Git版本策略

先锋主干多稳定分支

得到一个稳定版本后，将此稳定版本放到一个新分支上，针对此稳定版本的修修补补就在这个分支上进行，新功能不在此分支上开发，而在主干上进行新功能的开发。 这是业界采用较多的模式。

稳定分支上的有些修改，比如缺陷修复，需要合并到主干， 但有些特定修改，是不需要合并到主干的。这时需要千万注意，合并准确的文件到主干。

对于不能合并到主干的情况，常见的是再拉一个分支，这个分支专门为少数特定情况而用，但从全局讲，可能会导致太多分支，不同分支间混乱，所以这并不推荐。推荐宁愿采用配置开关。

开发约定

1、 请始终保持release分支为随时可以发布的稳定版本。

2、 请始终保持master分支为正在开发的版本。待master版本测试稳定之后在，在从该稳定版本，人为指定拉取release分支。

3、 上述两个分支，release、master均应为受保护分支。需要登陆gitlab，创建合并分支请求，待管理员同意之后，才会执行实际的合并操作。

4、 应当先在本地将代码同最新的分支，预先合并一下，自我检测，才开始向系统提交合并请求。提交代码。

5、 禁止在发生合并冲突时，通过强制合并来实现合并。应该人为的根据冲突出，自行调整后，避免冲突后。重新发起合并请求。

6、 个人开发使用的分支，请使用名字缩写+功能。如：xq-bugfix。

7、 尽量将commit -m 细化，并尽量做好该次commit 的内容。方便以后的版本查找和管理。

**权限**

不同角色，拥有不同权限，下面列出Gitlab各角色权限

**1. 工程权限**

| **行为** | **Guest** | **Reporter** | **Developer** | **Master** | **Owner** |
| --- | --- | --- | --- | --- | --- |
| 创建issue | ✓ | ✓ | ✓ | ✓ | ✓ |
| 留言评论 | ✓ | ✓ | ✓ | ✓ | ✓ |
| 更新代码 |  | ✓ | ✓ | ✓ | ✓ |
| 下载工程 |  | ✓ | ✓ | ✓ | ✓ |
| 创建代码片段 |  | ✓ | ✓ | ✓ | ✓ |
| 创建合并请求 |  |  | ✓ | ✓ | ✓ |
| 创建新分支 |  |  | ✓ | ✓ | ✓ |
| 提交代码到非保护分支 |  |  | ✓ | ✓ | ✓ |
| 强制提交到非保护分支 |  |  | ✓ | ✓ | ✓ |
| 移除非保护分支 |  |  | ✓ | ✓ | ✓ |
| 添加tag |  |  | ✓ | ✓ | ✓ |
| 创建wiki |  |  | ✓ | ✓ | ✓ |
| 管理issue处理者 |  |  | ✓ | ✓ | ✓ |
| 管理labels |  |  | ✓ | ✓ | ✓ |
| 创建里程碑 |  |  |  | ✓ | ✓ |
| 添加项目成员 |  |  |  | ✓ | ✓ |
| 提交保护分支 |  |  |  | ✓ | ✓ |
| 使能分支保护 |  |  |  | ✓ | ✓ |
| 修改/移除tag |  |  |  | ✓ | ✓ |
| 编辑工程 |  |  |  | ✓ | ✓ |
| 添加deploy keys |  |  |  | ✓ | ✓ |
| 配置hooks |  |  |  | ✓ | ✓ |
| 切换visibility level |  |  |  |  | ✓ |
| 切换工程namespace |  |  |  |  | ✓ |
| 移除工程 |  |  |  |  | ✓ |
| 强制提交保护分支 |  |  |  |  | ✓ |
| 移除保护分支 |  |  |  |  | ✓ |

PS: 关于保护分支的设置，可以进入Settings->Protected branches进行管理

**2. 组权限**

| **行为** | **Guest** | **Reporter** | **Developer** | **Master** | **Owner** |
| --- | --- | --- | --- | --- | --- |
| 浏览组 | ✓ | ✓ | ✓ | ✓ | ✓ |
| 编辑组 |  |  |  |  | ✓ |
| 创建项目 |  |  |  | ✓ | ✓ |
| 管理组成员 |  |  |  |  | ✓ |
| 移除组 |  |  |  |  | ✓ |

git开发流程规范

最近公司在由 svn 转为 git 做项目管理，之前在小团队开发的时候最多也就用到 dev 分支和 bugfix 分支，来到新的公司团队比较大，各自负责各自的功能模块，为了大家更规范的开发，公司参考了一些资料。制定了一下规则。

分支命名规则

主分支：master

开发分支：developer

功能分支：feature - 分支名称

分支发布：release - 版本号

bug 分支修复：bugfix - 版本号

操作步骤

管理员「项目负责人」创建 git 仓库，建立 developer 分支

git branch develop

git push -u origin develop

项目成员「开发者」clone 项目，在本地建立自己功能分支

git clone 项目 git 地址

git checkout -b develop origin/develop

创建本地功能分支

git checkout -b feature-[name-desc] develop

在自己的分支上进行开发 ： git add ，git commit 等，注意此时不要 push 到远程分支（origin）。

功能完成后可以直接合并本地的 developer 分支后 push 到远程仓库，合并的时候很大几率发生冲突，此时需要 merge ，merge的时候确保不影响项目其他成员，如果多个人都操作了同一个类，最好当面确认后在进行修改。等合并完成确认无误后，删除本地开发分支

git checkout develop

git pull origin develop //确保本地 developer 分支为最新的

git merge feature-[name-desc]

git push

git branch -d feature-[name-desc] //删除本地分支

发布分支

git checkout -b release-0.1 develop

一旦准备好了发版，合并修改到 master 分支和 developer 分支上，删除发布分支

合并修改到 master 分支

git checkout master

git merge release-0.1

git push

合并修改到 developer 分支

git checkout develop

git merge release-0.1

git push

删除发布分支

git branch -d release-0.1

为 master 分支打发版 tag

git checkout master

git tag -a 0.1 -m "Initial public release" master

git push --tags

bug 修复分支，如果正在开发功能的同时，developer 上发现了线上 bug，或者未上线的 bug，我们可以开一个 bugfix 分支来修复 bug

https://www.jianshu.com/p/e06973f3fae0?utm\_campaign=maleskine&utm\_content=note&utm\_medium=seo\_notes&utm\_source=recommendation