Update GitFlow.md
阿谨 authored just now

78c5a574

GitFlow.md 8.12 KB

GitFlow开发流程规范

GitFlow流程介绍

master:

- 主分支，产品的功能全部实现后，最终在master分支对外发布
- 该分支为只读唯一分支，只能从其他分支(release/hotfix)合并，不能在此分支修改
- 另外所有在master分支的推送应该打标签做记录,方便追溯
- 例如release合并到master，或hotfix合并到master
- Master分支上存放的是最稳定的正式版的代码,任何人不允许在Master上进行代码的直接提交，只接受合入,并设置相应的合入权限
- 该分支上的代码必须是经过多轮测试且已发布的release分支或hotfix分支合并进去，合并后生成相应的TAG，方便追溯（命名建议为：TAG+相应版本号）

develop:

- 主开发分支，基于master分支克隆
- 包含所有要发布到下一个release的代码
- 该分支为只读唯一分支，只能从其他分支合并
- feature功能分支完成，合并到develop(不推送)
- develop拉取release分支，提测
- release/hotfix 分支上线完毕，合并到develop并推送

feature:

- 功能开发分支，基于develop分支克隆，主要用于新需求新功能的开发
- 功能开发完毕后合到develop分支
- feature分支可同时存在多个，用于团队中多个功能同时开发，属于临时分支，功能完成后可选删除

release:

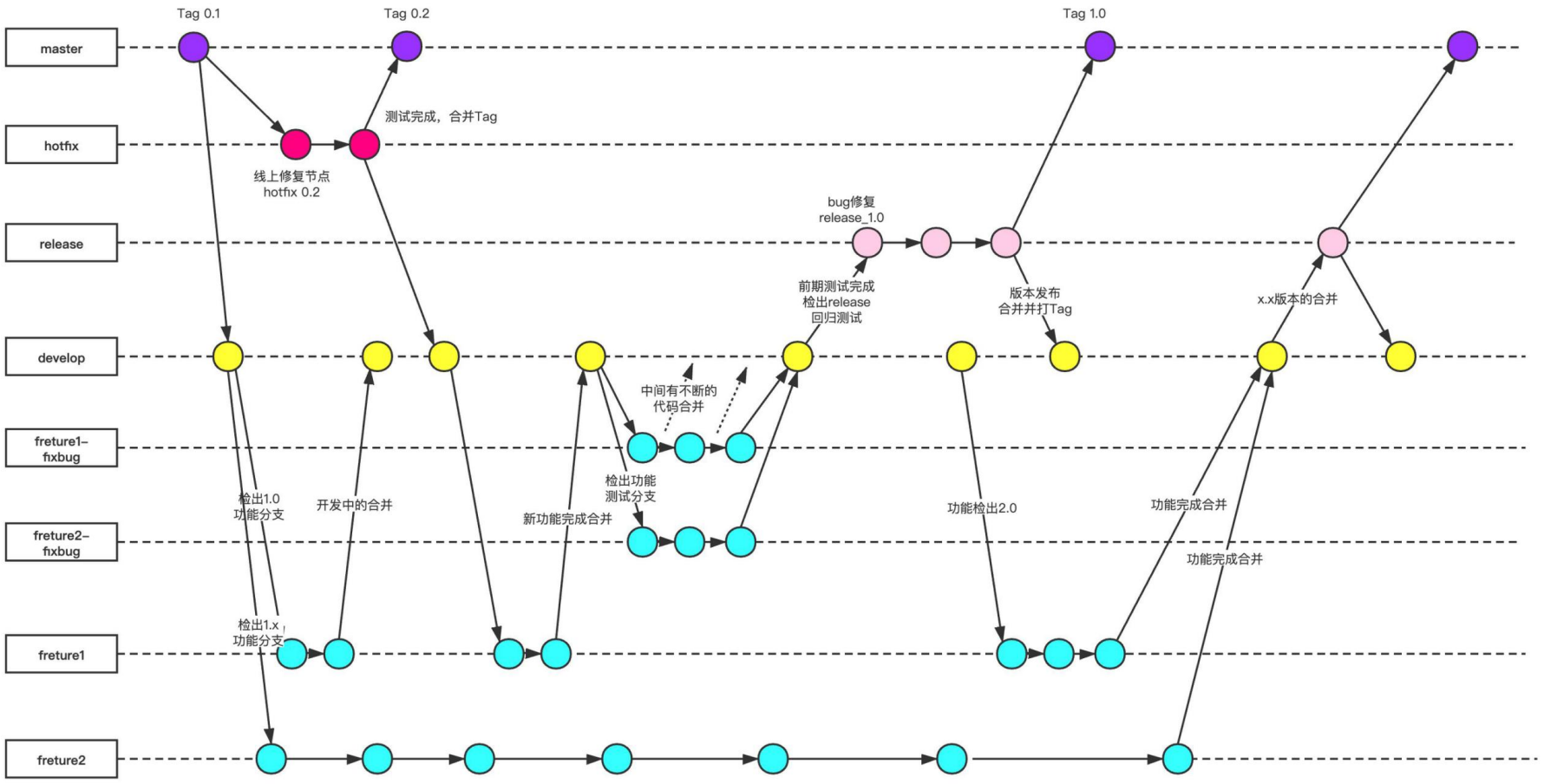
- 发布前分支，基于feature分支合并到develop之后 ，从develop分支克隆
- 主要用于提交给测试人员进行功能测试，测试过程中发现的BUG在本分支进行修复，修复完成上线后合并到develop/master分支并推送(完成功能)，打Tag
- 属于临时分支，功能上线后可选删除

hotfix:

- 补丁分支，基于master分支克隆，主要用于对线上的版本进行BUG修复
- 修复完毕后合并到develop/master分支并推送，打Tag
- 属于临时分支，补丁修复上线后可选删除
- 所有hotfix分支的修改会进入到下一个release

工作流程

示例图：



> develop不能直接进行提交，只允许检出和合并
>
> develop不允许被合并到其他分支，其他分支需要更新develop的代码只能rebase
>
> 所有需要修改的代码，都由原有分支检出（master、develop等），再合并

新功能开发：

- 1.从develop检出feature_version_name（例）分支，进行新功能开发
- 2.功能开发完成merge request，请求合并到develop（不严格要求，即未完成也允许请求合并）
- 3.功能开发完成后删除本地及远程分支（或者规定在下个版本开始时删除）
- 注：每次提交前基于develop进行rebase（变基）操作，理论上在未合并前不应该重复rebase，如果出现需要的情况，在分支上出现提示远程代码有更新时删除远程分支，以本地的记录为基准

> **rebase（变基）**
>
> 将当前的修改在最新的节点进行基点改变，相当于把当前的修改全部移动到最新的节点上进行的修改，可以在变基期间就在本地解决了冲突等问题，不需要在merge中去解决这些问题。

> **提交规范：Commit#Type**
>
> 1. **add** 新功能（feature）
> 2. **fix** 修补bug
> 3. **docs** 文档（documentation）,如 修改README、注释
> 4. **style** UI样式改动，如调整颜色、更改圆角、字体
> 5. **refactor** 重构
> 6. **test** 增加测试
> 7. **chore** 构建过程或辅助工具的变动，修改版本号、混淆配置

修复BUG：

> 前提：release定义为预发布分支，只进行最终的回归测试

- 1.从develop中检出feature_version_name_fixbug分支,进行bug修复
- 2.每修复一个bug生成一次提交（【fixbug】detail）(可以快速的定位到问题代码)
- 3.修复完成后进行merge request
- 4.从develop进行前期的开发测试
- 5.开启最终的回归测试后启动release_version预发布分支，从develop检出，回归测试中出现bug只在release_version分支解决，最终merge到develop和master中，并打上Tag。
- 6.并行开发过程中，非当前版本分支只允许rebase，不允许合并请求。

> 优点：release分支中的代码是有保证且相对稳定的，可以快速追溯到问题点（改动被控制）。develop分支严格控制在开发的整体流程中，master分支被严格保护。
>
> 缺点：操作较繁琐。

线上问题修复：

- 1.从master对应tag的节点检出hotfix分支。
- 2.修复并进行测试。
- 3.完成后merge到develop和master中，并打上Tag。
- 注：正常情况下只允许存在一个hotfix分支，避免分支混乱导致难以控制的问题，并在merge后删除。

GitFlow操作流程

- commit:本地提交，无限制
- pull:在需要push之前，切换到develop中进行pull操作
- push:切换到需要push的分支，push前先rebase到develop，再进行push

> 操作流程
>
> 1. 检出分支
> 2. 将修改的代码提交到本地
> 3. 切换到develop，pull最新的代码
> 4. 切换到自己的分支
> 5. push前的rebase，先pull下来develop，再rebase到develop，进行push
> 6. 在gitlab上进行Merge Requests,将分支上的修改合并到develop中
> 7. 审核者进行code review后进行代码合并

创建git-flow仓库

示例图（Sourcetree - 快捷键：option+command+F）：

新建相关分支:

示例图（Sourcetree - 快捷键：option+command+F）：

分支示例图：

> 如果是使用命令行，控制检出命令就行
>
> 如： **git checkout -b xxx**

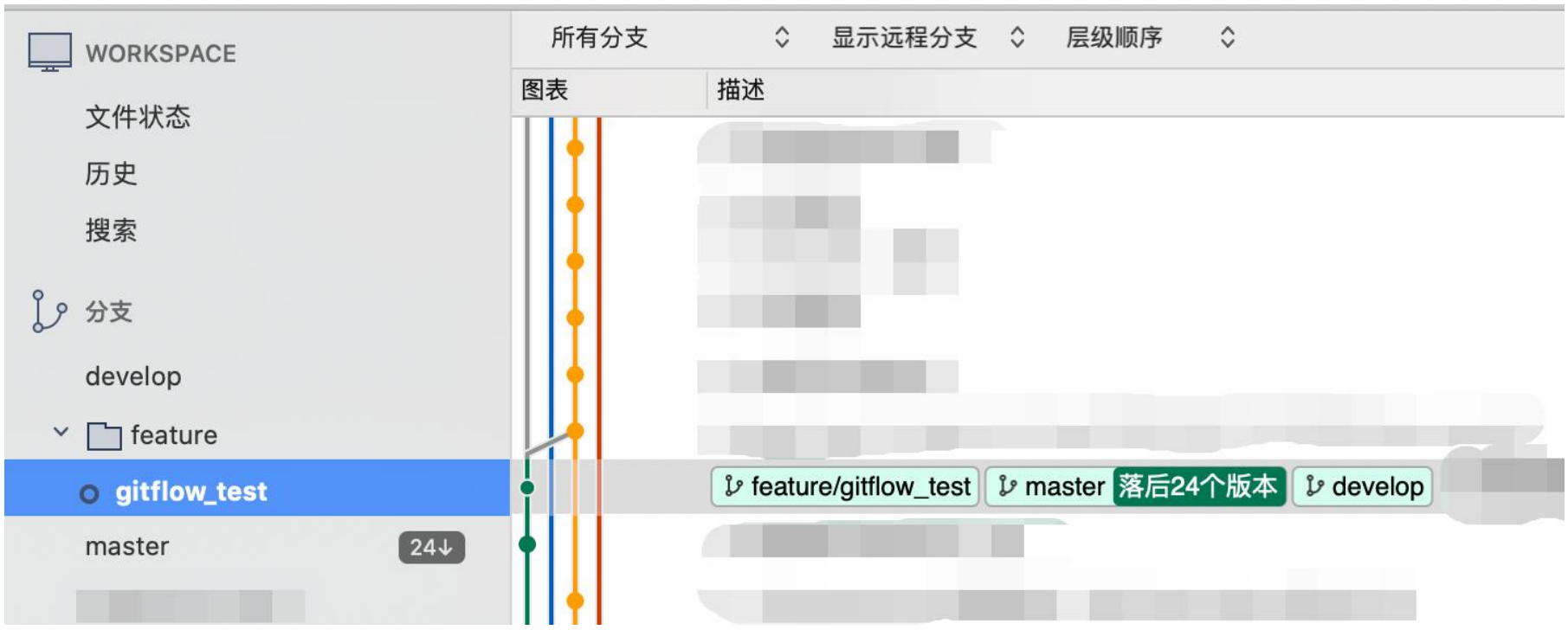
使用示例(SourceTree):

> *这里只是示例和注意事项，因为每个人的操作习惯都不一样，建议参考，但不强制要求执行*

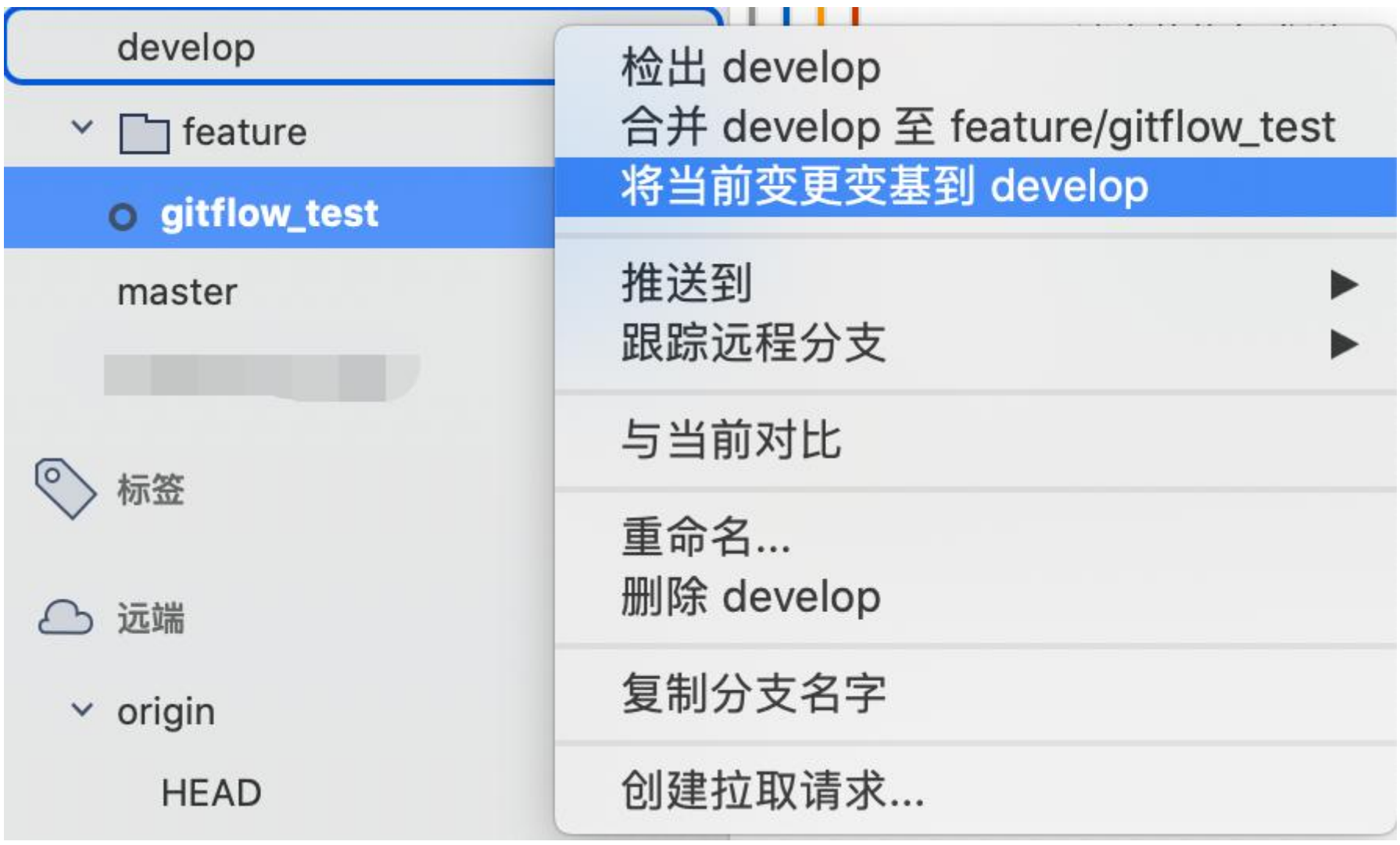
> 创建新的功能分支，如果目前有正在进行的功能分支，需要切换到develop中，否则会影响当前的功能分支进度。

示例及注意事项:

例1: 检出分支



例2:push前的rebase，先pull下来develop，再rebase到develop，进行push



例3:分支未合并到develop，多次push后会出现下面这种情况，这时候删除当前分支的远程分支即可，再进行rebase和push

例4:完成功能开发（Sourcetree - 快捷键：option+command+F）

> 不建议在SourceTree上进行这个操作，因为上面的操作流程中已经包含了这个操作了，而且更加灵活，可多次提交和合并，只需要在merge后删除本地分支和远程分支即可

例5:多分支提交后的流程支线（这两个分支都基于最底下的点检出的，但是按上面流程操作，分支就不会产生交叉）