Git分支规范

## 现状与问题

图1现状模型图

目前项目拥有master，hotfix，release，feature四个分支，但每个分支的功能和作用并没有起到作用。

Master：

1. 发版后，release分支并没有及时的合并到master。Master分支不能与发布版本保持一致，一旦发布版本出现问题后，没办法及时切换到发布的节点进行修改。
2. 权限没做好控制，每个人都拥有权限进行提交，合并。容易造成误提交等问题，导致master不能与发布的版本保持一致，对发布版本进行修改时，容易因为误提交而引入新的问题。
3. 合并后，没打好tag。不能立马定位到对应的版本

Hotfix：

1. 目前虽然有hotfix分支，但实际修改发布版本时，却很少在该分支上进行修改。
2. 现场问题有的在release上修改，有的在hotfix上修改，相对比较混乱。分不清什么时候在hotfix上修改，什么时候在release上修改。
3. 在hotfix上修改后，没及时合并到master和release（无develop，直接采用release）上。

Release：

1. 多人多功能在该分支上进行开发，许多功能都直接在release上进行开发或修改。比较难以控制每次提测的内容。
2. 现场问题，测试问题，开发问题，都在改分支上进行修改。若现场问题，修改完后，发布到现场容易引入其他问题。
3. 开发，冒烟，稳定均在该分支上。容易开发修改了问题，顺道把测试的问题一道修改了，容易破坏稳定环境的代码。
4. 发布完后，没及时的合并到master。

Feature：

1. 每创建一个feature分支，开发完后，对无用的feature分支没及时清理，分支多了，不便管理。
2. 无developt分支提供合并，直接合并到release分支上。

## gitFlow分支模型



图2.gitflow分支模型

Master:为已发版源码

1. 项目创建时，会自动创建master分支，有且只有1个，该分支一直存在，始终不删除
2. Master分支不做开发，修改
3. master分支的更新来自于release或hotfix分支合并
4. master分支只能由主程序员操作
5. 每次合完一次master需要打一个Tag

Develop:为主开发分支，包含已发布版本，提出版本

1. 主程序员创建，有且只有1个。该分支一直存在，始终不删除
2. develop分支上可以直接修改，可以合并其他分支
3. 基于develop分支可以创建feature分支
4. 该分支由master分支创建出来
5. 测试通过后该分支会打出一个release分支，供进一步为发布做准备

Release:全部测试过的代码，为发布准备的分支

1. 该分支基于develop分支创建
2. 可直接在该分支上进行修改
3. 测试通过发布后，需要将该分支代码合并到master，同时需要将该代码一起合并到develop上
4. 该分支为临时分支，发版并合到master和develop后，该分支废弃，需要进行删除

Hotfix:快速修复分支

1. Hotfix分支来自master对应的版本
2. 发布后，发现发布版本有bug时，需要创建hotfix分支进行修改
3. 修改后需要将修改的内容合并到master和develop分支，以保证master分支与发布的版本一致
4. 合到master和develop分支后，该分支进行删除

Feature：开发新功能分支

1. 该分支基于develop分支创建
2. 新功能在该分支上进行开发
3. 自测通过后，将该分支合并到develop分支
4. 该分支为临时分支，合并到develop分支后，该分支便可以废弃

## 项目分支

### 项目流程

#### 流程模型



图3.执行流程图

#### 环境对应



#### 补充说明



### 开发阶段

新功能主要在Feature分支上进行。

#### 基本规范

1. 基于develop分支
2. 由指定人员进行创建
3. 命名为【feature/功能】
4. 合并到develop分支后，测试通过并发布后删除该分支
5. 合并到develop分支后，便不再该分支上进行修复bug

#### 创建基准

1.新功能，且工时大于3天或不在本次发版范围内

2.本版本不发版

3.删除时间点为发布后

#### 注意事项

1. 使用的数据库为开发环境的数据库
2. 合并前需要自测是否有重大问题

### 测试阶段

测试与修改缺陷阶段主要在develop分支上进行

#### 基本规范

1. Develop分支由指定的人员进行创建
2. Develop分支由指定的人员进行合并
3. 从release分支合并后，开发人员需要在该分支上进行冒烟测试
4. 冒烟测试通过后，由测试人员在该分支上进行初步测试
5. 测试出现问题，开发人员需要在该分支上进行修复
6. 修复完后，确认准备进入发布阶段，需要基于该分支由特定人员创建release分支，进入稳定环境
7. 进入稳定环境后，需要下次发布的feature便可合并到该分支上，继续测试

#### 合并基准

1. 全量脚本（第一次提测提供，作为初始化）
2. 增量脚本
   1. db\_xtypt初期采用全量脚本(前期测试期间开发库保持与测试库一致，方便提供全量脚本)
   2. 后期采用增量脚本
   3. 其余模式采用增量脚本
3. 测试人员在develop上发现bug，开发人员直接在develop分支修改bug，在开发环境修改脚本自测通过后提供脚本交由测试人员在初测环境上执行

#### 注意

1. 自测后合并到develop分支上，若发现有feature有重大问题，可直接将该feature分支排除。
2. 若不包含其他功能的话，暂时不将feature合并到到分支上
3. 该分支使用的数据库为冒烟环境的数据库
4. 测试在该分支上进行测试

### 发布阶段

当在develop分支上测试通过后，准备发布时，便进入release分支，在该分支上进行发布准备。

#### 基本规范

1. 由指定的人员创建该分支
2. 该分支基于develop分支之上
3. 只有通过develop分支测试完后才进入release分支
4. 命名方式为【release/版本号】
5. 若在release分支上发现有缺陷，可直接在该分支上进行修复
6. 当该release测试没问题，发布后，需要将该分支合并到master和develop上。

#### 创建基准

1. 初测结束，且准备发版时，将develop分支合并到release上
2. 开发人员需要对初测后修改的脚本进行合并，并在做好稳定环境的冒烟测试，冒烟测试通过后，提交脚本给测试进行稳定环境搭建。
3. 稳定环境bug在release上进行修改。脚本在开发环境上进行修改，修改后自测提交测试执行

#### 注意事项

1. 该分支使用的数据库为稳定环境的数据库
2. 在该分支上进行稳定测试

### 发布上线

发布后，需要将release合并到master上。Master需要控制与发布后的版本一致。

#### 基本规范

1. 由指定的人将release合并到master
2. 合并后在master上打好tag，方便快速定位版本
3. 合并结束后，将上传合并权限关闭

注意：

为了保持master环境与上线后的环境一致，需要控制好master上传权限，预防误操作导致master引入其他代码。

### 快速修复

当发布后，发现发布的版本有缺陷时，需要快速进行修复，此时需要hotfix该分支。

#### 基本规范

1. 由指定的人从master发布版本创建hotfix分支
2. 在该分支上进行缺陷修复
3. 修复后，发布并合并到master和develop上
4. 该分支为临时分支，合并后该分支删除
5. 命名方式为【hotfix/版本号】

#### 使用基准

1. 当现场有问题时，在hotfix进行修改，并将修改后的代码合并到master与develop上
2. hotfix采用master对应备份数据库进行修改，脚本修改后合并到开发数据库与稳定环境数据库中

## 分支命名规范

|  |  |  |
| --- | --- | --- |
| 命名规范 | | |
| feature | Release | Hotfix |
| 【feature/功能】 | 【release/版本号】 | 【hotfix/版本号】 |