链滴

Git 使用规范

作者：sucre

原文链接：https://ld246.com/article/1588997187627

来源网站：链滴

上午说了网盘可以用SVN，那么代码其实也可以在SVN上管理，但为什么我们不用SVN呢，很大程度是因为svn对分支的支持不如git好，比如方便性、灵活性等等。

GIT VS SVN

	GIT	SVN
优点	• 适合多人开发 • 公共服务器压力和数据量都不会太大 • 分支操作灵活 • 方便解决冲突 • 离线工作 • 利用hook便于开发实现CI/CD	• 方便管理 • 集中式服务器易于管理 • 代码一致性高 • 适合开发人员不多
缺点	• 学习周期较长 • 代码保密性差	• 服务压力大 • 无网状态不能工作 • 多版本管理功能较弱

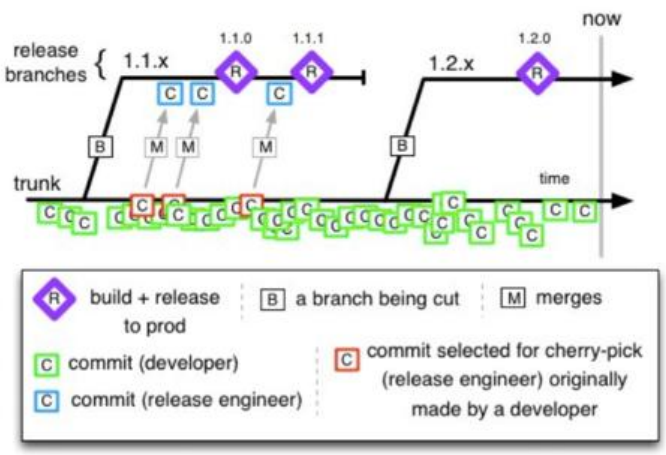
这里做了一个git和SVN的对比，都很明显，不过有一点，git代码保密性差的意思应该是，一旦clone就公开了所有代码、分支和版本信息，

SVN不一样，svn在分支管理时，多个分支是在不同的文件夹下；

然后我们介绍下git的分支管理模型，也是我们后面开发时候需要的一些流程和规范，主要介绍三种分管理模型，最后还有我们目前使用的一种综合三类模型优点的模式；

- TrunkBased
- GitFlow
- AoneFlow

GIT分支管理——TrunkBased



优点

- 避免多分支合并的困扰
- 随时可发布

缺点

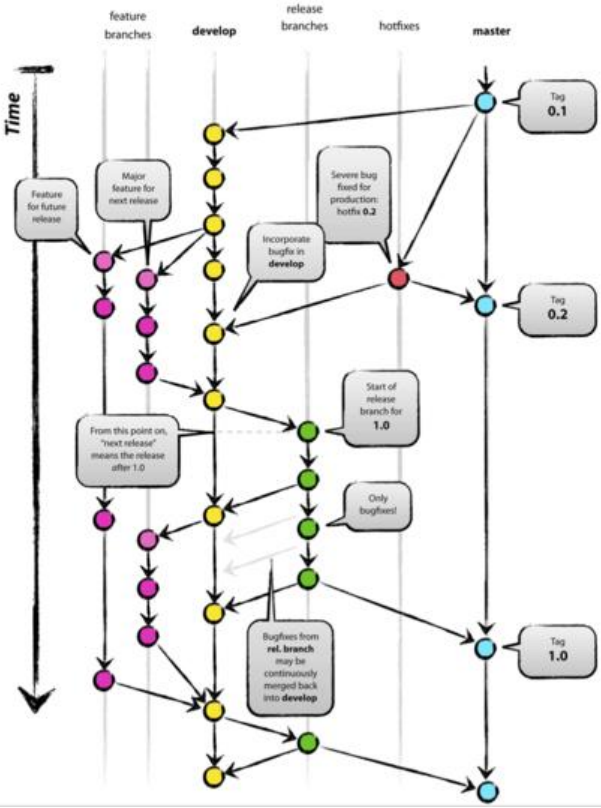
- 无法避免发布未完成的feature

首先说一下TrunkBased，这个就比较简单了，只有一个master分支，个人过程太简单，只适合项目开始的时候，赶紧开发出一个版本，以后再换其他的分支管理方式。

缺点很明显，没有测试分支，只能自己本地测试，有时候自己测试难免有忽略的点。

或者我有个功能feature1直接合并到master分支，让测试团队去测试，但是另一个feature2要上线，把没有测试通过的feature1给发布出去了。

GIT分支管理——GitFlow



主要分支

- master
- develop

协助分支

- feature
- release
- hotfix

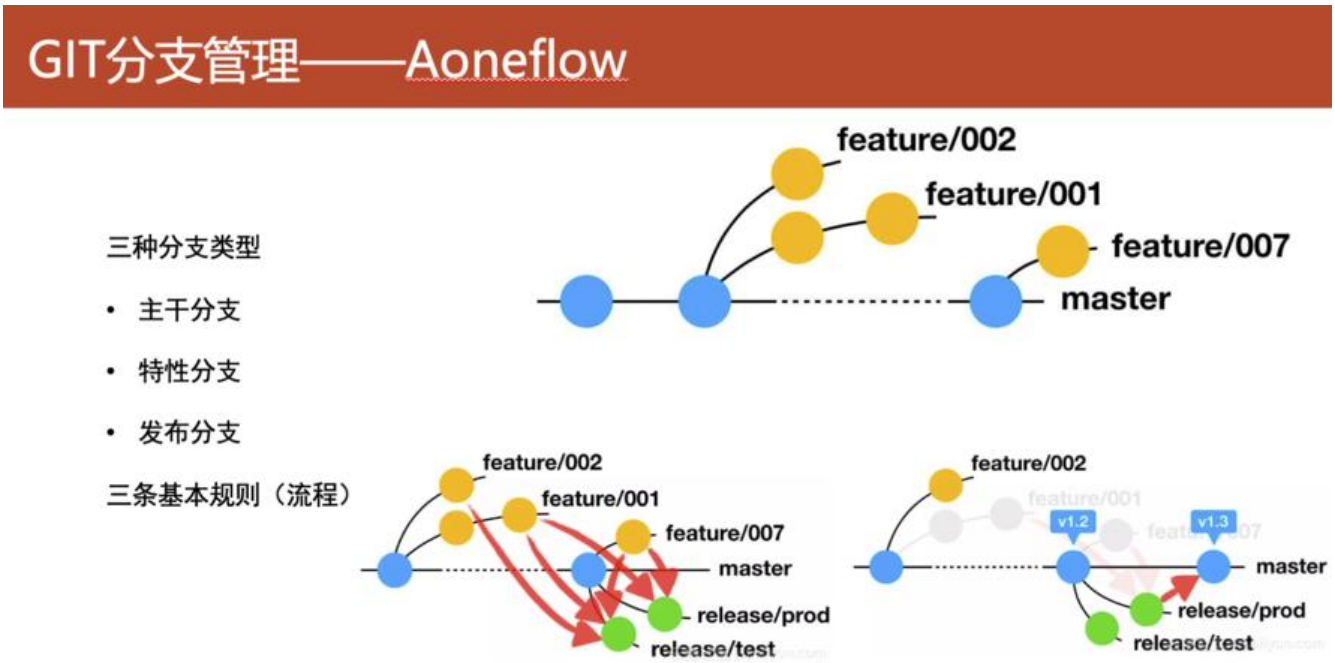
原文链接：Git 使用规范

接下来是gitflow，特点：引入feature分支，避免对master的污染，且多个新特性同时开发，不会互影响；

但同时带来一个难题，就是feature的颗粒度设计，如果颗粒较大，比如模块级，会导致feature分支期存在，等到feature完成，进行代码合并的时候，容易出现大量的代码冲突，

如果颗粒较小，又会存在大量的feature分支，维护成本高，开发来回切换分支，搞得晕头转向，把码写到错误的分支也会发生；

主要分支可以看做为基准，就是代码的修改都在我的基础之上，最后所有的代码改动直接反馈到主要支上。



然后是AoneFlow，在 AoneFlow 上你能看到许多其他分支模式的影子。它基本上兼顾了 TrunkBased 的“易于持续集成”和 GitFlow 的“易于管理需求”特点，同时规避掉 GitFlow 的多分支的麻烦。

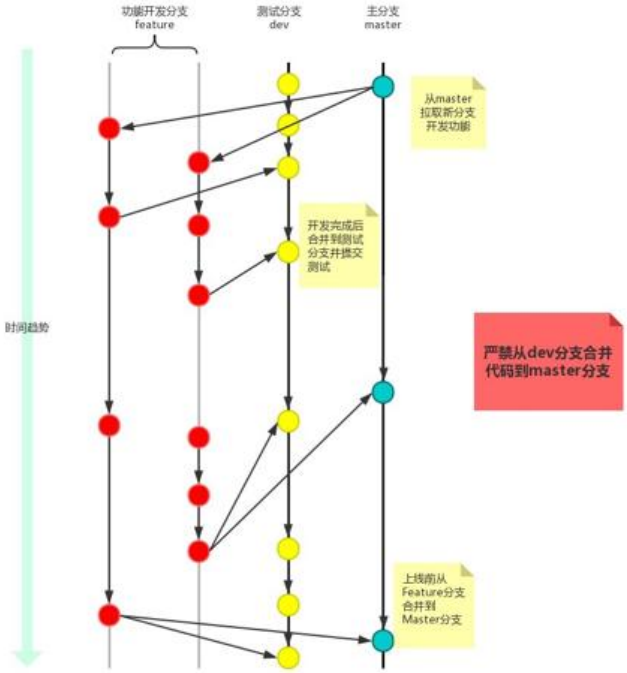
三条基本规则：

-开始工作前，从主干创建特性分支

● 通过合并特性分支，形成发布分支。

● 发布到线上正式环境后，合并相应的发布分支到主干，在主干添加标签，同时删除该发布分支关联特性分支。

最后介绍一下我们目前的git管理方式，比较简单；



首先说一下我们在开发时，一般会创建3个分支：开发、测试、master分支。

一般我们在开发功能、修复线上bug都会创建一个新的分支，都算成是开发分支。

我们在gitlab上创建项目时，默认就是master分支，就是我们的线上发布分支。

项目初始化完成后，我们开始项目功能的开发，就需要在此基础上发起一个功能开发的分支，我们可叫做开发分支，

命名可以以jira号命名，比如：jira-22;

当功能开发完成后，合并到测试分支test；

测试通过，合并到master分支上，我们也可以在线提PR，由leader进行代码review，然后手动合并。

提交代码时，我们可以在message里指定jira号，比如：git commit -m “完善xx功能 #jira-22”；

GIT使用规范

- master 是什么?
* Anything in the master branch is deployable
* 禁止直接提交到 master 分支

- 稳定分支
* master 是稳定分支
* 其他分支 都是不稳定分支

- 在分支上做什么
* 一个分支做且只做一件事 (包括但不限于功能开发 修复Bug 整合 发布 紧急修复 等)
* 如果要做多件事，开多个分支
* 尽量避免多个人在同一分支上做事

- 分支生命周期
* 如果做完一件事，分支的生命周期就结束了，要删掉这个分支
* 分支的生命周期，越短越好，不要长期持有一个分支

- 从哪里来，到哪里去
* 分支的上游只能是 master
* 分支的下游只能是 release

- 如何上线
* 测试的版本就是上线的版本，上线的版本必须经过测试

原文链接：Git 使用规范