2. Git 工作流

1 引言

本文档目的是通过规范化的流程，使得产品、开发与测试等各职能人员能更高效的协同工作，通过规范化的流程使得产品高效稳定运行。

在多组员，多项目等环境进行协同工作时，如果没有统一规范、统一流程，则会导致额外的工作量，甚至会做无用功。所以要减少版本冲突，减轻不必要的工作，就需要规范化的工作流程。

本文档旨在介绍如何用 Git 管理项目代码，并不是 Git 的使用教程。Bitbucket 的协同工作指南参考这里。

2 权限管理

Git 的授权思路：

- 公司内部代码开放。即代码在公司内部，对项目组成员一视同仁的开放。
- 公司对代码库进行合理分解，对每个代码库分别授权。即某个代码库对团队成员完全开放，对其他团队完全封闭。

一个项目（Project）分为多个库（Repository），每个库可以设置权限仅允许特定用户访问。多个库之间通过外部引用的方式链接到一起，并通过 lib 的方式隔离源代码。

对于每个库的用户而言，可以读取库的所有代码。库的不同分支可以对用户设置不同的写权限。比如 master 分支仅允许资深软件工程师进行 merge、push 之类的写操作，而 develop 分支允许普通的软件工程师进行 push 之类的写操作。

3 Git 工作流

工作流是一个更高级的主体，是针对针对项目流程管理和开发协同约定而言。关于工作流的介绍可以参考这里，相应的翻译参考这里。本文约定在本项目中采用的工作流为 Gitflow，Sourcetree 原生支持 git-flow 工具扩展。参考了这篇简书的内容编写。

3.1 总则

- 统一使用 Git 作为版本控制的主要工具

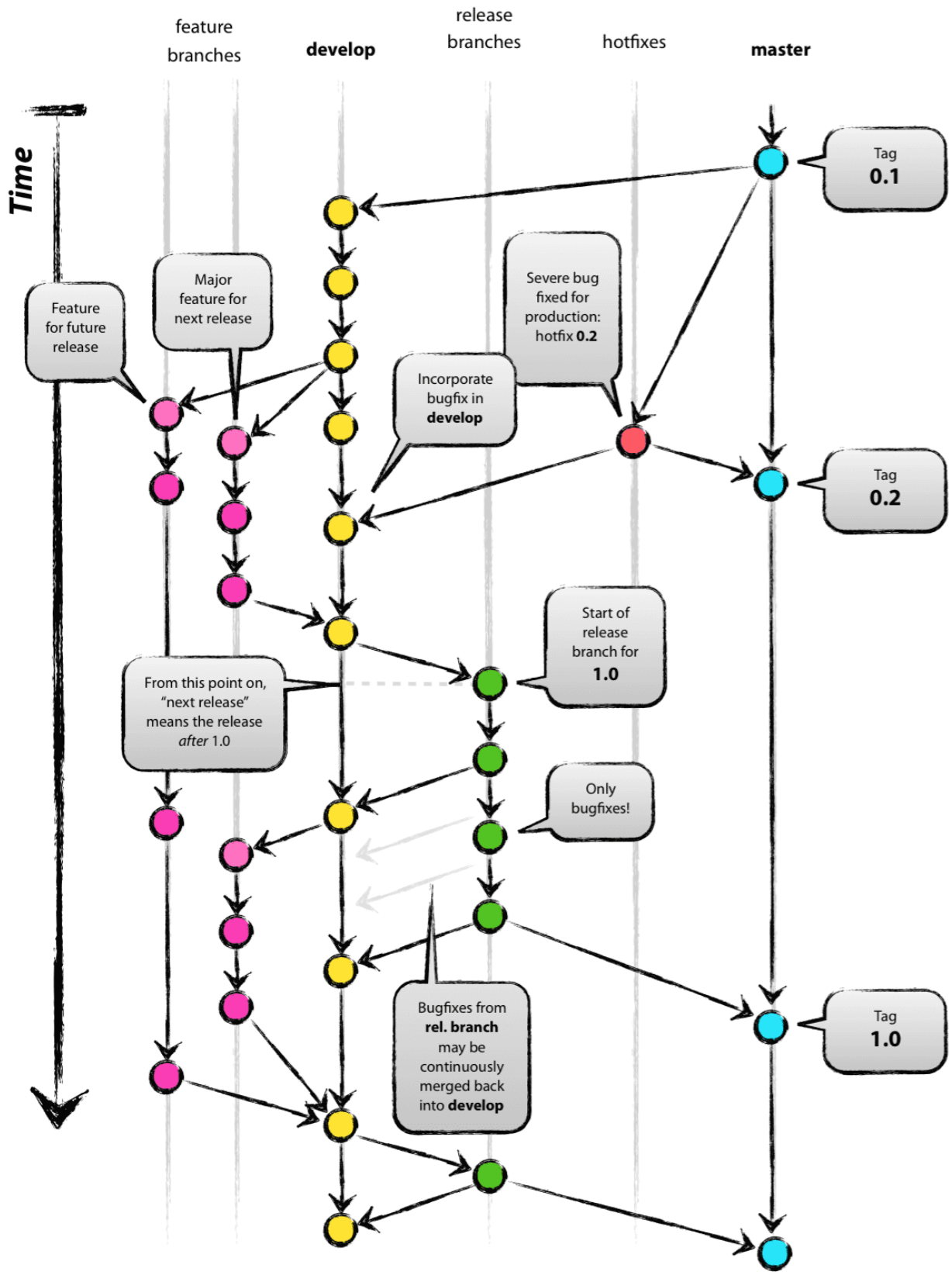
- 统一使用 Gitflow 工作流

3.2 提交的准则

- 除了源码、配置、说明文档、测试数据等相关的东西，其他软件 build 产生的临时文件（如 Debug 文件下面的所有内容），均不能提交进入源码仓库，需要添加到.gitignore 文件中忽略掉；
- 撰写规范的提交说明，见后面的 commit message 规范。一份好的提交说明可以帮助协作者更轻松有效地配合工作。
- 严格按照工作流切换到指定的分支，开发相应的功能。

3.3 分支简述

对一个库而言，在一个库里面，分为若干分支，比如 master 和 develop 分支。master 分支一般是稳定的主分支，代表了产品功能的稳定版本。而 develop 分支则用于开发过程中添加新功能的版本，是有待测试的不稳定版本，可能存在各种各样的 bug。当然实际工作中我们参考的 Gitflow 流程拥有更多的分支。


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

分支分为：

- master: 源码的主分支
- develop: 功能集成分支
- feature: 功能开发分支
- release: 发布分支
- hotfix: 维护分支

3.3.1 master 分支

master 分支用于存储正式发布的历史，为该分支的所有提交分配一个版本号（tag）。不允许直接往这个分支提交代码，只允许从 release 分支和 hotfix 分支进行 merge。

3.3.2 develop 分支

develop 分支作为功能集成开发的主分支，是相对稳定的分支，用于日常开发，包括代码优化、功能性开发的父分支。

3.3.3 feature 分支

每个新功能位于一个自己的分支，用 develop 分支作为父分支。当新功能完成时，合并回 develop 分支。

3.3.4 release 分支

一旦 develop 分支上有了做一次发布的足够功能，就从 develop 分支上 fork 一个发布分支。

新的发布分支用于开始发布循环，从这个时间点开始之后新的功能都不能再加到这个分支上，这个分支只应该做 bug 修复、文档生成和其他面向发布任务。

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。

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。

3.3.5 hotfix 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。

3.3.6 命名约定

- 主分支名称：master
- 主开发分支名称：develop
- 标签（tag）名称：v*.RELEASE，其中*为版本号。
- 新功能开发分支名称：feature-*，其中*为新功能简述，如：feature-spi
- 发布分支名称：release-*，其中*为版本号，如：release-1.0.0
- master 的 bug 修复分支名称：hotfix-*，其中*为 bug 简述，如：hotfix-spi-bug

4 Commit message 规范

统一的 git commit 日志标准便于后续代码 review，版本发布以及日志自动化生成。

首先不允许在 git commit 增加-m 或—message 参数，提交的时候用 gti commit 命令进入多行的 commit messages 书写模式。

基本语法如下：

```
<type>:
<subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

type 表示提交类别，subject 表示标题行，body 表示主体描述内容。

type 的类别说明：

- feat: 添加新特性
- fix: 修复 bug
- docs: 仅仅修改了文档
- style: 仅仅修改了空格、格式缩进等等，没有改变代码逻辑
- refactor: 代码重构，没有加新功能或者修复 bug
- perf: 优化相关，比如提升了性能

- test: 增加了测试相关的内容，比如测试用例
- other: 其他未能分类的场景

footer 一般加上与外部链接的说明，或者针对某个 issue 的操作。没有可不填。

完整的格式如下：

```
#类型：描述主要变更内容，50 个字符以内
#空行
#主体：更详细的说明文本，建议 72 个字符以内。需要描述的信息包括：
# * 为什么这个变更是必须的？ 它可能是用来修复一个 bug，增加一个 feature，提升性能、可靠性、稳定性等等；
# * 如何解决这个问题？ 具体描述解决问题的步骤；
# * 是否存在副作用、风险？
#空行
#需要的话可以添加一个链接到 jira 的 issue 或其他文档，也可以直接输入 jira 的 issue 的键，Bitbucket 会直接识别。
```

例子：

```
feat: 增加了 spi 驱动接口

利用库函数实现了 4 线制 spi 驱动，通过 dma 传输数据。

AG18-51
```

可以通过 git config 命令配置 commit 的模板，在每次 commit 的时候参考模板填写。需要注意的是，commit message 是否符合要求需要人为检查，一般是 review 的时候检查，务必养成良好习惯。