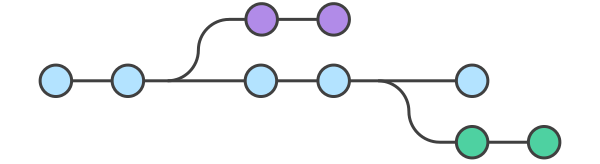
Git Flow

# 概述

Git 作为一个版本控制管理系统，不可避免涉及到多人协作。

协作必须有一个规范的工作流程，让大家有效地合作，使得项目井井有条地迭代开发。"工作流程"在英语里，叫做"workflow"或者"flow"，原意是水流，比喻项目像水流那样，顺畅、自然地向前流动，不会发生冲击、对撞、甚至漩涡。



# 目前现状以及愿景

## 现状

1. 最新开发分支：dev 客户生产分支：vXXX.XXX分支
2. 分支开发流程：项目当前迭代在dev分支开发，项目组员可自由往dev提交最新代码改动(前提：提交无错误不影响编译)。一般情况，要求开发成员开发联调完几个小需求或者一个大需求后，将代码更新至dev分支上，同时从dev分支编译最新的库或者jar包安装部署到测试环境，tapd需求改为转测试，同时提醒测试人员开始验证。PS：如果需求涉及到前后端一起改动，一定要前后端全部开发完成并联调成功后，由最后完成的开发人员更改需求并通知测试人员测试。
3. 发bate流程：测试环境所有需求和bug都测试通过之后，由项目负责人统一通知前后端开发人员打tag，tag号由项目负责人统一拟定。tag打完之后将tag号告知运维人员（武松/李晓），由运维人员打包部署bate环境。项目负责人打出当前迭代对应的分支(当前迭代为ADC3.8，则分支为v3.8)，该分支作为以后解决v3.8客户的现场测试问题以及生产问题分支。同时项目负责人开始着手迭代版本的文档更新。其它开发人员有后续迭代需求开发时，可继续在dev分支上开发提交。
4. bate环境测试验证及发版：bate环境复验出来的问题，开发人员在新打出的迭代分支（v3.8）上开发，所有bate上复验的bug都解决完成后，在当前迭代分支(v3.8)上统一打出最终发版的tag，运维打正式版的升级包。同时由项目负责人将当前迭代分支(v3.8)上的改动内容，同步到dev分支上。

## 会议目的

Git 很灵活，每个团队都可以做出适合自己团队的开发流程（或标准），以适应不同的项目需求和特质，我们现在也是要找到适合ADC项目开发流程。

现在主要基于以下几点展开：

(1) 了解ADC现有的分支管理流程，基于现状，进行适当的改进；

(2) 列举现在ADC分支（版本）管理中，遇到的主要问题（主要矛盾）；

(3) 拟定引入Git Flow 分支管理流程，然后进行个性化的调整。（目前使用的流程也是类似GitFlow的）

# 功能驱动

采用"**功能驱动式开发**"（Feature-driven development，简称FDD）。

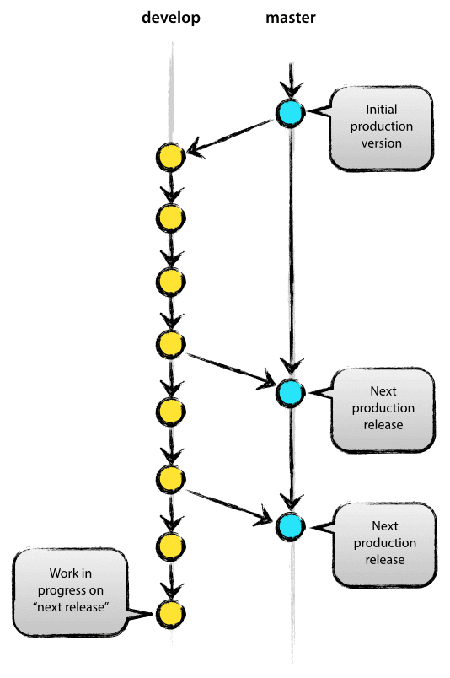
它指的是，需求是开发的起点，先有需求再有功能分支（feature branch）或者补丁分支（hotfix branch）。完成开发后，该分支就合并到主分支，然后被删除。

# Git flow

最早诞生、并得到广泛采用的一种工作流程，就是Git flow。

## 特点

它最主要的特点有两个。



首先，项目存在两个长期分支。

主分支master

开发分支develop

前者用于存放对外发布的版本，任何时候在这个分支拿到的，都是稳定的分布版；后者用于日常开发，存放最新的开发版。

其次，项目存在三种短期分支。

功能分支（feature branch）

补丁分支（hotfix branch）

预发分支（release branch）

一旦完成开发，它们就会被合并进develop或master，然后被删除。

## 小结

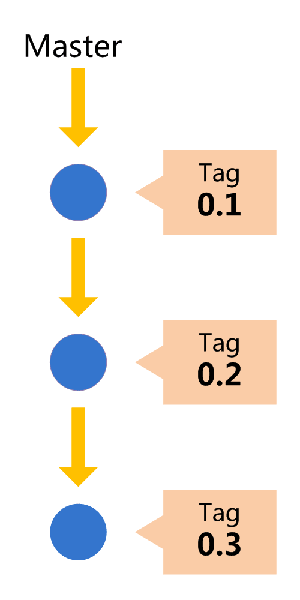
Git flow的优点是清晰可控，缺点是相对复杂，需要同时维护两个长期分支。大多数工具都将master当作默认分支，可是开发是在develop分支进行的，这导致经常要切换分支，非常烦人。

注意：**这个模式是基于"版本发布"的，目标是一段时间以后产出一个新版本**。

# Git Flow 实践

## 主分支Master

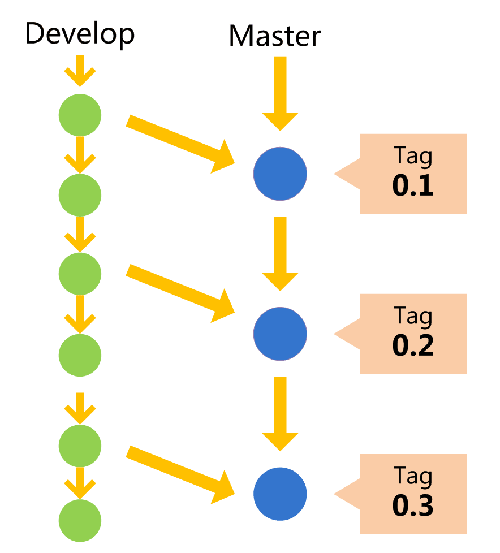
首先，代码库应该有一个、且仅有一个主分支。所有提供给用户使用的正式版本，都在这个主分支上发布。



Git主分支的名字，默认叫做Master。它是自动建立的，版本库初始化以后，默认就是在主分支在进行开发。

## 开发分支Develop

主分支只用来分布重大版本，日常开发应该在另一条分支上完成。我们把开发用的分支，叫做Develop。



这个分支可以用来生成代码的最新隔夜版本（nightly）。如果想正式对外发布，就在Master分支上，对Develop分支进行"合并"（merge）。

Git创建Develop分支的命令：

　　git checkout -b develop master

将Develop分支发布到Master分支的命令：

# 切换到Master分支  
　　git checkout master

# 对Develop分支进行合并  
　　git merge **--no-ff** develop

## 临时性分支

前面讲到版本库的两条主要分支：Master和Develop。前者用于正式发布，后者用于日常开发。其实，常设分支只需要这两条就够了，不需要其他了。

但是，除了常设分支以外，还有一些临时性分支，用于应对一些特定目的的版本开发。临时性分支主要有2种：

　　\* 功能（feature）分支

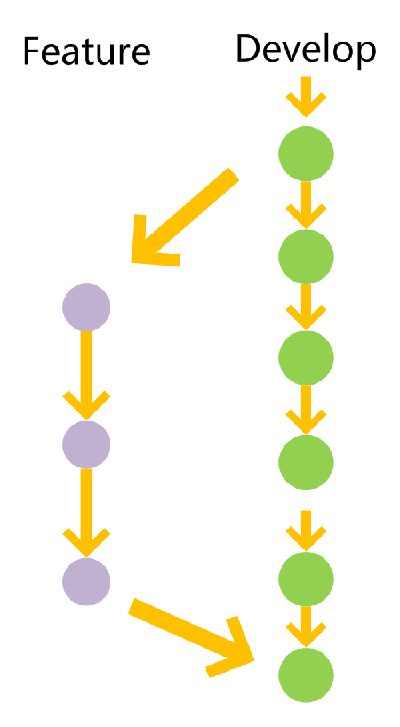
　　\* 预发布（release）分支

这2种分支都属于临时性需要，使用完以后，应该删除，使得代码库的常设分支始终只有Master和Develop。

## 功能分支

接下来，一个个来看这三种"临时性分支"。

第一种是功能分支，它是为了开发某种特定功能，从Develop分支上面分出来的。开发完成后，要再并入Develop。



功能分支的名字，可以采用feature-\*的形式命名。

创建一个功能分支：

　　git checkout -b feature-x develop

开发完成后，将功能分支合并到develop分支：

git checkout develop

git merge --no-ff feature-x

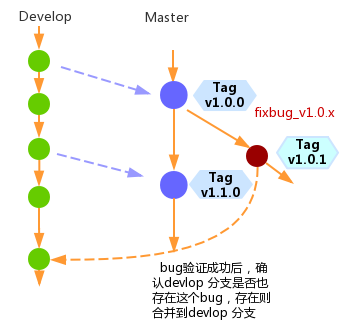
删除feature分支：

git branch -d feature-x

## 修补历史版本bug分支

最后一种是修补bug分支。软件正式发布以后，难免会出现bug。这时就需要创建一个分支，进行bug修补。

修补bug分支是从Master分支上面分出来的。修补结束以后，再合并进Master和Develop分支。它的命名，可以采用fixbug-\* 的形式。



创建一个修补bug分支：

git checkout -b fixbug-v1.0.0 v1.0.0

修补结束后，并且测试通过；再合并到develop分支：

git checkout develop

git merge --no-ff fixbug-v1.0.0

最后，删除"修补bug分支"：

git branch -d fixbug-v1.0.0

注意： 如果其他版本也需要修复该bug，可以checkout 到对应版本上,使用merge合并修改（或者cherry-pick）

# 注意事项

1 对于需求(或fixbug)的递交，commit-msg 可以带上tapd 上的 ID 。

git commit -m "fixbug(ID1008465): 修复文件解析问题: UTF-8转换为GBK的问题!"