Git分支管理

Master分支

`Master`分支主要作用是保持简单干净的分支关系、进行版本（`Tag`）管理等；

1. 发布的分支，只从`Develop`分支进行`merge`合并，不允许`commit`进该分支;
2. 每次发布（上线）新版本，请在此分支对应节点打标签`Tag`;

Develop分支

`Develop`分支主要作用：作为功能分支`feature/**`的基础、进行代码处理（合并、冲突解决等）等；如果不强调`Master`分支的作用，可以省略掉`Develop`分支，由`Master`分支承担起`Develop`分支的相关作用；

1. 所有功能分支均从`Develop`拉取；
2. 功能测试完成后，将功能分支`feature/**`合并到`Develop`；
3. 每日拉取`Develop`分支的最新代码，并合并到自己的功能分支`feature/**`：

> **说明**：此点比较重要，高频率地同步`Develop`分支的代码到`feature/**`分支，可以有效地避免过期引用问题（develop中已经更新的类、方法、常量等删除或过时标记后，`feature/**`分支还不自知，怡然自乐地自我开发，最后陷入代码修改的深渊之中）；

4. `Develop`分支的权限进行限制，只有管理员才可以进行代码合并（`feature/**`到`Develop`、`Develop`到`Master`），`feature/**`合并到`Develop`需要进行`pull request`操作；

Test分支

此分支非必须，只针对`多个开发功能`并行测试且仅有1台测试机（测试资源有限）的情况：

1. 按需从`Develop`分支拉取，测试完之后可立即将其删除；如果想创建多个测试分支，请先创建测试分支目录；
2. 将开发完成待测试的功能（1个或n个）分支合并到`test`分支；
3. 在功能分支`feature/**`中进行bug修复，修复后再合并到`test`分支中，勿在`test`分支中直接进行bug修复；
4. 测试通过之后`test`分支便与开发人员无关，测试同步并决定上线的功能分支可发起`pull request`合并到`Develop`；

Feature分支

通过创建`feature/**`分支，达到隔离开发的目的；

1. 开发之前，从`Develop`中拉取功能分支`feature/**`，之后在`feature/**`中进行代码开发；
2. 开发完成后，合并到`test`分支（如果没有，自行创建）进行测试；
3. 测试通过后，`pull request`到`Develop`；

Bugfix分支

进行bug修复使用；

1. 以`Develop`分支为基础，在`bugfix`分支目录下创建`bug-**`分支；

> **说明**：如果没有`Develop`分支，可以`Master`分支为基础，拉取bug分支；如果有`Develop`分支，因为`Develop`分支始终与`Master`进行同步，并且，bug修复也可以视为功能开发，所以从`Develop`分支切出更加合理；

2. bug修改后，合并到`test`分支（没有时主动创建）进行测试；
3. 测试通过后，`pull request`到`Develop`分支；

4. 管理员合并`Develop`到`Master`，准备上线部署；

其他：

1. 为了保持git的分支流简洁清晰，所有`bug-**`分支、`feature/**`分支、`test`分支，在上线部署成功之后可以删除;