密级：公开

对酒当歌 Git Flow 分支管理策略

深圳市对酒当歌电子商务有限公司

目　录

1. 前言

如果你严肃对待编程，就必定会使用”版本管理系统”（Version Control System）。

眼下最流行的”版本管理系统”，非 Git 莫属。

相比同类软件，Git 有很多优点。其中很显著的一点，就是版本的分支（branch）和合并（merge）十分方便。有些传统的版本管理软件，分支操作实际上会生成一份现有代码的物理拷贝，而 Git 只生成一个指向当前版本（又称”快照”）的指针，因此非常快捷易用。

但是，太方便了也会产生副作用。如果你不加注意，很可能会留下一个枝节蔓生、四处开放的版本库，到处都是分支，完全看不出主干发展的脉络。

同时针对目前对酒当歌开发团队使用 Git 并没有统一的分支管理策略，所以编写该文档为后续新员工培训、代码管理、自动化发布提供标准。

Git flow 工作流借鉴自 Vincent Driessen 提出了一个分支管理的策略（中文简译版），Gitflow 工作流定义了一个围绕项目发布的严格分支模型。虽然比功能分支工作流复杂几分，但提供了用于一个健壮的用于管理大型项目的框架。

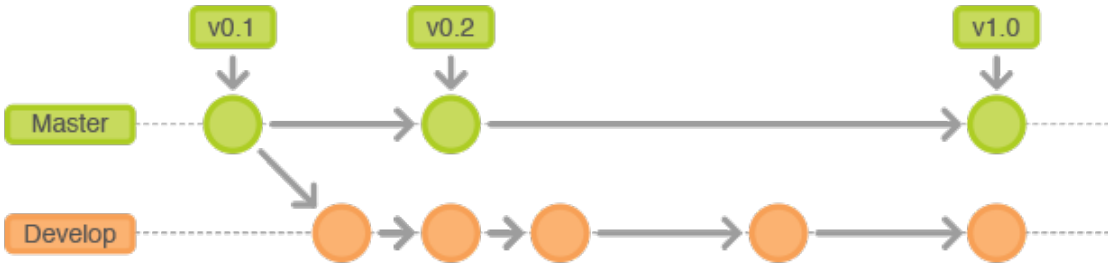
Gitflow 工作流没有用超出功能分支工作流的概念和命令，而是为不同的分支分配一个很明确的角色，并定义分支之间如何和什么时候进行交互。除了使用功能分支，在做准备、维护和记录发布也使用各自的分支。当然你可以用上功能分支工作流所有的好处：Pull Requests、隔离实验性开发和更高效的协作。

理论上，这些策略对所有的版本管理系统都适用，Git 只是用来举例而已。如果你不熟悉 Git，跳过举例部分就可以了。

2. 分支管理

一、历史分支

相对使用仅有的一个 master 分支，Git flow 工作流使用 2 个分支来记录项目的历史。master 分支存储了正式发布的历史，而 develop 分支作为功能的集成分支。这样也方便 master 分支上的所有提交分配一个版本号。



剩下要说明的问题围绕着这 2 个分支的区别展开。

二、功能分支

每个新功能位于一个自己的分支，这样可以 push 到中央仓库以备份和协作。但功能分支不是从 master 分支上拉出新分支，而是使用 develop 分支作为父分支。当新功能完成时，合并回 develop 分支。新功能提交应该从不直接与 master 分支交互。

注意，从各种含义和目的上来看，功能分支加上 develop 分支就是功能分支工作流的用法。但 Gitflow 工作流没有在这里止步。

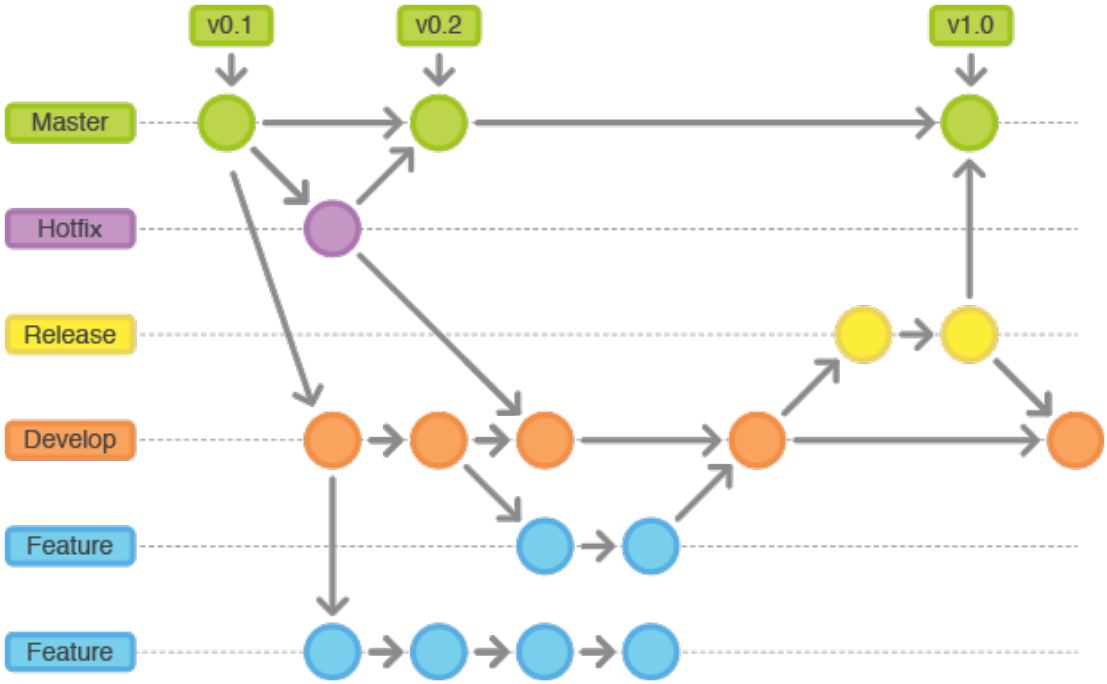
三、发布分支

一旦 develop 分支上有了做一次发布（或者说快到了既定的发布日）的足够功能，就从 develop 分支上 fork 一个发布分支。新建的分支用于开始发布循环，所以从这个时间点开始之后新的功能不能再加到这个分支上 —— 这个分支只应该做 Bug 修复、文档生成和其它面向发布任务。一旦对外发布的工作都完成了，发布分支合并到 master 分支并分配一个版本号打好 Tag。另外，这些从新建发布分支以来的做的修改要合并回 develop 分支。

使用一个用于发布准备的专门分支，使得一个团队可以在完善当前的发布版本的同时，另一个团队可以继续开发下个版本的功能。 这也打造定义良好的开发阶段（比如，可以很轻松地说，『这周我们要做准备发布版本 4.0』，并且在仓库的目录结构中可以实际看到）。

常用的分支约定：
用于新建发布分支的分支: develop
用于合并的分支: master
分支命名: release-* 或 release/*

四、维护分支



维护分支或说是热修复（hotfix）分支用于生成快速给产品发布版本（production releases）打补丁，这是唯一可以直接从 master 分支 fork 出来的分支。修复完成，修改应该马上合并回 master 分支和 develop 分支（当前的发布分支），master 分支应该用新的版本号打好 Tag。

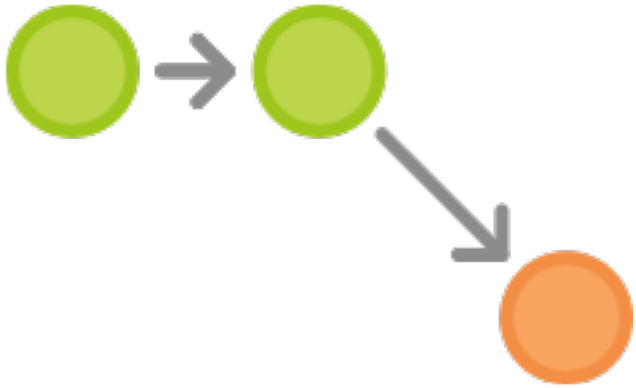
为 Bug 修复使用专门分支，让团队可以处理掉问题而不用打断其它工作或是等待下一个发布循环。你可以把维护分支想成是一个直接在 master 分支上处理的临时发布。

常用的分支约定：
用于新建维护分支的分支: master
用于合并的分支: master && develop
分支命名: hotfix -* 或 hotfix /*

3. 快速开始

下面的示例演示本工作流如何用于管理单个发布循环。假设你已经创建了一个中央仓库。

一、创建开发分支



第一步为 master 分支配套一个 develop 分支。简单来做可以本地创建一个空的 develop 分支，push 到服务器上：

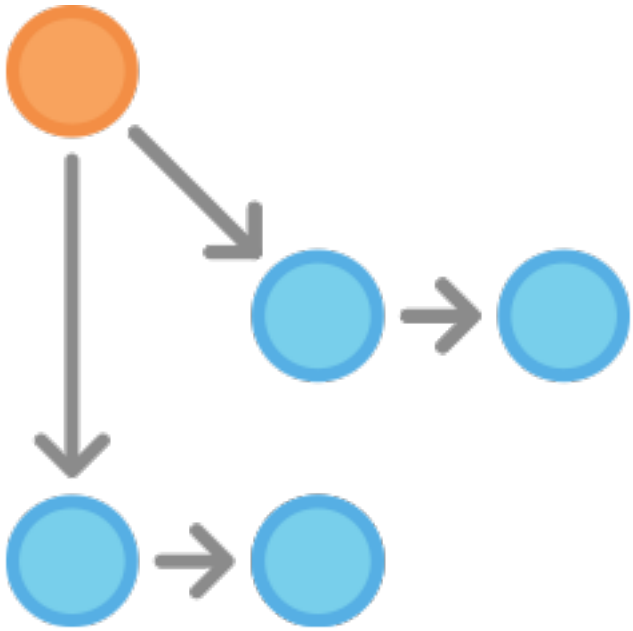
```
git branch develop
git push -u origin develop
```

以后这个分支将会包含了项目的全部历史，而 master 分支将只包含了部分历史。其它开发者这时应该克隆中央仓库，建好 develop 分支的跟踪分支：

```
git clone ssh://user@host/path/to/repo.git
git checkout -b develop origin/develop
```

现在每个开发都有了这些历史分支的本地拷贝。

二、小红和小明开始开发新功能



这个示例中，小红和小明开始各自的功能开发。他们需要为各自的功能创建相应的分支。新分支不是基于 master 分支，而是应该基于 develop 分支：

```
git checkout -b some-feature develop
```

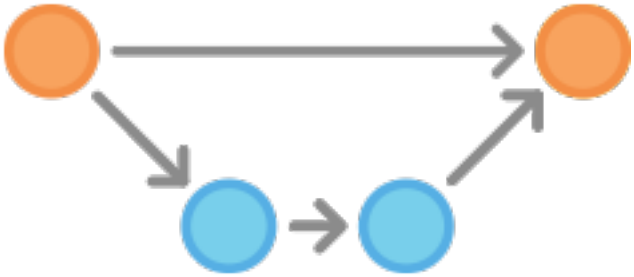
他们用老套路添加提交到各自功能分支上：编辑、暂存、提交：

```
 git status
```

```
 git add
```

```
git commit
```

三、小红完成功能开发



添加了提交后，小红觉得她的功能OK了。如果团队使用Pull Requests，这时候可以发起一个用于合并到develop分支。否则她可以直接合并到她本地的develop分支后push到中央仓库：

git pull origin develop

git checkout develop

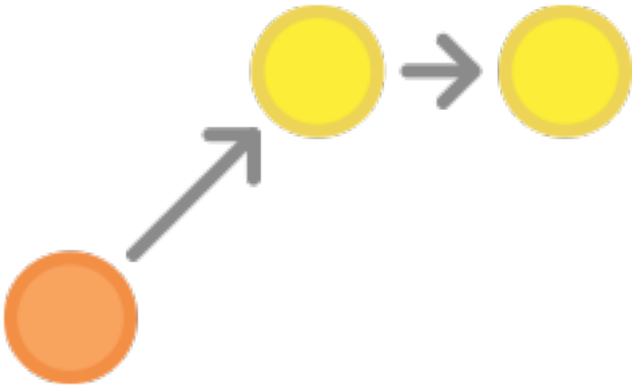
git merge some-feature

git push

git branch -d some-feature

第一条命令在合并功能前确保develop分支是最新的。注意，功能决不应该直接合并到master分支。冲突解决方法和集中式工作流一样。

四、小红开始准备发布



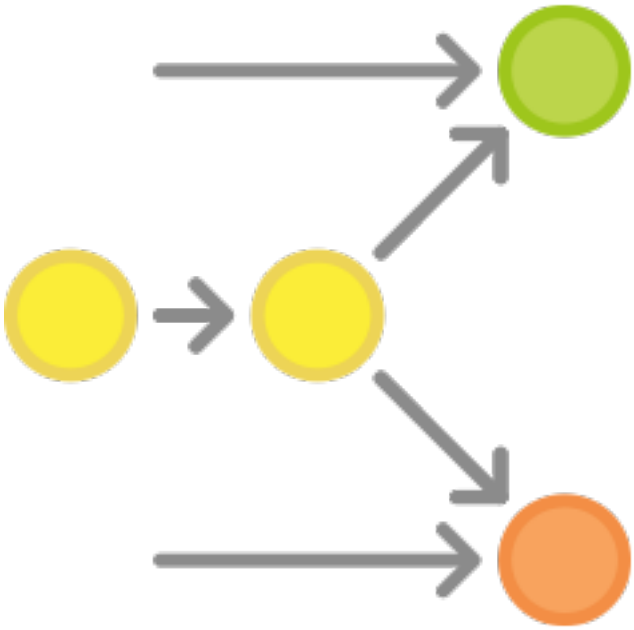
这个时候小明正在实现他的功能，小红开始准备她的第一个项目正式发布。像功能开发一样，她用一个新的分支来做发布准备。这一步也确定了发布的版本号：

git checkout -b release/v0.1.0 develop

这个分支是清理发布、执行所有测试、更新文档和其它为下个发布做准备操作的地方，像是一个专门用于改善发布的功能分支。

只要小红创建这个分支并 push 到中央仓库，这个发布就是功能冻结的。任何不在 develop 分支中的新功能都推到下个发布循环中。

五、小红完成发布



一旦准备好了对外发布，小红合并修改到master分支和develop分支上，删除发布分支。合并回develop分支很重要，因为在发布分支中已经提交的更新需要在后面的新功能中也要是可用的。另外，如果小红的团队要求Code Review，这是一个发起Pull Request的理想时机。

git checkout master

git merge release/0.1.0

git push

git checkout develop

git merge release/0.1.0

git push

git branch -d release/0.1.0

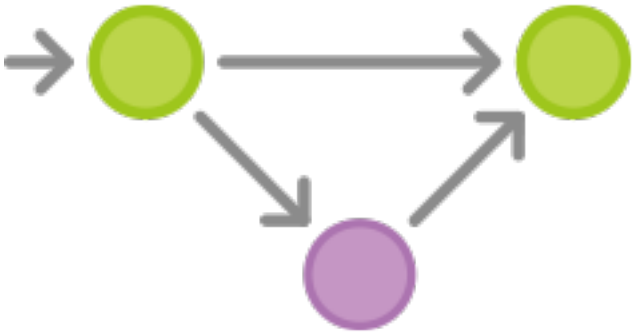
发布分支是作为功能开发（develop分支）和对外发布（master分支）间的缓冲。只要有合并到master分支，就应该打好Tag以方便跟踪。

git tag -a v0.1.0 -m "Initial public release" master

git push --tags

Git有提供各种勾子（hook），即仓库有事件发生时触发执行的脚本。可以配置一个勾子，在你push中央仓库的master分支时，自动构建好对外发布。

六、最终用户发现 Bug



对外发布后，小红回去和小明一起做下个发布的新功能开发，直到有最终用户开了一个Ticket抱怨当前版本的一个Bug。

为了处理Bug，小红（或小明）从master分支上拉出了一个维护分支，提交修改以解决问题，然后直接合并回master分支：

git checkout -b hotfix/0.1.1 master

Fix the bug

git checkout master

git merge hotfix/0.1.1

git push

就像发布分支，维护分支中新加这些重要修改需要包含到develop分支中，所以小红要执行一个合并操作。然后就可以安全地删除这个分支了：

git checkout develop

git merge hotfix/0.1.1

git push

git branch -d hotfix/0.1.1

4. 项目部署

以上章节，描述了基于 Git flow 工作流的分支管理策略，部署环节我们也同样沿用 Git flow 的分支策略进行部署。

origin/develop 开发环境
origin/release/* origin/hotfix/* 预发布环境
origin/master 线上环境

5. 引用

http://blog.jobbole.com/23398/ 阮一峰：Git 分支管理策略

http://nvie.com/posts/a-successful-git-branching-model/ A successful Git branching model

https://www.sourcetreeapp.com/ sourcetree 分支管理工具

http://semver.org/lang/zh-CN/ 语义化版本 2.0.0