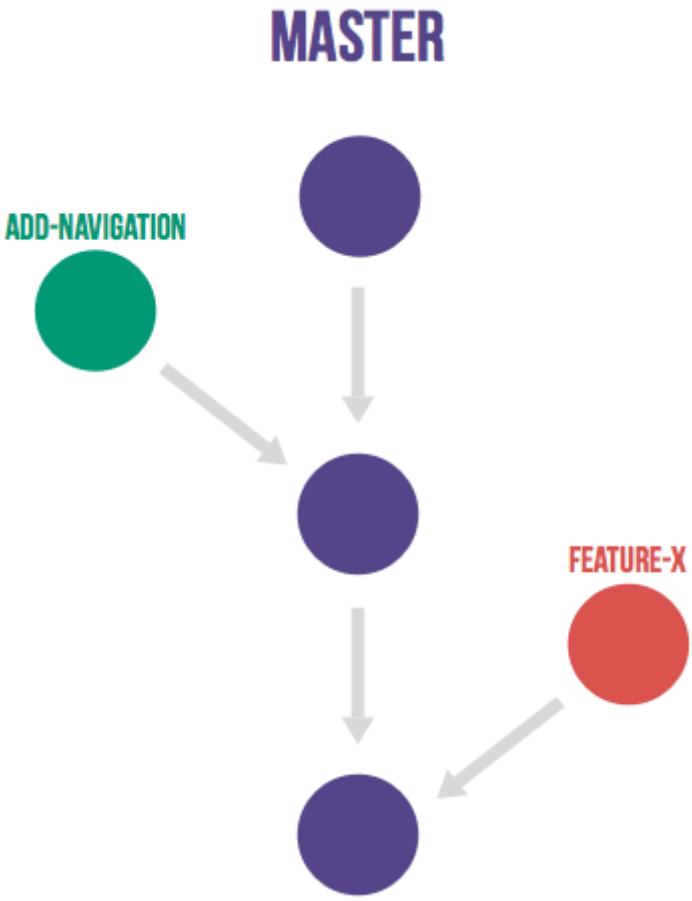

图 2 GitHub Flow 示意图

在 GitHub Flow 中，master 分支中包含稳定的代码，它已经或即将被部署到生产环境。任何开发人员都不允许把未测试或未审查的代码直接提交到 master 分支。对代码的任何修改，包括 Bug 修复、热修复、新功能开发等都在单独的分支中进行。不管是一行代码的小改动，还是需要几个星期开发的新功能，都采用同样的方式来管理。

当需要修改时，从 master 分支创建一个新的分支，所有相关的代码修改都在新分支中进行。开发人员可以自由地提交代码和提交到远程仓库。

当新分支中的代码全部完成之后，通过 GitHub 提交一个新的 pull request。团队中的其他人员会对代码进行审查，提出相关的修改意见。由持续集成服务器（如 Jenkins）对新分支进行自动化测试。当代码通过自动化测试和代码审查之后，该分支的代码被合并到 master 分支。再从 master 分支部署到生产环境。

GitHub Flow 的好处在于非常简单实用，开发人员需要注意的事项非常少，很容易形成习惯。当需要修改时，只要从 master 分支创建新分支，完成之后通过 pull request 和相关的代码审查，合并回 master 分支就可以了。

第三，GitLab Flow

上面提到的 GitHub Flow，适用于特性分支合入 master 后就能马上部署到线上的这类项目，但并不是所有团队都使用 GitHub 或使用 pull request 功能，而是使用开源平台 GitLab，特别是对于公司级别而言，代码作为资产，不会随意维护在较公开的 GitHub 上（除非采用企业版）。

GitLab Flow 针对不同的发布场景，在 GitHub Flow（特性分支加 master 分支）的基础上做了改良，额外衍生出了三个子类模型，如表 2 所示。

分支模型	说明	图示
带生产分支	1. 无法控制准确的发布时间，但又要求不停集成的。 2. 需要创建一个production分支来放置发布的代码。	<图3>
带环境分支	1. 要求所有代码都在逐个环境中测试通过。 2. 需要为不同的环境建立不同的分支。	<图4>
带发布分支	1. 用于对外界发布软件的项目，同时需要维护多个发布版本。 2. 尽可能晚地从master拉取发布分支。 3. Bug的修改应先合并到master，然后cherry pick到release分支。	<图5>

表 2 GitLab Flow 的三个分支

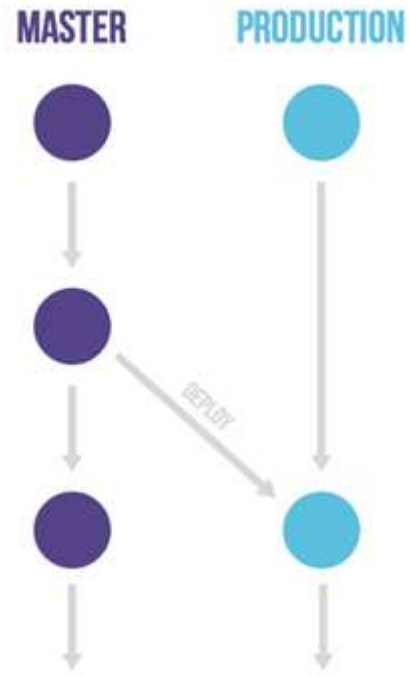

图 3 带生产分支的 GitLab Flow

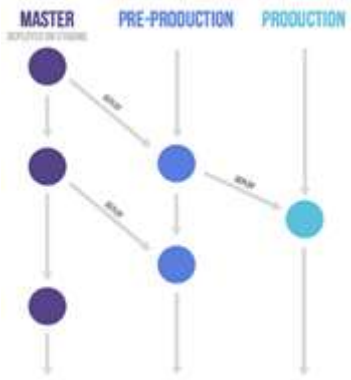

图 4 带环境分支的 GitLab Flow

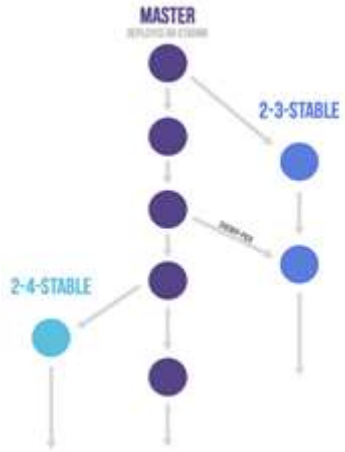

图 5 带发布分支的 GitLab Flow

GitLab Flow 的特性分支合入 master 用的是“Merge Request”，功能与 GitHub Flow 的“pull request”相同，这里不再赘述。

通过 Git Flow、GitHub Flow 和 GitLab Flow（3 个衍生类别） 这几个具体模型的介绍，我给你总结一下特性分支开发的优缺点。如表 3 所示。

优点	缺点
1. 不同功能可以在独立的分支上做开发，消除了功能稳定前彼此干扰的问题。 2. 容易保证主干分支的质量：只要不把没开发好的特性分支合入主干分支，那么主干分支就不会带上有问题的功能。	1. 如果不及时做merge，那么把特性分支合到主干分支会比较麻烦。 2. 如果要做CI/CD，需要对不同分支配备不同的构建环境。

表 3 特性分支开发的优缺点

选出最适合的分支策略

上面我跟你讲到的分支模型，都是 IT 研发领域比较流行的。虽然有些策略带上了代码平台的标识，如 GitHub Flow，但并不意味着该策略仅限于 GitHub 代码平台使用，你完全可以在自己搭建的代码平台上使用这些策略。

接下来，我就总体归纳一下什么情况下应该选择什么样的分支策略。如表 4 所示。

序号	情况	适合的分支策略
1	开发团队系统设计和开发能力强 有一套有效的特性切换的实施机制，保证上线后无需修改代码就能够修改系统行为 需要快速迭代，想获得CI/CD所有好处	主干开发
2	不具备主干开发能力 有预定的发布周期 需要执行严格的发布流程	Git Flow
3	不具备主干开发能力 随时集成随时发布：分支集成后经过代码评审和自动化测试，就可以立即发布的应用	GitHub Flow
4	不具备主干开发能力 无法控制准确的发布时间，但又要求不停集成	GitLab Flow（带生产分支）
5	不具备主干开发能力 需要逐个通过各个测试环境验证	GitLab Flow（带环境分支）
6	不具备主干开发能力 需要对外发布和维护不同版本	GitLab Flow（带发布分支）

表 4 不同情况适用的代码分支策略

国内互联网公司的选择

GitLab 作为最优秀的开源代码平台，被多数互联网大公司（包括阿里、携程和美团点评等）所使用，这些大厂也都采用特性分支开发策略。当然，这些大公司在长期持续交付实践中，会结合各自公司的情况做个性化的定制。

比如，携程公司在 GitHub Flow 的基础上，通过自行研发的集成加速器（Light Merge）和持续交付 Paas 平台，一起完成集成和发布。

再比如，阿里的 AoneFlow，采用的是主干分支、特性分支和发布分支三种分支类型，再加上自行研发的 Aone 协同平台，实现持续交付。

总结

今天，我主要给你介绍了各种代码分支策略的特性。

你应该已经比较清晰地理解了“主干开发”和“特性分支开发”两种策略的各自特性：

1. “主干开发”集成效率高，冲突少，但对团队个人的开发能力有较高要求；
2. “特性分支开发”有利于并行开发，需要一定的流程保证，能保证主干代码质量。

相信在没有绝对自信能力的情况下，面对绝大多数的场景，企业还是会选择“特性分支开发”的策略。所以，我给你介绍了几种主流的特性分支方法，并对比了各类策略的优劣，以及它们适用的场景。

接下来，你就可以根据自己所在项目的具体情况，参考今天的内容，裁剪出最适合自己团队的分支策略了。

思考题

1. 开源性质的项目，为什么不适合用主干开发的分支策略？
2. 如果你所在的团队只有 5 人，而且迭代周期为 1 周，你会采用什么样的分支策略？

欢迎你给我留言。

上一篇　03 | 持续交付和DevOps是一对好基友

下一篇　05 | 手把手教你依赖管理

精选留言 (29)

写留言

康美之心 ...
2018-07-14

12

微服务架构下，比较适合主干开发，一个微服务(含1-4个API)根据复杂性和规模通常由1-3的开发进行开发(含单元测试的开发)，1位测试进行API自动化测试开发，单元测试和API测试都集成到Pipeline上，一旦变动代码提交到master上后，就自动启动Pipeline上的build，在build这里会自动完成覆盖率100%的单元测试，单元测试通过，自动触发FVT deployment，部署成功后，自动触发FVT API自动化测试，FVT测试通过后；自动打出...

展开

作者回复: 讲的好，主干开发除了cherry pick以外，都简单直接

禾子先生
2018-07-13

4

对于开源项目不适合用主干开发。我的理解是更应该采用gitlab flow，为了保证功能稳定性，在贡献者的能力参差不齐和对整体架构和功能不一定完全了解情况下，修改的代码可能会引起其他问题。通过带版本分支的方式发布，稳定的推进和测试新代码的影响。

禾子先生
2018-07-13

3

对于第二个问题，我的思考是使用github flow，虽然追求效率，但必须保证线上版本是一个稳定。大家都在开发分支中进行开发，快速测试和迭代，合并到master时冲突也少。也考虑过使用主干开发，不过但从稳定角度来说，还是选择了前者

作者回复: 很棒

bullboyin...

2

2018-07-18

在自动化测试还没怎么做到位的情况下，为了控制合并到master的都是经过测试的代码，我们增加了一个SIT分支。每次有新特性需要开发时，从master分支check out一个特性分支进行开发，可能涉及到多人协同开发。自测完成后先发起merge request到SIT分支，如果有冲突则不允许merge，以实现不同特性分支间的隔离。如果没冲突且有集成测试人员空闲，则完成merge并安排测试，测试通过后由masters再发起merge request将特性分...

展开

作者回复: 您提到利用SIT分支来保证合并到master的都是经过测试，我们强烈建议使用SmartMerge（本专栏第7讲） 来代替SIT，可以更及时地发现代码集成冲突的问题，其次可以更便捷地决策出用于上线的最大集合。

提到的第一个困惑，我们采用SmartMerge后，对应会有SmartMerge的分支，第一个问题演变为“怎么限制开发不要从SmartMerge的分支创建新分支，而只能从master分支创建新分支？”我认为这个演变过来的问题非常好。可以从两个方面着手来规范。
1）如果在gitlab UI上创建分支的话，我们可以很容易地限制只能从master创建新分支。
2）另一方面我们不打算限制git客户端的行为，仅当git客户端向gitlab服务端push新分支的时候做相应的检查。为了判断新分支是否从master拉取且尚未和其他分支做merge，只需查看该分支和master的merge-base的commit之间，是否存在merge的commit即可，如果有，则不允许push。

如果没有采用SmartMerge的方式，其实方法本质是相同的。如果短期内还是只能通过SIT分支的话，做法和SmartMerge的方式是一样的。
口诀为：在服务端直接控制新分支的创建，且拒绝客户端push不规范的新分支。

第二个困惑，假设你们尚未使用SmartMerge的方式，可以通过在你们的开发流程中额外增加一个活动来搞定。用于集成的SIT分支在编译打包前，务必和master做merge，打包测试后如果没问题的话，SIT和master分支做fast forward的检查，如果是fast forward，那么SIT merge到master分支产生的commit，内容和SIT 的HEAD是相同的，也就无需再一次进行测试了。如果你们master分支的变更只能来自SIT分支，那么这个fast forward是很容易得到保证的。

纳米

2018-07-12

2

您好 我是一个非开发的人员。弱弱问下，最后表里总结的生产分支 环境分支中，开发人员自身是check out哪个分支代码出来开发 又是往哪个分支集成呢。能否在这两个分支上帮细化下。非常感谢。

思考题2 我理解 既然人很少 迭代周期也较为宽松 可能这一周大部分人都工作在一个功能...

展开

作者回复: 特性分支模式下，都是从production拉取的开发分支，然后合并到master，master做持续集成，为了不影响持续集成，所以有了从master拉出来的环境分支进行部署

要考虑团队人员的能力，如果新人较多，就使用特性分支，反之使用主干开发，github flow的pull request其实也是特性分支

有道测试组
2018-12-26

1

选用什么样的开发方式，也有一定是基于历史遗留原因或者大家的编程习惯~
开源软件不适合主干开发是说，我们会假设有若干贡献者，并且贡献者的水平参差不齐，会带来一粒老鼠屎，坏了一锅粥的问题。 但如果我们的pr 提交到master分支，会经过完善的自动化测试（包括各种配置，各种机型，我们将要做的所有测试）， 测试通过才让合入。这样其实主干开发流程简洁，能加快开发效率。 ...
展开

大M
2018-07-17

1

如果有不同环境呢，比如灰度环境，这样情况下我觉得是gitflow 比较合适。
展开

董永刚
2018-07-17

1

分支类型中，带生产分支，带发布分支，带产品分支的分别是什么样的场景呢?

作者回复: gitlab几个flow，除了用于集成的master分支外，额外还配置了其他用于测试或上生产的分支。

拿带生产的分支来说，如果公司约定每周四才能发布到生产，而团队于周二在master上就集成好了V3.6版本，为了不影响后面的集成，master分支需要先merge到PRODUCTION分支，周四上线只需取PRODUCTION分支即可。

当然，gitlab flow也只是提供一个解决方案而已，只要能在发布窗口时正确快捷地找到用于发布的commit，同时又不影响集成即可。

至于提供带环境分支的，就是用分支模型来规约发布流程，保证用于上线的代码经过层层测试。做法就是约好了只能用PRODUCTION分支上线。比如通过PRE-PRODUCTION环境测试后，才能把该环境对应的分支合入到PRODUCTION分支。

小胖胖
2018-07-13

1

gogs被gitlab 应该选择哪个？有什么区别吗

展开

作者回复: 主要还是看你能hold住哪个，比如我个人属于rubyist，所以用ruby写的gitlab肯定就是我的首选了，如果你比较偏爱或擅长go，那就gogs了。

相对来说我个人觉得gogs比较轻量化，有好处有坏处，比如你要大量二次开发，gitlab service形式就比较友好，而且最近也开源了HA方案

徐卫
2018-07-12

1

你好，问下。老师帮我看下理解对不对。主干开发是否只有一个分支，开发的代码提交到这个分支，发布也是用此分支。文中讲到的特性切换怎么做的？ 我个人理解是在那个发布的版本上打标签，特性切换是从tag上拉出一份代码部署？

作者回复: 主干开发也可以有多个分支，除了master，其他是发布分支，master是用来持续集成的，也就是大家只往master push代码

特性切换不是分支，是代码或框架实现的功能，就理解为功能开关好了，也就是说有些代码即使上线了，也能通过开关保证它不工作

云学
2018-07-12

1

到底用哪种分支策略和团队的业务分工有关，如果修改的代码交织严重，肯定是主干，如果不严重，可以主干可以分支

小胡子
2018-07-12

1

我们团队主干和开发两个分支并过来并过去，我该怎么解释这是种什么不好的方式呢。。。

作者回复: 持续交付要求至少有一条分支随时能够进行发布，只要遵循这个原则，仅2条分支并无大碍

白天不懂爷...
2018-07-12

1

老师，您好，最近公司想用gitlab做配置中心，请问贵公司gitlab的高可用是怎么做的呢？谢谢

作者回复: 最新版本应该给了和GitHub类似的解决方案，我们是对gitlab做了分片，每个分片是一个仓库，在集群之前增加了基于nodejs ssh2修改的proxy，仓库做简单的定时备份

春之绿野
2019-06-04

因为在开源软件上，个人开发的话每个人的质量无法保证，所以不能用主干开发。第二种情况用主干开发，因为人数少，特性分支的话迭代周期太长

飞机翅膀上
2019-05-01

老师好，我们现在一个项目具有很多功能，还没有做微服务，甚至soa还没有做，产品在迅速迭代，目前一个master分支，一个test，和十几个特性分支，特性分支需要合并到test测试通过才可以将特性分支合到master发布
由于特性分支太多导致合到test分支经常有冲突，甚至编译不通过。所以每天都要拉取master分支到特性分支做合并，尽管如此，还是会有冲突...

展开

llgeek
2019-03-22

有两个疑问请老师帮忙解答

如果采用GitHub flow，一个长期的feature分支，需要如何跟master同步？
feature分支提交pr后，测试完成，在分支发布或者合并到master在主干发布，如何评析优劣？
谢谢老师

展开

木有昵称
2019-03-02

我们公司项目多，根据项目大小程度，基本上特别小的项目一个人开发直接就一个master分支；普通的项目是一个master分支加一个develop分支，多个开发人员使用develop分支进行开发，自行提交解决冲突，测试环境随时使用develop分支发布上线，生产线需要上线时由开发负责人专人负责合并到master上进行发布。

展开

Bruce晓勇
2019-01-15

1如果直接在特性分支交付，如何保证项目组代码合并到master?
2在特性分支上做持续交付，对管理上比较难统计每个系统的持续交付数据，不知怎么考虑，因为不知道哪个分支是哪个系统的

展开

kursk.ye
2019-01-05

特性切换（feature toggle）翻译得不准确，叫 功能开关更合适

展开

smartzs
2018-12-11

老师，主干开发 和 带发布分支模型 很像啊，带发布分支约等于主干开发了吧？

作者回复: 还是有一点区别的，比如可以暂时预设多个发布分支，以针对一段时间内的多版本并行需要