

（一）分支规范

1.主分支

1. master ：随时可在生产环境中部署的代码
2. dev： 保存当前稳定并且最新的开发分支(多人开发同一分支)

2.辅助分支

1. bugFix: 用于线上紧急bug修复，生命周期截止到bug修复上线；
2. feature: 用于紧急需求开发上线；
3. dev\_local: 用于本地需求开发，不在下次上线范围内；

3.命名规则

1. feature分支:feat\_概述；
2. bug分支：bug\_bug概述；
3. dev\_loca: dev\_local\_概述；

（二）commit规范

1.commit 提交： 不建议单纯为了保存代码而commit，让你的commit记录更加清晰（在你本地代码出现问题需要回退时，可根据commit快速回到你出错前的阶段）

2.commit 注释：

1. 需求开发： feature+“-”+修改模块名称： 修改内容概述；
2. Bug修复： bugFix + “-”+修改模块名称： 修改内容概述；

注：可根据注释方便查看到本次提交的目的，以及涉及到的模块（当前多个项目在同一分支，难于维护，考虑拆分项目）；

3.多次commit合并为一次：利用 rebase 命令；

（三）push规范

1.master: 禁止直接push；

1.dev:确定你本地代码无误，提交后不会影响他人，确地本次提交在下次的上线内容中， 不要频繁的push；

2.bugFix: 多人在线同时开发，需要push remote；

3.feature: 多人在线同时开发，需要push remote；

4.dev\_local: 禁止push，提测时，合并到dev;

注：push后，需要提测时通知QA（当前专人部署， 以后QA自动化部署测试）， 多人同时开发测试时push，确定你本地代码无误，提交后不会影响他人，不要频繁的push；

（四）分支合并

1.bugFix、feature: 上线无误后， 在github web 项目界面提交merge request请求（通知到master管理员，及时确认merger request 请求），master merger请求确认后需要将当前 修改merger到dev（或者通知dev相关开发人员merge）， 给master打一个tag标签，以上都完成无误后删除当前分支;

2.dev：上线无误后，需要在github web 项目界面提交merge request请求（通知到master管理员，及时确认merger request 请求）， 给master打一个tag 标签；

3.dev\_local本地分支：开发新的需求，需要提测，merger到dev， push；

（五）注意点

Pull之前，最好把本地修改需要提交的提交，不需要的stash，方便merger出现问题，本地回退到前一次的commit；

Master 每次merge打Tag标签， 方便出现问题回退；

进行分支切换时， 将当前分支修改时将修改stash一下， 再次切换回来时unstash 回来；