**GIT使用规范**

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件标识： | GIT使用规范 |
| 当前版本： | 1.0 |
| 密级等级： | 内部公开 |
| 作 者： | XX、OO |
| 完成日期： | 2018/4/20 |

# 文档介绍

## 文档目的

本文档为Fintech团队内部规范文档，主要目的是管理团队内部的GIT使用规范、代码上线发布流程以及新员工入职指引。

## 文档范围

本文档包括文档说明；GIT的基本操作；代码上线发布流程；以及注意事项等内容。

## 读者对象

本文档的读者包括了如下人员：

1. Fintech项目管理人员；
2. Fintech产品经理；
3. Fintech软件开发人员；
4. Fintech软件测试人员

## 参考文献

《GIT Flow 工作流程》

Git分支管理策略 : <http://www.ruanyifeng.com/blog/2012/07/git.html>

# GIT的基本操作

### GIT简单入门

#### **1创建仓库** cd到工程目录（应该是空的），执行

git init

不过我们一般是不需要自己创建的，多半是公司服务器已经有仓库了，我们直接在工程clone下来就行。

// 从远程仓库拉取到本地

**git clone** [**git@gitlab.remarkfin.com:fintech/titan.git**](mailto:git@gitlab.remarkfin.com:fintech/titan.git)

#### 2开始工作 仓库设置好就可以开始工作了，先查看一下分支情况

git branch // 查看本地分支

git branch -r // 查看远程分支

创建功能分支

// 从dev分支创建一个feature\_x分支

git checkout -b feature\_x dev

#### 3 提价代码

当功能开发完成后需要提交代码

// 添加指定文件

git add <filename>

// 添加所有文件

git add .

// 提交

git commit -m "代码提交信息"

// 推送到服务器的feature\_x分支中

git push origin feature\_x

代码提交完成后需要在gitlab网页上提交merge request请求。

注意：当一个功能模块多次提交时，需要合并commit 信息，以保证提交信息清晰规范

git log // 查询提交日志

git rebase -i 【logId】 // 合并提交信息

git push origin 【branch】 --force-with-lease // 推送到远程分支

#### 4 合并远程代码

git fetch origin // 拉取远程分支代码

git rebase origin/dev // 合并远程dev分支代码到当前分支

注意:当合并远程分支时可能会出现冲突，这时需要解决冲突，并且重新提交

解决冲突：

// 修改本地冲突代码 // 添加指定文件

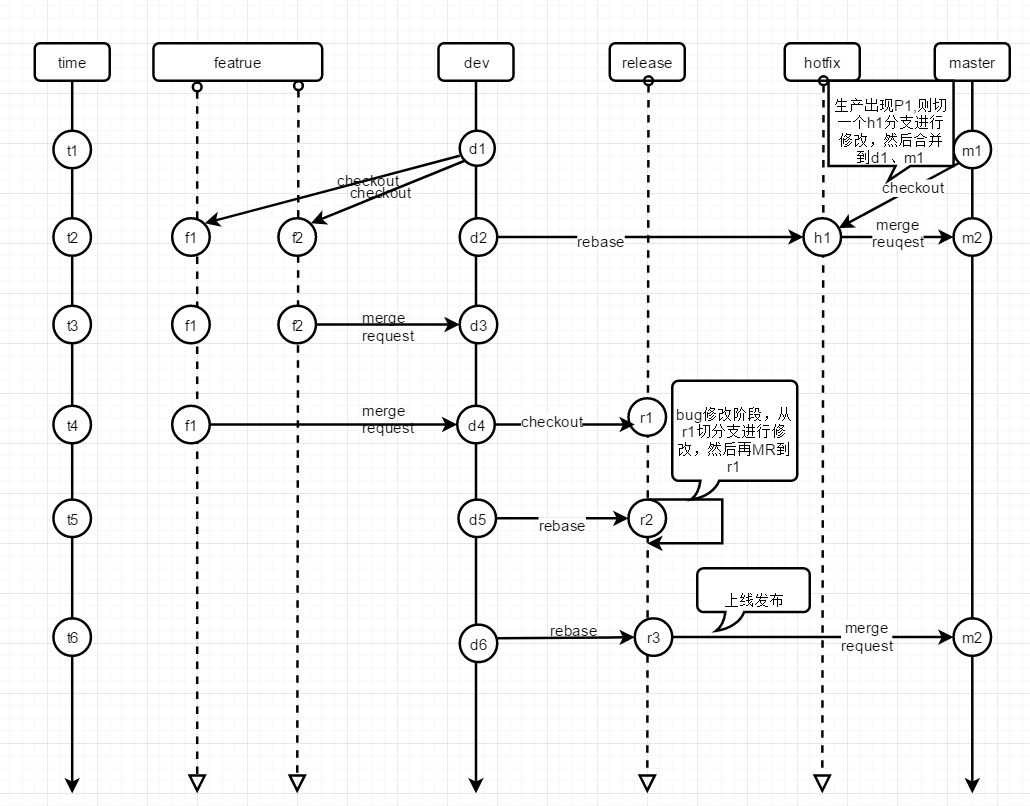
git add .git rabase –continue// 如果还有冲突，继续执行以上冲突

git push origin 【branch】 --force-with-lease

#### 5 GIT其他使用场景（欢迎大家完善补充）

# 代码发布流程

代码发布流程图：



说明：该图以时间线为导向

#### 正常开发发布流程:

##### 开发

t1: 迭代开发开始，d1代码与m1代码分支一致

t2: 有两个功能模块需要开发，分别从d1 checkout 分支为：f1、f2

t3: f2分支代码开发完成，merge request 到 d2

t4: f1分支代码开发完成，merge request 到 d3;

##### 测试

代码提测，checkout dev分支为r1

t5:测试r1分支代码出现bug,

a开发人员自行切对应的bug分支进行修改，

b并merge request到r2;

c完成所有的bug修复后需要dev rebase release 最新的代码，

##### 发布

t6: 上线发布:

1. 将release 代码 merge request 到 master
2. dev 需要再次 rebase release 最新的代码到dev

完成发布后需要删除对应的featrue 和 release 分支；只保留dev 和 master分支

#### Hotfix发布

说明：当线上生成环境出现P1 bug时，需要紧急修复上线。

t1:生产代码出现P1 bug 时；则从master切一个分支为hotfix 分支 进行修改；

t2:P1 bug 修复完成后：测试将对 hotfix 分支进行测试验证。测试通过后

1. merge request 到master 分支，进行上线发布
2. dev rebase hotfix 分支代码，保证之后的代码不会出现该P1bug

#### 特别注意：代码发布流程必须遵守以下约定

1. 同一个迭代功能代码必须同时上线发布，不支持部分上线发布；
2. 当前迭代未完成上线发布时，dev分支不接受当前迭代以外的代码合并请求；

# 分支命名规范

每个迭代开始时、或者当前迭代结束时；需要对gitlab上的分支整理，只保留dev和master分支，并保证dev分支和master分支代码一致；删除对应的featrue 、release、bug、hotfix分支。

Featrue 功能分支命名规范：

featrue\_issuesID\_上线日期。例如：featrue\_304\_20180420

BUG修改分支命名规范：

bug\_issuesID\_上线日期。例如：bug \_304\_20180420

hotfix紧急上线分支命名规范：

hotfix\_issuesID\_上线日期。例如：hotfix \_304\_20180420