黑客派

成熟的 Git 分支模型

作者：LieBrother

原文链接：https://hacpai.com/article/1545227068112

来源网站：黑客派

<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="https://ig.hacpai.com/bing/20180321.jpg?imageView2/1/w/960/h/520/interlace/1/q/100">
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<blockquote>
 今天介绍一下工作中会用到的 Git 分支模型。
</blockquote>
先贴上图以表敬意

<h3 id="闲言">闲言</h3>
在学校不管是自己写课程设计还是给老师做项目，有 2 到 3 个人一起协作开发时就会使用 Git 但是只是简单用了它所提供的代码协作功能，在学校的项目，比如课程设计，开发完老师检查完就没维护了，给老师做项目也是，基于项目的特征：没有持久性、一次性开发，所以没有应到 Git 分支模。在企业中，一个应用往往是有比较长的生命线，由很多个迭代项目开发构成，这时要解决几十甚至百人的代码协作问题，就需要一套完整的规范的代码开发流程。
我还记得当初大四的时候，去了一家企业实习，当时小团队只有 3 个开发人员，Git 使用没有规，只有一个 master 主分支，项目也没有管理规范，来一个需求点就做。当时经常出现代码覆盖，各代码合并，线上代码也不知道是哪个节点的代码。。。到我走的时候，也没使用上这个分支模型。毕后入职了某银行，不说分支模型了，Git 都没用上，直到今年跳槽到互联网公司才了解到这个分支模。因此，你工作不一定会真正用到这个分支模型，如果是在互联网企业，很有可能会使用上。
有些小伙伴看到这张偌大的图觉得有些晕，很认真地说，这是一张大家都在用的图，特别是互联企业。如果是还没有工作的小伙伴，可能有些陌生，没事，我们来看一下这些内容。
<h3 id="分支介绍">分支介绍</h3>
master ： 这个分支的代码是发布到生产的代码
 develop ： 这个分支的代码是预发布到生产的代码
 release ： 这个分支的码是新版本发布到生产的代码
 feature ： 这个分支的代码是新需求开发的码
 hotfix ： 这个分支的代码是紧急修复生产 bug 的代码
<h3 id="场景设想">场景设想</h3>
下面列举一些可能你在工作中会经常面对的场景

  组长分配新需求下来，安排下周上线（假设是 1227 号），你看看当前有没有下周版本的支？有的话很简单，checkout 下周分支（feature_app1.1.0_1227）来开发就行，没有的话这时需要建分支，从 develop 分支创建新的 feature 分支（feature_app1.1.0_1227），然后将对应的 pom.xml 版本号修改成 1.1.0-SNAPSHOT，注意命名，比如这里我用 feature 做前缀，你也可以自己设定一规则。
  开发完 feature_app1.1.0_1227 需求，移交了测试，很遗憾，测试出现了 n 个 bug，这依旧在 feature_app1.1.0_1227 上修复 bug。
  终于到了发版前一天，测试 MM 说 n 轮测试完了，没问题，拉上线版本，再做一次回归试。这时，你就需要把 feature_app1.1.0_1227 分支合并到 develop 分支，然后从 develop 分支中建新的分支 release_app1.1.0_1227，然后修改对应的版本号为 1.1.0-RELEASE。
  到了发版日早上了，测试 MM 用了 release_app1.1.0_1227 版本测试了一番，又发现了个 bug。别慌，只要不是生产的 bug，都好解决。这时你要在 release_app1.1.0_1227 修复 bug，记不能在 feature_app1.1.0_1227 上修改，feature_app1.1.0_1227 分支已经没有多大作用了，只用看代码提交记录。
  安安全全的到了晚上，开始发版了，发完版突然发现了有异常，定位问题后发现是有一行

码写错了，跟组长确认后，在 release_app1.1.0_1227 分支上做了修改，重新打包后发版，验证了一时间，没问题了。。。
  发版总算完成了，这时，别忘记把 release_app1.1.0_1227 版本合并到 develop 和 master 分支。还有一点很重要的，把 develop 分支代码合并到 1227 以后的版本（如果已经有 1227 以后的本的话）。注意：这个步骤合并代码要谨慎，如果有别人的代码合并冲突比较大，需要找那个开发的事一起合并代码。总算可以睡个好觉了。。。
  告别了旧需求，迎来了新需求，接下来的需求开发就按上面的步骤走。。。
  第二天，突然生产上一直报 NullPointerException，定位发现是一行代码没有判空导致的三番确认，原来这个数据以前是不为空的，现在确实需要支持有些数据为空的，需要紧急修复这个 bu，和组长确认之后，从 master 分支上拉了一个 hotfix_app1.1.1_1228 分支代码，修复了 NullPointe Exception，打包后上线，验证没问题后，把 hotfix_app1.1.1_1228 分支合并到 develop 和 master 分支，并把 develop 分支合并到 1227 以后的版本。

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr pt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in >
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
好了，一大坨的文字描述了基于分支模型开发的过程。不同公司在应用过程中可能会有些微小的同，但是整体流程都是差不多的。比如有的公司可能会把 release 合并到 master 后，用 master 代发布到生产，发版当时有异常，再从 master 分支上拉 hotfix 分支进行修复。上面描述的步骤就不一了，发版时出现异常，直接在 release 上修复。这些小的差别就不用计较太多啦。
希望本文能够让你认识到有这么一个标准的 Git 分支模型，在不管工作上还是学习上，在需要分管理的时候，回忆起有这么一个图，根据你的场景再应用进去，肯定会少走很多弯路。