##测试环境
0、本地线下开发环境
1、线上delevop环境: 开发测试使用 功能特性分支测试
2、线上准生产模预发布环境：用于发布新版本前，测试功能 预发布前测试
3、线上生产环境

##版本号管理
使用3位数作为版本管理
v 0.0.1
1、当修复一个小bug，则版本号的第三位自增1，如v 0.0.2
2、当新增一个功能特性，则版本号第二位自增1，如 v 0.1.2
3、当有新的模块或框架大调整时，则版本号第一位自增1，如v 1.1.2

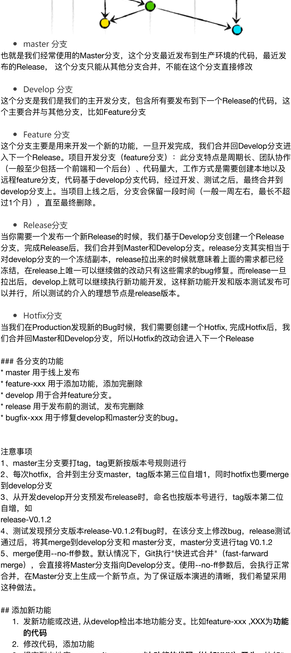
##版本更新列表checklist
1、每次发布新版本需列出所做变更 Version.txt里做对应说明，列明
时间版本号，如
　v 0.0.1　　2017-05-26
　1、系统首次上线
测试根据version.txt的变更进行release新发版本测试跟踪

##版本分支
it-flow 流程中包含 5 类分支，分别是 master、develop、新功能分支（feature）、发布分支（release）和 hotfix。这些分支的作用和生命周期各不相同，master 分支中包含的是可以部署到生产环境中的代码。develop 分支中包含的是下个版本需要发布的内容，从某种意义上来说，develop 是一个进行代码集成的分支，当 develop 分支集成了足够的新功能和 bug 修复代码之后，通过一个发布流程来完成新版本的发布。发布完成之后，develop 分支的代码会被合并到 master 分支中。

其他3类分支如下表

分支类型	命名规范	创建自	合并到	说明
feature	feature-XXX	develop	develop	新功能
release	release-XXX	develop	develop 和 master	一次新版本的发布
hotfix	hotfix-XXX	master	develop 和 master	生产环境中发现的紧急 bug 的修复

这三类分支只在需要时从 develop 或 master 分支创建。在完成之后合并到 develop 或 master 分支。合并完成之后该分支被删除。这几类分支的名称应该遵循一定的命名规范，以方便开发人员识别。



- master 分支

也就是我们经常使用的Master分支，这个分支最近发布到生产环境的代码，最近发布的Release。 这个分支只能从其他分支合并，不能在这个分支直接修改

- Develop 分支

这个分支是我们是我们的主开发分支。但会所有要发布到下一个Release的代码，这个主要合并与其他分支，比如Feature分支

- Feature 分支

这个分支主要是用来开发一个新的功能，一旦开发完成，我们合并回Develop分支进入下一个Release。项目开发分支（feature分支）：此分支特点是周期长、团队协作（一般至少包括一个前端和一个后台），代码量大，工作方式是需要创建本地以及远程feature分支，代码基于develop分支代码，经过开发、测试之后，最终合并到develop分支上，当项目上线之后，分支会保留一段时间（一般一周左右，最长不超过1个月），直至最终删除。

- Release分支

当你需要一个发布一个新Release的时候，我们基于Develop分支创建一个Release分支，完成Release后，我们合并到Master和Develop分支，release分支其实相当于对develop分支的一个冻结副本，release拉出来的时候就意味着上面的需求都已经冻结，在release上唯一可以继续做的改动只有这些需求的bug修复，而release一旦拉出后，develop上就可以继续执行新功能开发，这样新功能开发和版本测试发布可以并行，所以测试的介入的理想节点是release版本。

- Hotfix分支

当我们在Production发现新的Bug时候，我们需要创建一个Hotfix, 完成Hotfix后，我们会并回Master和Develop分支，所以Hotfix的改动会进入下一个Release

各分支的功能
* master 用于线上发布
* feature-xxx 用于添加功能，添加完删除
* develop 用于合并feature分支。
* release 用于发布前的测试。发布完删除
* bugfix-xxx 用于修复develop和master分支的bug。

注意事项
1、master主分支要打tag，tag更新按版本号规则进行
2、每次hotfix，合并到主分支master，tag版本第三位自增1，同时hotfix也要merge到develop分支
3、从开发develop开分支预发布release时，命名也按版本号进行，tag版本第二位自增，如
release-V0.1.2
4、测试发现预分支版本release-V0.1.2有bug时，在该分支上修改bug，release测试通过后，将其merge到develop分支和 master分支，master分支进行tag V0.1.2
5、merge使用--no-ff参数。默认情况下，Git执行"快进式合并" (fast-farward merge)，会直接将Master分支指向Develop分支。使用--no-ff参数后，会执行正常合并，在Master分支上生成一个新节点。为了保证版本溯进的清晰，我们希望采用这种做法。

添加新功能
1. 发新功能或改进, 从develop检出本地功能分支。比如feature-xxx ,XXX为**功能的代码**
2. 修改代码，添加功能
3. 提交到本地库，**commit message以 功能的代码（比如XXX）开头。**比如" 重新设计API接口"
4. 本地自测，测试功能。如果有bug，修改代码本通过跳到2：
5. 将develop分支merge到feature分支，解决冲突，push到远程，提交pull请求
6. 找人 review代码并merge到develop

一个开发人员典型的提交流程：

```
//新建分支
git checkout develop
git pull origin develop
git checkout -b myfeature

//在分支上开发
git add ***
git commit -m "*****"

//在分支开发过程中合并develop分支到本分支（先把自己的工作commit到本地）
git checkout develop
git pull origin develop
git checkout myfeature
git merge develop

（如果没有冲突，就继续开发。如果有冲突，执行下面过程）
首先在本地解决冲突，再把冲突解决commit
git add ***
git commit -m "*****"


//在分支开发结束，需要将本分支推到远程
git push
```

紧急修复Bug
1. 从master检出本地Bug修复分支。比如 bugfix-XXX
2. 修复bug
3. 本地测试通过
4. 提交到本地库，**commit message以 问题的代码 bug-XXX开头。**比如"BUG-57 修复 版本太多显示不全"
5. push分支到远程
6. 部署到线上准生产环境测试通过
7. 提交pull请求
8. 找人 review代码并merge到develop、master，master打tag
9. 部署到生产

发布新版本
1. 给develop添加tag：v+版本号+a
2. 从develop检出release_latest
3. 部署到线上develop开发环境测试 **【QA】开发环境非feature集合测试**
4. QA测试人员测试，如有bug，按照 修复bug流程 修复，并merge到release_latest。重新部署，
5. 测试通过后release添加tag：v+版本号+b **【QA】准生产环境预发布测试**
6. 部署到准生产测试环境，线上准生产测试
7. 如有bug，按照 修复bug流程 修复，并merge到release。重新部署。
8. 给release添加tag：v+版本号
9. 发起pull请求，merge到master，master打tag，部署到生产

##代码部署
- 部署测试

部署测试的代码，是经过自测且实现功能点代码，部署前需要通知到其他开发人员。

- 部署正式

代码经过测试之后，已经是稳定代码，达到了上线标准之后就可以部署代码：遇到严重问题需回滚代码。
部署前需通知相关负责人。

##代码回滚
回滚条件：正式线上遇到重大bug