前言

为了方便开发人员后续追溯bug原因、版本发布、changelog自动生成、以及更好的团队开发协作，特制定此git使用规范。

分支介绍

- master分支
    - 所有提供给用户使用的正式版本，都基于这个分支发布
    - 开发人员不得在这个分支进行任何修改操作，此分支应当配置为受保护分支
    - 此分支为只读分支
- release/* 分支
    - 基于 develop 或 master 分支检出。用于准备发布新阶段版本或者修复线上bug版本
    - 此分支为预上线分支,用于给测试人员在测试环境进行测试
    - 此分支仅包含要上线的功能代码，不上线的功能禁止合并到release分支
    - 此分支为只读分支，测试通过后通过PR合并到master分支
- develop分支

    - 主开发分支，包含了项目最新的功能和代码，所有开发都依赖develop分支进行
    - 此分支为只读分支, 只能从master、release、feature分支合并过来，任何时候都不能在此分支修改代码

- feature/* 分支

    - 功能分支，开发新的功能或者改动较大的调整，从develop分支切出feature分支
    - 开发完后，提交Pull Request到develop分支合并，并删除feature分支

- hotfix/* 分支

    - 生产环境bug修复分支，基于master分支检出
    - 属于临时分支，当生产环境出现 bug ，管理员基于 tag 创建 hotfix/ 分支、release/<版本号>分支，由开发人员在hotfix分支修复bug，修复完成后，再向 release 分支提交 pull request 申请。bug修复完成上线之后可删除此分支

- bugfix/* 分支

    - 预上线环境 bug 修复分支，基于 release 分支检出
    - 此分支属于临时分支，当提测阶段中存在 bug 需要修复，由开发人员基于 release 分支创建 bugfix/ 分支，然后在 bugfix/ 分支进行修复 bug 。 bug 修复完成后，再向 release 分支提交 pull request 申请。bug修复完成 release 分支测试通过之后可删除此分支

分支命名规范

release分支

以待发布的版本号为分支名称，如 release/v1.0.0, release/v1.0.0-alpha.1

hotfix分支

以修复人的名称加上JIRA上的bugId，如 hotfix/wuhan-2879

bugfix分支

以修复人员的名字为分支名称，如 bugfix/wuhan

feature分支

以featureId加功能简单描述为规则，如feature/2879-add-user

git工作流规范

功能开发

开发前克隆develop分支到本地 `git clone -b develop http://xxx.git`

第一步：新建feature分支

从develop分支检出功能分支并切换到该功能分支 `git checkout -b feature/xxx origin/develop`

第二步：同步主开发分支最新代码

当功能分支开发一段时间后，主开发分支可能又有其他开发人员推送了最新代码，需要及时合并以免到最后出现大规模冲突代码

```
git fetch origin
git rebase origin/develop
```

第三步: 提交feature分支

每次完成一个小功能开发都要及时提交，不要等到全部功能开发完成后再提交所有。在推送代码之前先执行一下第二步，拉取最新代码到当前分支

提交代码

```
git add .
git commit -m "描述这次提交改动的内容"

准备推送前，执行一下第二步同步远程最新代码，首次推送代码到远程服务器使用-u参数，后面直接使用git push即可
git push -u origin feature/xxx
```

当功能分支开发完成后，发起PR请求合并到develop分支，在推送feature分支之前最好先在本地压缩下历史提交记录。

压缩提交

对git命令不熟悉的人可以跳过此步骤

`git rebase -i origin/develop`

当rebase操作完成后，执行`git push`命令推送到远程，最后发起PR请求让管理员合并到develop分支

bug修复

bug修复流程功能开发相同，当所有功能都开发完成后且都合并到了develop分支，然后管理员通过PR的形式合并到release分支，测试人员基于release分支在测试环境测试，开发人员基于release分支新建bugfix分支，bug修复完成后合并到release分支，测试阶段每个开发人员分配的Bug比较多，不要改一个合一个，多改几个再合并。

bug修复流程示例

```
拉取最新代码到本地
git fetch origin

从release分支检出新的Bug修复分支
git checkout -b bugfix/wuhan origin/release/v1.0.0
```

```
提交代码到本地
git add .
git commit -m "描述这次提交改动的内容"

准备推送到远程前，更新最新代码
git fetch origin
git rebase origin/release/v1.0.0

推送到远程，拉取最新代码后如果有冲突先修复冲突再提交推送
git push -u origin bugfix/wuhan
```

代码提交规范

规定格式如下：

```
<type>: <subject>#issue_number  // 这行表示提交的主题

<description> // 这行表示提交的详细说明
```

其中，type、subject是必需的，description 可以省略。issue_number如果jira上有对应的bug一定要加上，如果是自测发现的bug可以不加；不管是哪一个部分，任何一行都不得超过50个字符。这是为了避免自动换行影响美观。

type取值说明

type用于说明 commit 的类别:

- feat: (feature)增加新功能
- fix: 修复bug
- docs: 只改动了文档相关的内容
- style: 不影响代码含义的改动，例如去掉空格、改变缩进、增删分号(注意不是css修改)
- build: 构造工具的或者外部依赖的改动，例如webpack，npm
- refactor: 既不是修复bug也不是添加新功能的代码重构
- revert：回滚到上一个版本，执行git revert打印的message
- test: 添加测试或者修改现有测试
- pref: 优化相关，提高性能或者用户体验的改动
- chore: 其他不影响src源码的改动，例如构建过程或辅助工具的变动、增加依赖库

issue_number

jira上的bug号，用#号加bugID表示， 如#7897

subject

主题（subject）描述是简短的一句话，简单说明此次提交的内容，如果提交内容很多，一句话描述不清楚，在`<description>`描述块补充。

执行`git commit --amend`会弹出一个编辑提交说明的窗口，主题和描述之间要换行

提交示例

```
git commit -m "fix: 上传文件未校验文件类型#itsm-2378"

git commit -m "feat: 新增用户管理模块"
```

版本发布管理规范

版本号介绍

Alpha：Alpha是内部测试版,一般不会对外发布。

Beta: 该版本相对于Alpha版已有了很大的改进，消除了严重的错误，但还是存在着一些缺陷，需要经过多次测试来进一步消除。这个阶段的版本会一直加入新的功能。

RC: Release.Candidate的简写，就是发行候选版本。和Beta版最大的差别在于Beta阶段会一直加入新的功能，但是到了RC版本，几乎就不会加入新的功能了，而主要着重于除错! RC版本是最终发放给用户的最接近正式版的版本，是正式版之前的最后一个测试版。

Release: 在前面版本的一系列测试版之后，终归会有一个正式版本，是最终交付用户使用的一个版本。该版本有时也称为标准版。

版本格式：

主版本号.次版本号.修订号（X.Y.Z），版本号递增规则如下：

主版本号(major)：一般改动很大，不兼容低版本

次版本号(minor)：当你做了向下兼容的功能性新增

修订号(patch)：当你做了向下兼容的问题修正。

注意：高版次升级，低版次需要清零，如当前版本号是v1.0.2，需要将次版本号从0升级到1，升级后的版本号应该为v1.1.0；

版本发布

切换到生产分支，一般是master分支，先执行`git pull`命令拉取最新代码，然后执行打标签命令，例如`git tag -a v1.0.0`命令。执行这条命令会弹出一个交互式窗口，按字母i键进入编辑模式，按123数字顺序罗列这次更新的内容。然后按esc键退出编辑模式，再按`shift + :`键进入命令行模式，输入wq后按enter回车键保存退出。

如果更新的内容比较简单，可以一句话描述清楚这次发布更新的内容，也可以执行这样的命令`git tag -a v1.0.0 -m "描述这次发布更新的内容"`，通过-m参数表示这次发布的内容。加上-m参数不会进入交互式编辑模式。

> 注意：打标签一定要加上-a参数，表示附注标签

临时版本

临时版本号或者先行版本号，可以加到“主版本号.次版本号.修订号”的后面，作为延伸。格式为：

```
v1.1.0-alpha.1
v1.1.0-beta.1
v1.1.0-rc.1
```

推送标签

标签在本地打好后，需要推送到远程服务器。

```
// 推送单个标签
git push origin v1.1.0-alpha.1

// 推送所有标签
git push origin --tags
```

其他注意事项

1. 不要将本地配置文件提交到源代码管理中
2. 不要将第三方依赖项(如node_modules)提交到源代码管理中
3. 将你的git用户名配置为你的中文名字的英文全拼

参考链接

https://docs.gitlab.com/ee/topics/gitlab_flow.html

https://nvie.com/posts/a-successful-git-branching-model/

https://w3c.github.io/best-practices.html

https://w3c.github.io/git.html

https://feflowjs.com/zh/guide/rule-git-commit.html

https://www.ruanyifeng.com/blog/2015/08/git-use-process.html

https://jaeger.itscoder.com/dev/2018/09/12/using-git-in-project.html

https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

https://scicatproject.github.io/documentation/Development/Development_Methods.html

https://sethrobertson.github.io/GitBestPractices/

https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

https://reinh.com/blog/2009/03/02/a-git-workflow-for-agile-teams.html

https://zj-git-guide.readthedocs.io/zh_CN/latest/message/Conventional%E6%8F%90%E4%BA%A4%E8%A7%84%E8%8C%83/