鸟域道场

产品研发代码分支管理规范

（V1.0）

Nander 研发团队

2023 年 2 月

文档信息

文档编号				发布版本	
起草时间	2023-02-03	定稿时间		当前版本	V1.0
起草人	姓名	部门	电话	电子邮件	
	商鞅	鸟域行政司		26089183@qq.com	
审阅人			签发人		

文档修改记录

序号	修改时间	修改人	主要修改内容	存档版本
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

目录

1 目的

为了统一组织内开发人员和测试人员遵循相同的分支管理，支撑快速、高效、安全的迭代开发过程，并使相关人员能够快速理解和定位所需分支，特制定本规范。

2 适用范围

本规范适用于组织内所有自研产品与交付类项目的软件代码分支管理。

3 分支规则

3.1 分支类型

3.1.1 主分支

主分支贯穿整个项目的生命周期，自创建出来就不会被删除，会随着项目迭代不断地添加代码。主分支包含 master 分支与 develop 分支。

- **master分支**

主分支，代码是经过测试的、稳定的、处于可发布状态。master 分支是创建 Git 仓库时自动生成的。

- 命名规范：master
- 是否受保护：受保护
- 是否可 push 代码：不允许

- **develop分支**

开发分支，代码是相对稳定的、处于可测试状态。作为开发基线，确保多项功能并行开发过程中代码一致性。Git 仓库创建后，从 master 分支创建 develop 分支。

- 命名规范：develop
- 是否受保护：不受保护
- 是否可push代码：允许

3.1.2 支持分支

支持分支是为了程序员协同开发以及应对项目的各种需求而存在。当支持结束后，代码会合并回主分支中，支持分支一般会被删除。支持分支包含 feature 分支、release 分支、bugfix 分支和 hotfix 分支。

- **feature分支**

功能分支，用于开发新功能，从 develop 分支创建，支持结束后合并回 develop 分支。

■ 命名规范：feature_{模块/功能}
■ 是否受保护：不受保护
■ 是否可 push 代码：允许

- **release分支**

发布分支，从 develop 分支创建，基于此分支进行提测和测试，测试结束发布项目时创建新的 tag，并将代码合并到 master 分支和 develop 分支。

■ 命名规范：release_v{项目版本号}
■ 是否受保护：不受保护
■ 是否可 push 代码：允许

- **bugfix分支**

用于修复测试过程中发现 bug 的分支，从 release 分支创建，bug 修复后合并到 release 分支。

■ 命名规范：bugfix_v{项目版本号}
■ 是否受保护：不受保护
■ 是否可 push 代码：允许

- **hotfix分支**

用于修复线上紧急 bug 的分支，从特定版本的 tag 创建，接着修复 bug 和提测，测试通过发布时创建新的 tag，并将代码合并到 master 和 develop 分支。

■ 命名规范：hotfix_v{新版本号}

{新版本号}解释说明：

通常情况下，tag 版本号 Z 位递增后为新版本号，比如：tag 为 tag_v1.2.3,那么新版本号为 1.2.4，对应的 hotfix 分支命名为 hotfix_v1.2.4。

有时候将采用的新版本号已经存在，那么可以在原有版本号基础上追加后缀生成新版本号，后缀可以是bug的描述或部署环境的相关信息。

比如：智慧商企项目部署了版本为 1.2.3 的系统服务，之后，项目侧发现了 bug（dcoos 对接）需要修复，新版本号应该为 1.2.4，但是 tag_v1.2.4 已经存在，那么此次新版本可采用 tag_v1.2.3_dcoos,对应的 hotfix 分支命名为 hotfix_v1.2.3_dcoos。

■ 是否受保护：不受保护
■ 是否可 push 代码：允许

3.2 tag

tag 用于标识代码特定的版本，对应着 commit id。无论是功能迭代发布还是线上 bug 修复发布，都需要在支持分支上打 tag。

■ 命名规范：tag_v{支持分支对应的版本号}

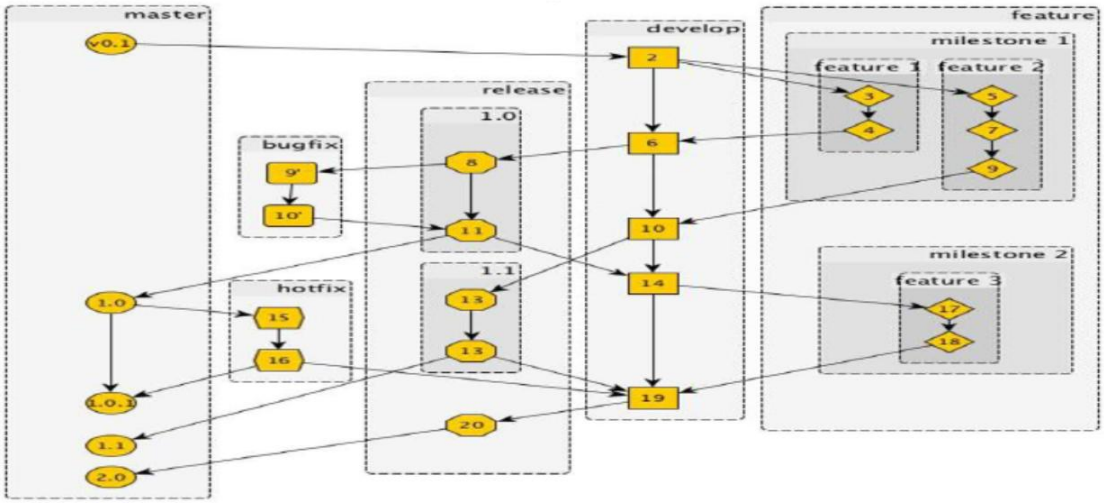
3.3 分支权限管理

Git 提供了三种角色来管理代码仓库：

Git 角色	权限说明
Master	可以添加tag、管理分支、仓库设置、克隆、提交、拉取代码。 为研发负责人、配置管理员分配 Master 角色。
Developer	可以克隆、提交、拉取代码。 为软件开发工程师分配 Developer 角色。
Visitor	只能浏览代码。 为项目经理、测试负责人等只需要代码查看权限的人员分 Visitor 角色。

3.4 分支使用流程

在产品研发过程中，项目成员通过执行相应的动作以达到对代码有效的管理。



场景	动作
建立新 Git 仓库	配置管理员创建 Git 仓库，基于 master 创建 develop 分支，将 master 分支设置为受保护，添加项目成员并分配 Git 角色。
开发一个模块或组件	软件开发工程师基于 develop 分支创建 feature 分支进行功能开发，开发完成后发起代码合并到 develop 分支请求和代码评审。评审组成员对代码进行评审。
联调	基于 develop 分支进行联调。
提测	配置管理员基于 develop 分支创建 release 分支，项目成员通过 release 分支进行提测和测试任务。
修复测试过程中的 bug	软件开发工程师基于 release 分支创建 bugfix 分支修复 bug，修复完成后发起代码合并到 release 分支请求和代码评审。
测试版本更新	评审组成员对代码进行评审，评审通过后将代码合并到 release 分支。
发版	配置管理员基于 release 分支创建 tag,将代码合并到 master 分

	支和 develop 分支。
修复线上 bug	软件开发工程师基于 master 创建 hotfix 分支进行线上问题修复，开发完成后提交代码并发起代码评审。评审组成员对代码进行评审。测试人员基于此分支进行测试。发版时，配置管理员基于 hotfix 分支创建 tag,并将代码合并到 master 分支和 develop 分支。

3.5 工具

通过分支模型使得代码版本得到有效的管理，但是涉及到多种分支手动创建和合并，造成操作复杂，为此，引入工具根据场景完成分支管理，达到降本增效。

3.5.1 gitflow-avh

提供了命令行方式，对Git基础命令进行了包装形成高级别的命令来满足分支模型场景中多分支的使用。Windows版本Git客户端已集成。

- 官网：gitflow-avh

3.5.2 Sourcetree

可视化的Git客户端，支持可视化操作分支模型。

- 官网：Sourcetree

3.5.3 Idea 插件：Git Flow Integration

基于gitflow-avh提供了可视化的操作页面。

- 官网：gitflow4idea

安装过程：

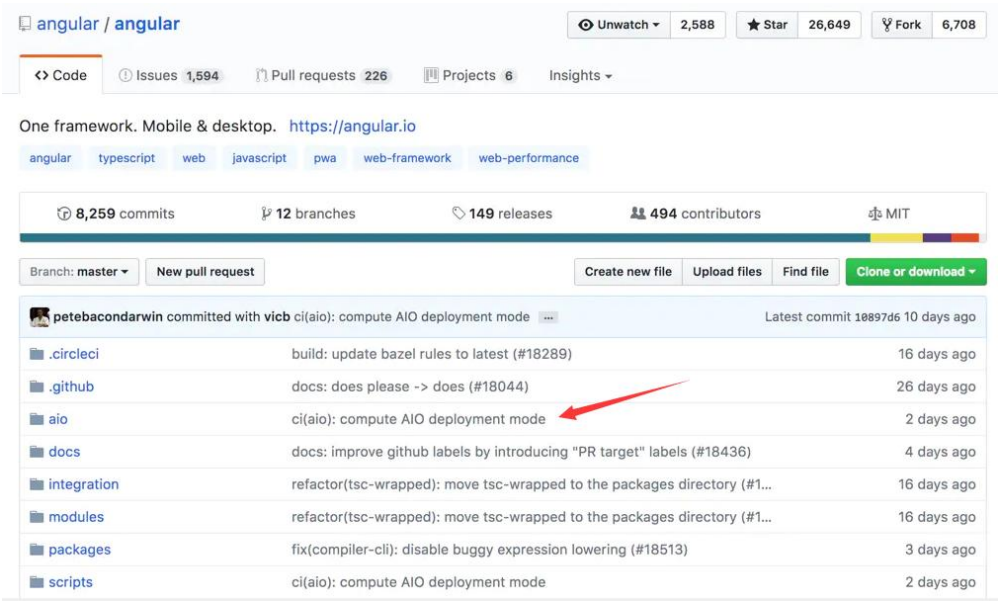
- 打开idea
- 打开settings下的plugin里搜索Git Flow Integration插件

4 Git Message 规范

修改代码并且提交之前，常会使用 git commit -m xxx 命令来描述我们代码改动的内容，但是很多不规范，随处可见 git commit -m `update`，以致于不能清晰的知道每次提交代码的变更内容，为了方便定位及管理，对于 message 进行规范化。

4.1 Angular 规范

按照该规范每次提交的Commit message效果如下图：



message包括三个部分：Header，Body 和 Footer。详细的格式定义如下：

```
# <header> 包含三部分：type(<scope>): <subject>

// 空一行

# <body>

// 空一行

# <footer>


// 具体说明如下：

# type 字段（必需）

# type 用于说明 commit 的类别，只允许使用下面标识。

# * feat：新功能（feature）

# * fix：修补 bug

# * docs：文档（documentation）

# * style：格式（不影响代码运行的变动）

# * refactor：重构（即不是新增功能，也不是修改 bug 的代码变动）

# * perf：性能（提高代码性能的改变）

# * test：增加测试或者修改测试

# * build：影响构建系统或外部依赖项的更改（maven,gradle,npm 等等）

# * ci：对 CI 配置文件和脚本的更改
```

```
# * chore：构建过程或辅助工具的变动，对非 src 和 test 目录的修改
# * revert：Revert a commit
```

最常用的就是 feat 和 fix 两种 type；

```
# scope 字段（可选）
# scope 用于说明 commit 影响的范围，比如数据层、控制层、视图层等等，视项目不同而不同。

# subject 字段（必需）
#   subject 是 commit 目的的简短描述，不超过 50 个字符。
# * 以动词开头，使用第一人称现在时，比如 change，而不是 changed 或 changes
# * 第一个字母小写
# * 结尾不加句号（.）

# body 字段
#   Body 部分是对本次 commit 的详细描述，可以分成多行。下面是一个范例。
#   More detailed explanatory text, if necessary.  Wrap it to
#   about 72 characters or so.
#   Further paragraphs come after blank lines.
#
#   - Bullet points are okay, too
#   - Use a hanging indent
#
```

日常项目开发中，如果 Header 中 subject 已经描述清楚此次代码变更的内容后，Body 部分就可以为空。

```
# footer 字段
#   Footer 部分只用于两种情况。
# * 不兼容变动；
# * 关闭 Issue
```

日常项目中开发，Footer 不常用，可为空。

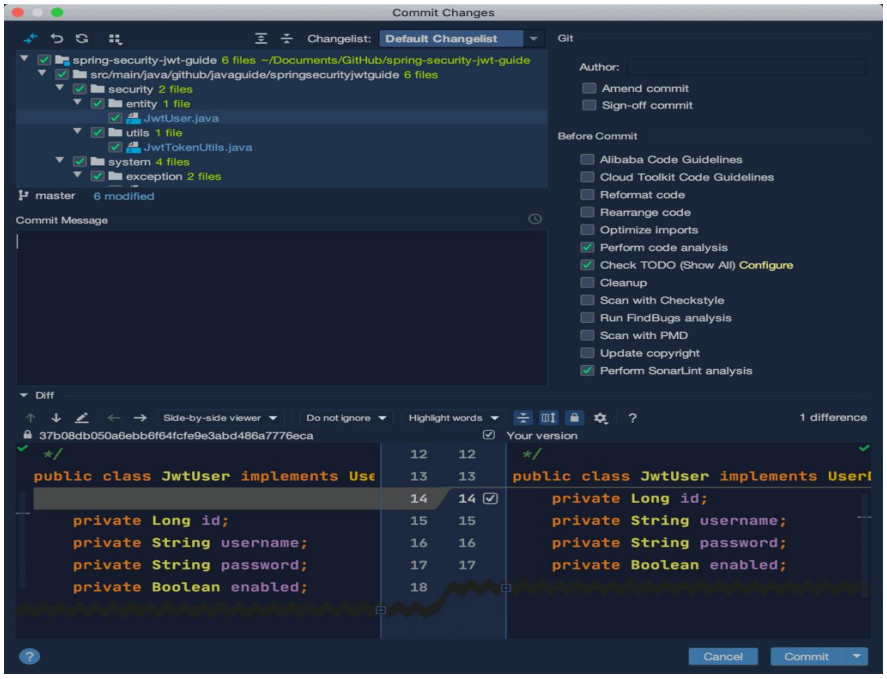
4.2 工具

通过 Git Message 书写格式遵循约定的提交规范，可以在 commit message 中直接查看代码变更对应的新特性、bug 修复、破坏性变更等类型，从而方便程序分析，比如：发起代码评

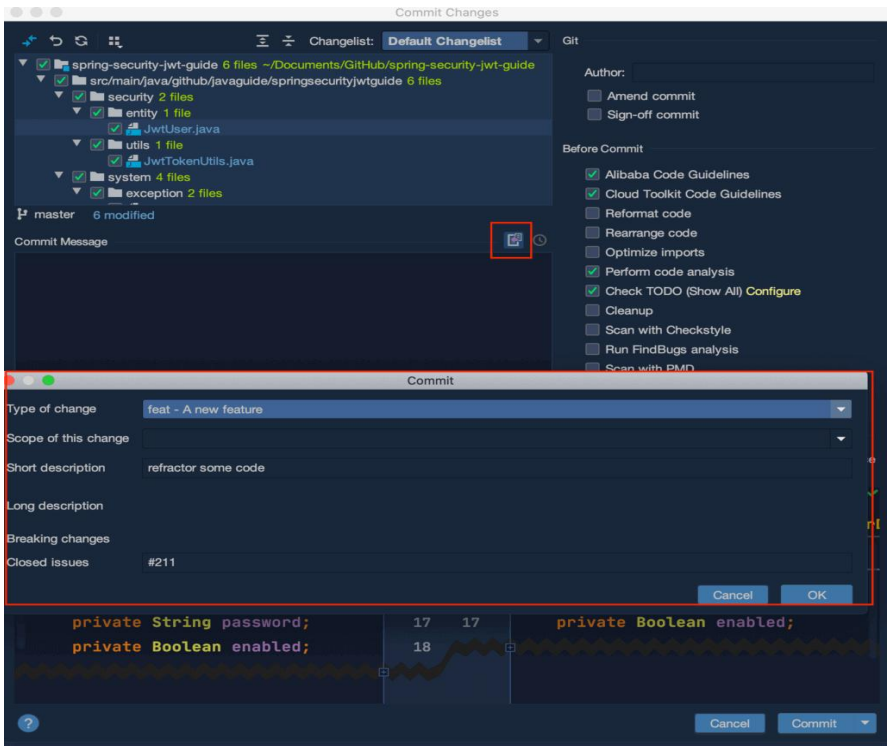
审时自动生成排版美观的描述信息。但由于涉及到多种内容段的创建，手工输入操作复杂，为此，引入工具完成 Git Message 书写，达到降本增效。

4.2.1 Git Commit Template 插件

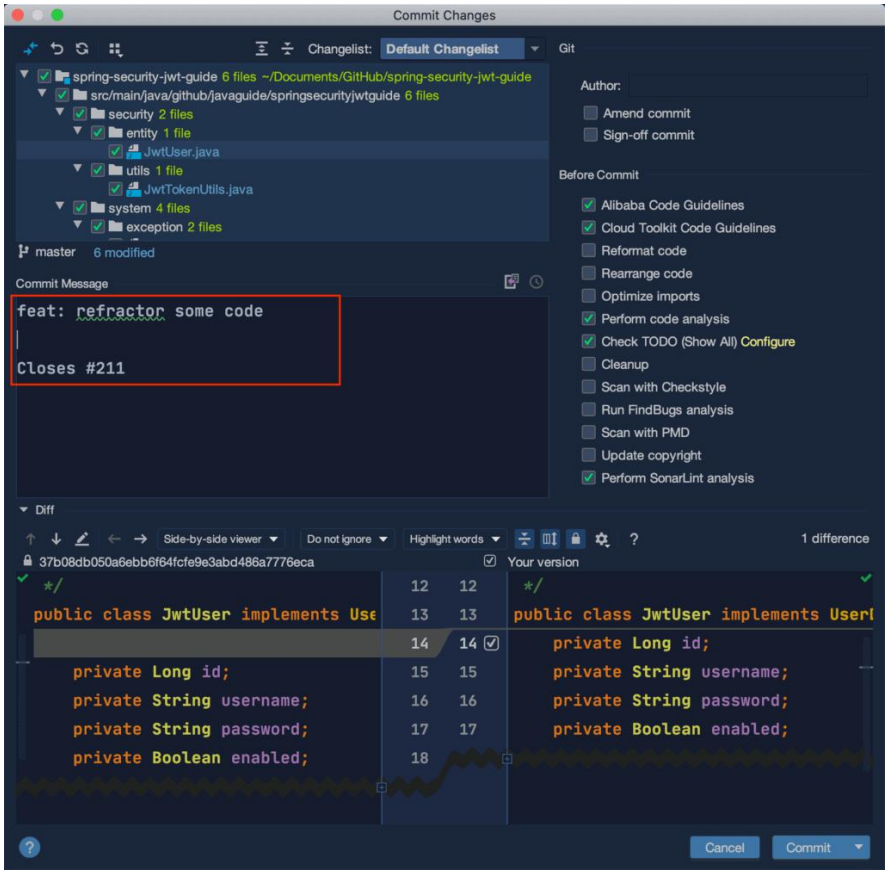
提供了可视化交互操作方式完成符合格式的Git Message录入。
没有安装这个插件之前，我们使用IDEA提供的Commit功能提交代码是下面这样的：



使用了这个插件之后是下面这样的，提供了一个 commit 信息模板的输入框：



完成之后的效果是这样的：



5 附则

- 本规范自发布之日起实施。
- 本规范由鸟域道场进行解释。
- 本文件著作权归北京鸟域花香科研室所有，用户除认真学习外不得以任何理由在未经同意的情况下擅自将文件用于其他用途。

6 参考文献

无