**版本控制规范**

**Version 1.0**

[1.目的 1](#_Toc38014838)

[2.适用范围 1](#_Toc38014839)

[3．工作程序 2](#_Toc38014840)

[3.1项目开发人员注意事项 2](#_Toc38014841)

[3.2代码分支管理 2](#_Toc38014842)

[4.版本号设置规则 8](#_Toc38014843)

[4.1软件版本命名规则 8](#_Toc38014844)

[4.2封包版本命名规则 9](#_Toc38014845)

# 1.目的

规范软件产品版本升级流程，清晰管理版本号，加强不同版本软件保存的可靠性。

# 2.适用范围

版本控制的范围包括：

源代码：用计算机编程语言编写的源代码文件

文档：需求规格说明书、总体设计说明书、数据库设计说明书、详细设计说明书等描述软件功能和结构的技术文档；项目计划等项目管理文档以及各种测试文档和用户文档

产品包：将源代码进行编译得到的可运行的软件系统

# 3．工作程序

## 3.1项目开发人员注意事项

3.1.1 项目开发过程中所有的所有代码必须上传至Github远程仓库。文档和代码需打包上传至QQ群，先建立相关文件夹再上传，严禁直接上传。

3.1.2 开发人员每周至少更新两次代码，将修改后的代码push到Github库，commit命令注释需认真填写。

3.1.3 开发人员更新或提交代码时如果发现有代码冲突，需立即找代码冲突的相关人员讨论查找原因，严禁直接强制提交。

3.1.4开发人员提交基线代码时，需要打附注标签。

git tag -a v0.1.0 -m "一段描述"

v0.1.0是软件版本号。（参考软件版本命名规则）

描述包括：简单描述这一版本中更矮、新增的主要功能，或者这一版本中已修复的缺陷。

git push origin v0.1.0

之后不要忘了push标签到远程仓库

3.1.5没有通过编译的代码不必提交。

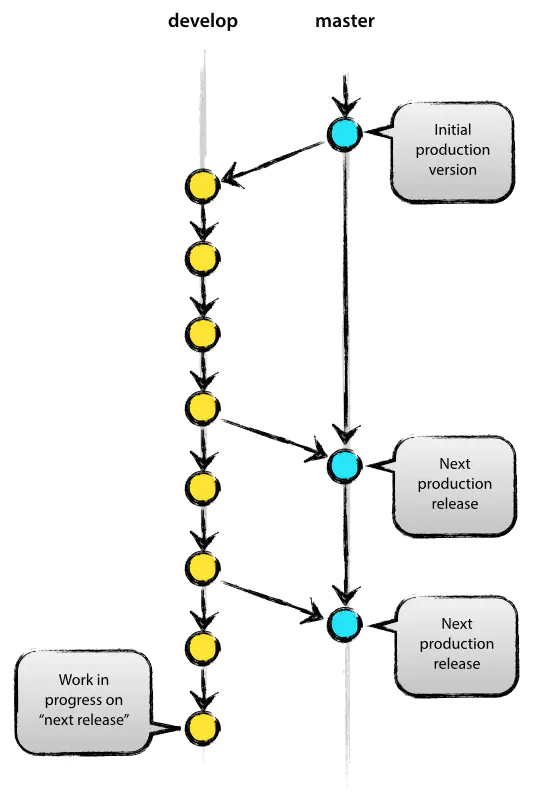
## 3.2代码分支管理

#### 3.2.1分支管理组成

首先，项目存在两个长期分支

1. **master主干**：在版本管理中，代码库应该仅有一个主干。此主干是和当前生产保持一致的，是可用的、稳定的可直接发布的版本，不能再主干上进行任何开发操作。git主干的名字，默认叫做 master，它是自动建立的。

2. **develop主开发分支**：因为不能在主干master上进行开发，那么就需要在基于主干master的基础上，创建一个开发主分支develop，开发主分支develop的代码永远是最新的，所有的新功能都是以此分支为基础进行开发的，该分支只是做合并操作，也不能在此分支进行实际开发。



其次，项目存在三种短期分支。

3.**feature功能开发分支：**功能开发分支，在develop上创建分支，采用“feature-” +“分支创建时间”+ “批次名称-”的命名规范。

例如：“feature-20190301-XXX”

此分支既作为需求开发分支又作为需求测试分支，所有需上线内容需在当前分支充分测试通过后，才可提交test分支与其他待上线分支代码进行合并，然后进行test分支回归测试。

4.**test测试分支：**test分支它是指发布正式版本之前（即合并到 master分支之前），我们需要有一个预发布的版本进行测试。

预发布分支是从develop分支上面分出来的，预发布部署生产验证无误，结束以后，必须向下合并进 master和develop分支以及develop衍生所有开发分支，保证各分支基线版本与生产基线同步。

5.**hotfix紧急bug分支：**项目上线后会遇到一些需要紧急修复的bug，那么就需要创建一个紧急bug修改分支，此分支需要从master直接拉取分支进行开发修改，修复完成后必须向下合并进 master和develop分支以及develop衍生所有分支，保证各分支基线版本与生产基线同步。

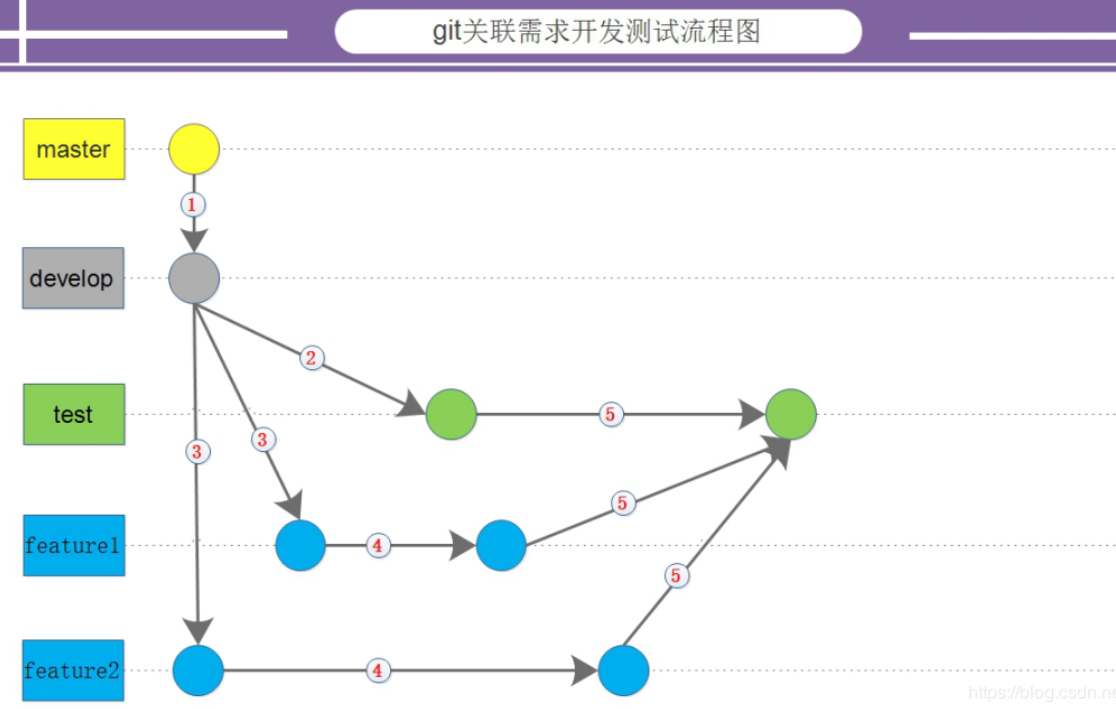
采用“hotfix-” +“分支创建时间”+“bug号或bug描述”的命名规范。

例如：“hotfix-20190116-001”

#### 3.2.2分支使用流程

**1.开发关联需求**

关联需求：A和B同时开发一个需求内容，A所开发的内容有部分是基于B开发的内容。



上图为关联需求的开发及测试流程图，以下为每个节点的介绍：

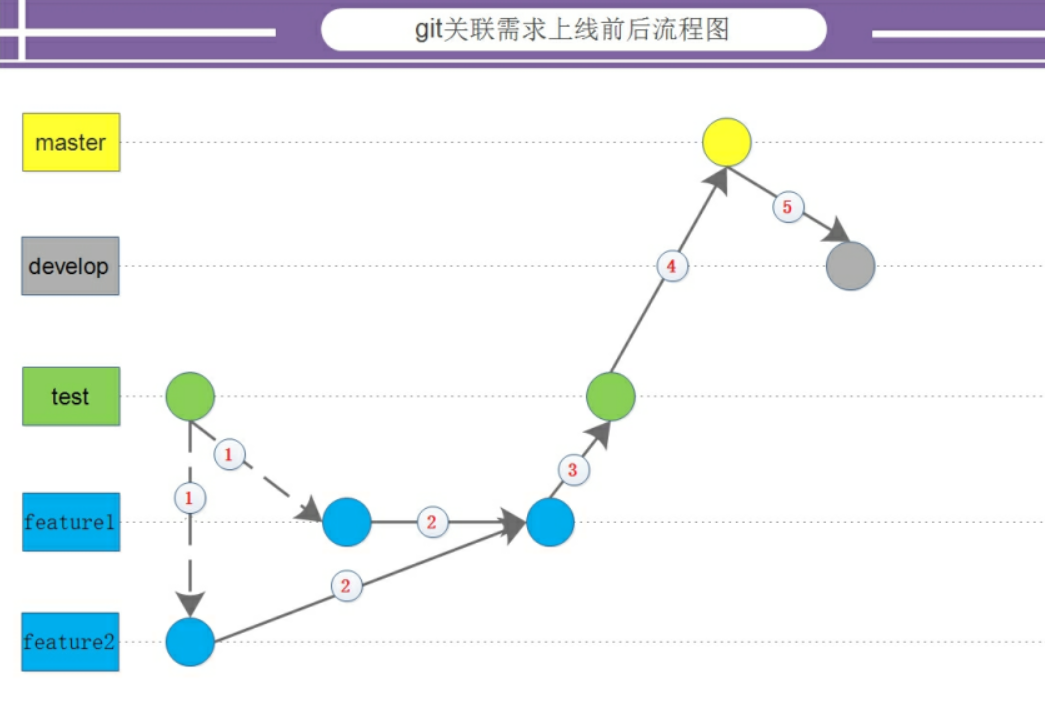
①　从master主干的基础上拉取develop分支作为开发分支

②　从develop分支的基础上拉取test分支作为测试分支

③　从develop分支的基础上拉取feature分支作为开发分支

④　在feature开发分支进行需求的开发

⑤　将并行的需求feature分支单独测试，然后合并到test测试分支进行上线测试操作



上图为关联需求的上线前后流程图，以下为每个节点的介绍：

①　在test分支测试通过

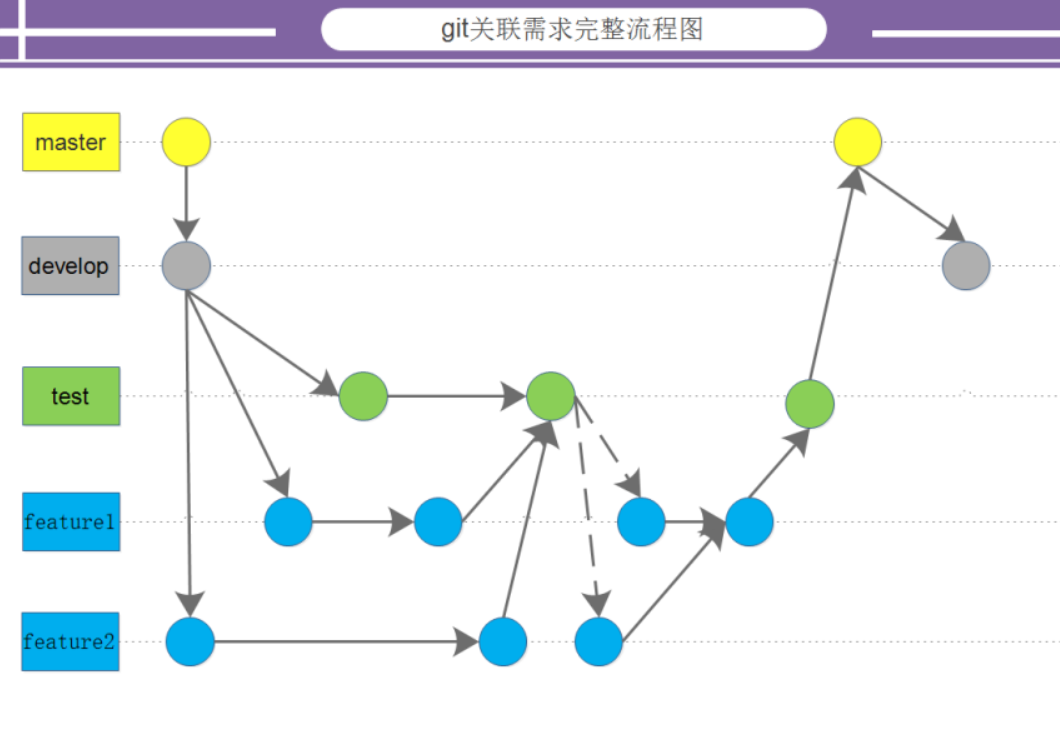
②　将feature2的开发分支合并到feature1的开发分支中

③　将合并后的feature1分支的内容合并到test分支回归测试

④　test分支回归测试完成，用master分支合并test分支上线

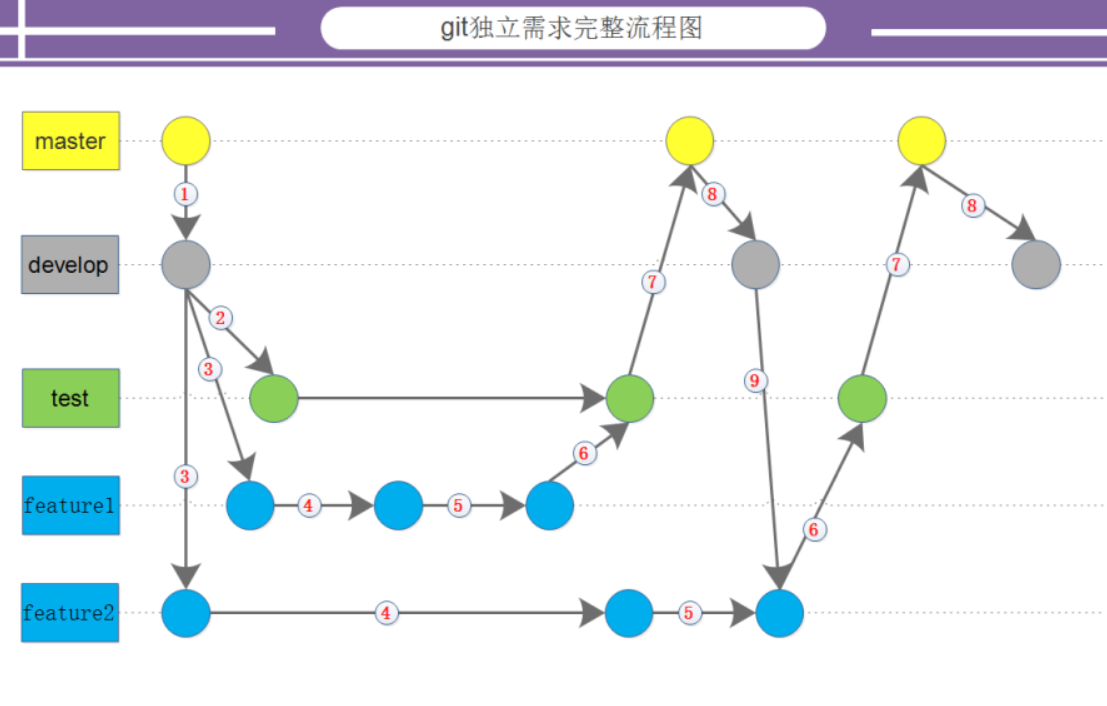
⑤　将master分支向下合并到develop分支及其他正在开发中分支

下图为关联需求分支的整体流程图：



**2.开发独立需求**

独立需求：A和B同时开发两个完全不相关的需求内容。



上图为独立需求的完整流程图，以下为每个节点的介绍：

①　从master主干的基础上拉取develop分支作为开发分支

②　从develop分支的基础上拉取test分支作为测试分支

③　从develop分支的基础上拉取feature分支作为开发分支

④　在feature开发分支进行需求的开发

⑤　对feature分支的内容进行测试

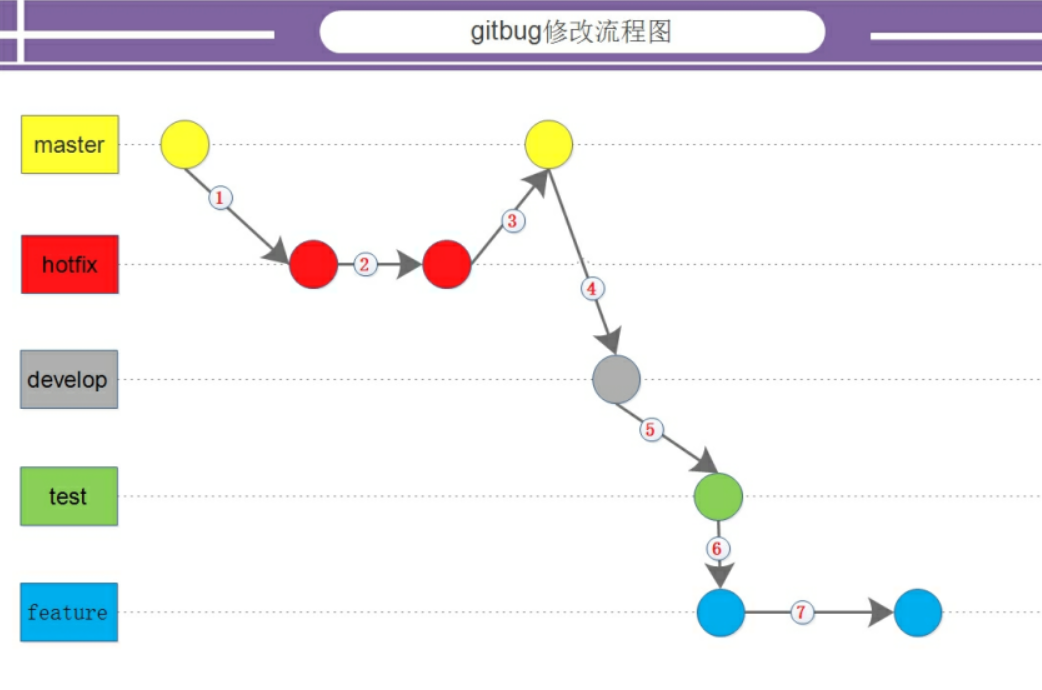
⑥　feature1分支测试通过后，将合并后的feature1分支的内容合并到test分支进行预上线测试

⑦　测试通过后，将test分支合并到master主干上线

⑧　将master主干合并到develop分支中，保持master主干和develop分支内容一致

⑨　将develop分支合并到feature2中，保证feature2分支已经包含已上线的feature1内容

**3.修复bug**



①　在master主干根据命名规范创建hotfix分支

②　hotfix分支内容修改并测试

③　hotfix分支测试通过后合并master主干上线

④　将master主干的内容合并到develop和test分支，保证develop分支内容与主干分支同步，从而保证与生产环境同步

⑤　将develop分支内容合并到test分支，保证test分支与主干同步

⑥　将test分支的内容合并到feature分支，保证feature包含hotfix分的内容

⑦　feature分支继续开发

# 4.版本号设置规则

版本号用来标识开发、测试、交付阶段的不同状态的产品。

## 4.1软件版本命名规则

版本格式： v<主版本号>.<副版本号>.<修订号>

**说明：**

1. 标准的版本号必须（MUST）采用 X.Y.Z 的格式，其中 X、Y 和 Z 为非负的整数，且禁止（MUST NOT）在数字前方补零。X 是主版本号、Y 是次版本号、而 Z 为修订号。每个元素必须（MUST）以数值来递增。例如：1.9.1 -> 1.10.0 -> 1.11.0。
2. 以 0.1.0 作为初始化开发版本，并在后续的每次发行时递增次版本号。

**版本号递增规则如下**：

1. 主版本号 X（X.y.z | X > 0）必须（MUST）在有任何不兼容的修改被加入公共 API 时递增。其中可以（MAY）包括次版本号及修订级别的改变。每当主版本号递增时，次版本号和修订号必须（MUST）归零。此版本号由项目组决定是否修改。
2. 次版本号 Y（x.Y.z | x > 0）必须（MUST）在有向下兼容的新功能出现时递增。在任何公共 API 的功能被标记为弃用时也必须（MUST）递增。也可以（MAY）在内部程序有大量新功能或改进被加入时递增，其中可以（MAY）包括修订级别的改变。每当次版本号递增时，修订号必须（MUST）归零。此版本号可由开发人员决定是否修改。
3. 修订号 Z（x.y.Z | x > 0）必须（MUST）在只做了向下兼容的修正时才递增。这里的修正指的是针对不正确结果而进行的内部修改。此版本号由开发人员决定是否修改。

## 4.2封包版本命名规则

\*\*\*\*. X.Y.Z.YYMMDD

**描述：**

项目名称\*\*\*\*： 项目拼音的首字母大写。

软件版本号X.Y.Z：软件的版本号。开发人员设置此软件版本号。规则参见“软件版本命名规则部分”。

日期YYMMDD： 六位数字表示，2位数字年，2为数字月，2位数字日。