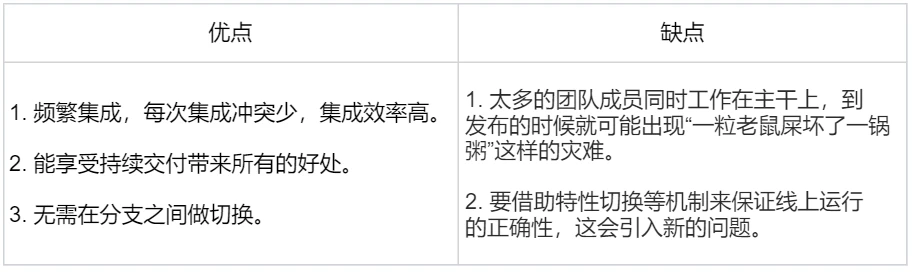
**代码分支策略**

**主干开发**

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。“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。

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。



**特性分支开发**

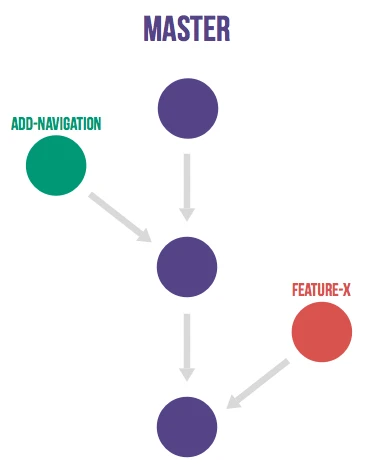
**Git Flow**

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。

**GitHub Flow**

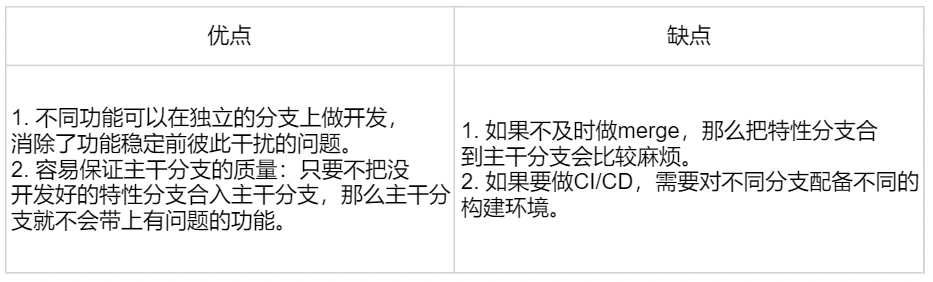
GitHub Flow 是 GitHub 所使用的一种简单流程。该流程只使用 master 和特性分支，并借助 GitHub 的 pull request 功能。



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。

**GitLab Flow**





**选出最适合的分支策略**



**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。

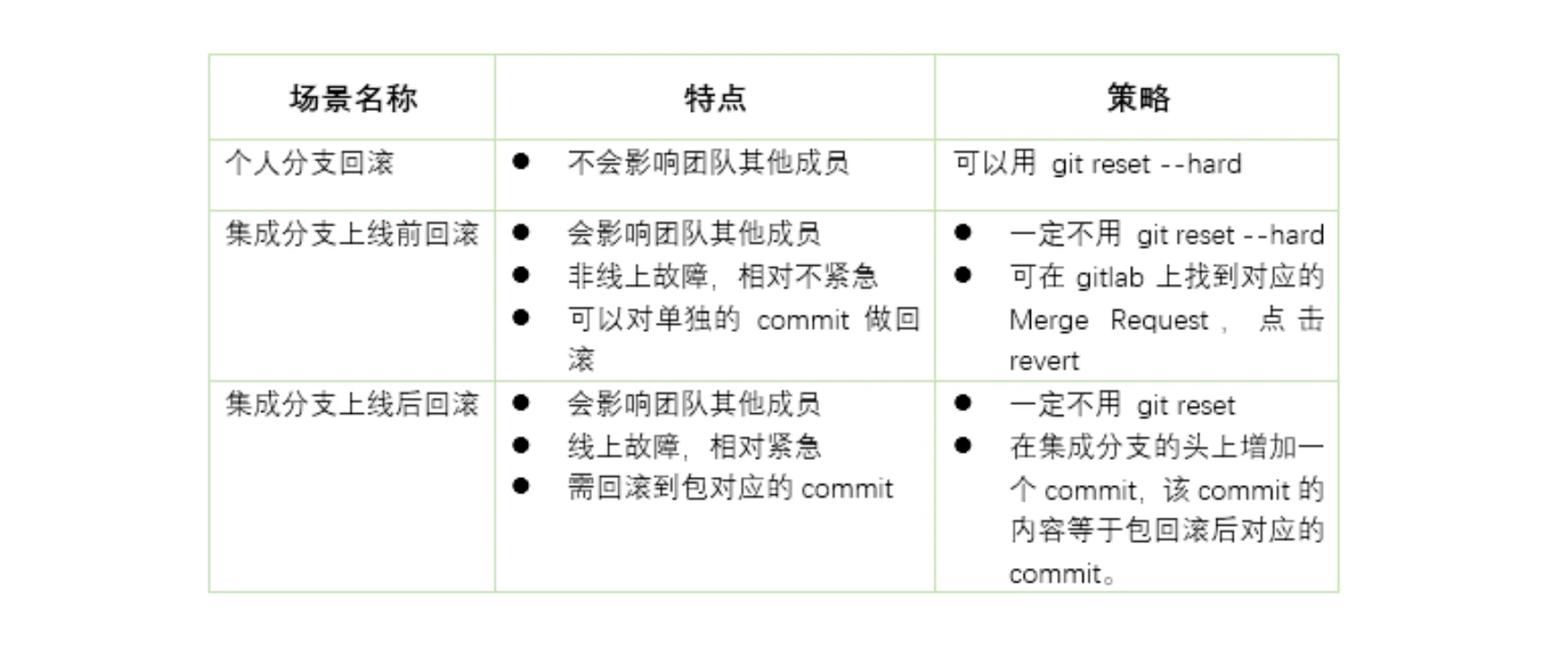
再比如，阿里的 AoneFlow，采用的是主干分支、特性分支和发布分支三种分支类型，再加上自行研发的 Aone 协同平台，实现持续交付。

我们使用的是带发布分支的策略，定版后release 分支不再合入新功能，只能合入bugfix。

**代码回滚**

代码回滚必须遵循的原则

集成分支上的代码回滚坚决不用 reset --hard 的方式，原因如下：集成分支上的 commit 都是项目阶段性的成果，即使最近的发布不需要某些 commit 的功能，但仍然需要保留这些 commit ，以备后续之需。开发人员会基于集成分支上的 commit 拉取新分支，如果集成分支采用 reset 的方式清除了该 commit ，下次开发人员把新分支合并回集成分支时，又会把被清除的 commit 申请合入，很可能导致不需要的功能再次被引入到集成分支。



第一、开发人员独立使用的分支上，如果最近产生的 commit 都没有价值，应该废弃掉，此时就需要把代码回滚到以前的版本。

git checkout feature-x

git reset --hard C3的HASH值

git push -f origin feature-x

第二、代码集成到团队的集成分支且尚未发布，但在后续测试中发现这部分代码有问题，且一时半会儿解决不掉，为了不把问题传递给下次的集成，此时就需要把有问题的代码从集成分支中回滚掉。

通过gitlab 页面上的revert 操作

第三、代码已经发布到线上，线上包回滚后发现是新上线的代码引起的问题，且需要一段时间修复，此时又有其他功能需要上线，那么主干分支必须把代码回滚到产品包 V0529 对应的 commit。

$ git fetch origin

$ git checkout master

$ git reset --hard V0529 # 把本地的master 分支的指针回退到 V0529，此时暂存区(index)里就指向 V0529里的内容了。

$ git reset --soft origin/master # --soft使得本地的master 分支的指针重新回到 V05javascript:;30，而暂存区(index)变成 V0529的内容。

$ git commit -m "rollback to V0529" # 把暂存区里的内容提交，这样一来新生成的commit的内容和 V0529 相同。

$ git push origin master # 远端的master也被回滚。