Branch Management Specification 分支管理规范

Version	Author	Time	Update	review	mark
V1.0	tim.zhao	2021-02-26	Created	zhiyong.liu	ok
V1.1	shaozhu.lin	2022-06-22	add version information	tim.zhao&zhiyong.liu	ok
V1.2	Sharon. Zhao	2022-08-18	add English translation		

一、Background 背景

统一供应链内部的研发流程，降低团队的管理成本，避免研发过程中的人为失误而造成事故。同时，统一规范后，对于后面的一系列的开发过程由系统完成，从而提高研发效率。

To unify the R&D process within the supply chain, reduce the management cost of the team, and avoid incidents caused by human errors in the R&D process. At the same time, after unifying the standard, the system will complete the following series of development processes, thereby improving the efficiency of research and development.

二、Definition of Branch分支定义

分支类型Type	Purpose 用途	Scenario 使用场景	Example 举例	Remark 备注
origin/test	对应test 环境 Corresponding Test Environment	保护分支 Protect Branch	——	
origin/uat	对应uat 环境 Corresponding UAR Environment	保护分支 Protect Branch	——	
origin/master		保护分支 Protect Branch	——	无实际工作意义 no actual meaning
origin/release	对应live环境 Corresponding Live Environment	保护分支 Protect Branch	——	
feature branch 分支	需求开发的分支 branch for requirement development	对应Jira Task Corresponding Jira Task	feature/SPLN-1000-add-something	
bugfix branch 分支	修复一个线上bug（非需求测试发现的）	对应Jira Bug Corresponding Jira Bug	bugfix/SPLN-1001-fix-something	跟随业务版本发布。 release according to the business version
个人开发分支 Individual Development Branch	个人开发工作，以主taskid+个人名字开头命名 Individual development work, named after the main taskid + personal name	对应Jira sub-task Corresponding Jira sub-task	feature/SPLN-1000 /zhangsan-add-something bugfix/SPLN-1000/lisi-fix-something	
hotfix branch 分支	线上问题紧急修复 solve live issues urgently	线上出现严重bug，紧急修复 serious online bugs require urgent fix	hotfix/SPLN-1000-fix-something	需要当天修复完成发布 Need to solve within the same day then release
adhoc branch 分支	业务紧急需求 urgent business requirement	业务出现严重问题，进行修复 serious business problem needed to be fixed	adhoc/SPLN-1000-add-something	需要快速修复完成发布 Need to solve ASAP then release
Version branch 版本分支	版本发布 version release	一个业务版本 a business version	version/uat-210126 version/release-201026	Uat version and live version
Temporary branch 临时分支	一般用于解决冲突 generally used for solving conflicts	feature分支上到test/uat时，有冲突需要解决 conflicts need to be solved	temp/fix-something	

tag	发布live时需要打tag add tag when releasing to live environment	发布live时需要打tag add tag when releasing to live environment	tag-201026-v1	

三、Branch Naming Convention 分支命名规范

分支名应该具备一定的可读性，由"分支类型和分支信息"组成，二者之间用'/'来切分。"/"用来分割层级，"-"用来分割描述内容。
The branch name should be readable to a certain extent, consisting of "branch type and branch information", separated by '/'. "/" is used to separate the hierarchy, and "-" is used to separate the description content.

分支类型就是以“feature”、“bugfix”、“hotfix”、“adhoc”、“temp”和“version”，发布的tag以“tag”开头命名

The branch type is "feature", "bugfix", "hotfix", "adhoc", "temp" and "version", and the released tag is named with "tag" at the beginning

分支信息主要来描述要做的工作，主task（bug）形式为：${jira}开头[+信息]，个人开发分支在主task(bug)的下面一级，形式为：个人名字[+信息]。信息只有英文。

The branch information is mainly used to describe the work to be done. The main task (bug) is in the form of: ${jira} at the beginning [+information], and the individual development branch is one level below the main task (bug) in the form: name[+information]]. Information is in English only.

feature/${jira}-add-something

feature/${jira}/{username}-add-something

hotfix/${jira}-fix-something

hotfix/${jira}/{username}-add-something

adhoc/${jira}-fix-something

adhoc/${jira}/{username}-fix-something

bugfix/${jira}-fix-something

bugfix/${jira}/{username}-fix-something

version/[uat/release]-${YYMMDD}

temp/fix-something

tag-${YYMMDD}-v{num}

本地安装githooks

Install githooks locally

四、Branch Management 分支管理

1.创建分支 Create Branch

release分支是基线分支 The release branch is the baseline branch

由于我们release分支对应线上（这一点与外界大多不一致，外界大多是master分支），我们平时开发需求、紧急修复都要从基于release分支进行。虽然，我们的master和release大多数时间是一致的，但也有不同的时候，我们统一从release分支拉。

Since our release branch corresponds to the line (this is mostly inconsistent with other organizations, which is mostly the master branch), our usual development requirements and emergency repairs must be carried out from the release branch. Although our master and release are consistent most of the time, there are also times when we pull from the release branch uniformly.

开发分支的创建Development branch creation

当我们工作编程的时候，如：开发一个需求，就需要在创建一个分支时，你可以在这个分支上尽情地尝试你的想法。你在分支上的任何改动都不会影响到主干分支，所以你可以尽情地做实验或者提交改动。如果你准备将它合并进主干分支，请让与你合作的同事review你的改动。

由于我们目前Jira管理工作内容方式是由sub-task来管理的，所以，我们的分支要和Jira的内容相互对应起来，如下图所示。

When we work on programming, such as: developing a requirement, we need to create a branch, and you can try your ideas on this branch. Any changes you make on the branch will not affect the main branch, so you can experiment or commit changes as you please. If you're ready to merge it into the main branch, ask your colleague to review your changes.

Since our current way of managing work content in Jira is managed by sub-tasks, our branches should correspond to the content of Jira, as shown in the following figure.

feature分支的创建Creation of feature branches

个人分支的创建Creation of personal branches

基于feature分支创建个人开发分支。注：即使是一个人开发或者只有一个sub-task，也要创建个人分支。

Create a personal development branch based on the feature branch. Note: Create a personal branch even if you are developing by one person or have only one sub-task.

Jira（工作任务）与Git（分支）的对应关系Correspondence between Jira (work task) and Git (branch)

和Jira task对应的是feature分支，和sub-task对应的是dev 个人开发分支。

Corresponding to the Jira task is the feature branch, and corresponding to the sub-task is the dev personal development branch

例外情况Exceptions

我们开发的需求依赖了一个没有上线的需求（假设在还在uat），我们依然从release分支切出feature分支，将被依赖的分支合并进来，基于合并后的分支进行开发。

注意：如果被依赖feature有发布或者延期风险，会造成feature不能按时发布，应尽量避免。

The requirements we develop depend on a requirement that is not online (assuming that it is still in uat), we still cut the feature branch from the release branch, merge the dependent branches into it, and develop based on the merged branch.

Note: If the depended feature has the risk of release or delay, it will cause the feature not to be released on time, which should be avoided as much as possible.

2.代码 Code commit

创建了分支之后，下一步就是做一些改动。当你添加、编辑或者删除了一个文件，都可以在当前的feature分支上添加一个Commit，这个步骤可以跟踪你的工作进度。

Commit也可以为你的工作创建一个透明的历史记录，这样可以帮助其他人去理解你做了什么、为什么这么做。每一个Commit都应该有一个关联的提交信息，用于描述为什么你做了这个特定的改动。

此外，每个Commit应该是一个拆分的独立单元。这可以帮助你在程序出现bug或者你决定从之前的某个地方重写的时候回滚你的改动。

$git commit -m “add some logic”

After creating the branch, the next step is to make some changes. When you add, edit or delete a file, you can add a Commit on the current feature branch, this step can track the progress of your work.

Commit also creates a transparent history of your work, which helps others understand what you did and why. Every Commit should have an associated commit message describing why you made that particular change.

Also, each Commit should be a split independent unit. This can help you roll back your changes if the program bugs or you decide to rewrite from somewhere before.

$git commit -m "add some logic"

3.代码Code push

自己程序写完自测后，可以push到自己的远端分支。

git push origin feature/SPLN-1000/{username}-fix-something

禁止force提交代码到远端。

After writing the self-test of your own program, you can push to your own remote branch.

git push origin feature/SPLN-1000/{username}-fix-something

Forbids force to submit code to the remote end.

4.代码merge request（MR）

只有mr才需要记录，commit历史不需要（gitlab有配置）

Merge Request用来将你的提交合并到feature分支，在合并到主分支之前，你的同事可以帮助您code review，并提出更改建议。

当我们的功能开发完成，需要提MR到feature主分支上。

注意，这里Merge到其他分支的代码，必须是要保证提交的代码是可以运行的。

带上feature的${jira}

Only mr needs to be recorded, commit history is not required (gitlab has configuration)

Merge Request is used to merge your commits into the feature branch. Your colleagues can help you with code review and suggest changes before merging into the main branch.

When our feature development is complete, we need to mention MR to the feature master branch.

Note that the code of Merge to other branches here must ensure that the submitted code can be run.

${jira} with feature

4.1.冲突的解决

这个时候能够如果发生冲突大多数发生在：

- 2+个同事一起开发一个需求，两个人代码有冲突
- Feature分支已经发生变更（比如：release分支发生变更，重新合并到feature分支）没有及时同步。

我们解决的方法是：

4.1. Conflict resolution

At this time, if a conflict occurs, most of it occurs in:

2+ colleagues develop a requirement together, and the two people have conflicting codes
The feature branch has changed (for example, the release branch has changed, and it has been re-merged into the feature branch) and has not been synchronized in time.

Our solution is:

5.Code review（CR）

当提交了MR后，就需要CR，可以指定具体模块的Owner或者项目组leader进行。

这里也是一个技术交流的地方，大家可以沟通技术和实现。

作为code review的人，也要为代码质量负责，不是简简单单的确定”Merge“。

目前，我们需要在需求开发完成向feature分支发起合并时，进行MR的code review工作。

5. Code review (CR)

When the MR is submitted, CR is required, and the owner of the specific module or the leader of the project group can be designated for it.

This is also a place for technical exchanges, where everyone can communicate technology and implementation.

As a code reviewer, you are also responsible for code quality, not simply "Merge".

At present, we need to perform MR code review work when the requirement development is completed and the merge to the feature branch is initiated.

6.Feature开发完成后提测

当功能开发完成后，开发需要完成自测，然后交由QA测试，将feature分支提MR到test分支。

6. Test after Feature development is completed

When the function development is completed, the development needs to complete the self-test, and then submit it to QA for testing, and the feature branch is MR to the test branch.

6.1.冲突的解决

将feature分支提MR到test分支时候可能会有冲突，但test分支包含了与该feature不一起发布的分支。

如果有冲突，参考下面的做法：

- 从release分支checkout一个临时分支temp
- 相关同事在temp分支上解决掉冲突
- 将temp分支合并到各自的feature分支
- 各自feature分支合并到test分支

6.1. Conflict Resolution

There may be conflicts when MR is raised from the feature branch to the test branch, but the test branch contains branches that are not released with the feature.

If there is a conflict, refer to the following practices:

checkout a temporary branch temp from the release branch
Relevant colleagues resolve conflicts on the temp branch
Merge the temp branch into the respective feature branch
The respective feature branches are merged into the test branch

7.Feature开发完成后交付UAT

当功能QA测试完成后，交由业务进行UAT测试：

- 不能确定uat版本的，将feature分支提MR到uat分支。

7. Deliver UAT after Feature development is completed

When the functional QA test is completed, the business is handed over to the UAT test:

If the uat version cannot be determined, MR the feature branch to the uat branch.

7.1.冲突的解决

如果有冲突，参考下面的做法：

- 从release分支checkout一个临时分支temp
- 相关同事在temp分支上解决掉冲突
- 将temp分支合并到各自的feature分支
- 各自feature分支合并到uat分支

7.1. Conflict Resolution

If there is a conflict, refer to the following practices:

checkout a temporary branch temp from the release branch
Relevant colleagues resolve conflicts on the temp branch
Merge the temp branch into the respective feature branch
The respective feature branches are merged into the uat branch

8.Testing或者UAT过程的修复bug

Feature提交测试后，QA进行测试会提的testing的bug；交付UAT，业务会提UAT的bug，对于这些bug的修复的修复流程如下。

另外情况，如果有些bug不立马修复，不跟这个feature一起上线，操作为：这个bug单独一个bugfix分支，后续跟着版本或者独立上线即可。

Fix bugs in Testing or UAT process

After the Feature is submitted for testing, QA will report the testing bugs for testing; when UAT is delivered, the business will report UAT bugs. The repair process for these bugs is as follows.

In other cases, if some bugs are not fixed immediately and do not go online with this feature, the operation is: this bug is a separate bugfix branch, and it can be launched with the version or independently.

9.发布

根据目前我们目前的研发流程，我们在周版本发布前才能确定要发布的内容，我们才可以说确定了一个版本，我们需要将这些版本的feature做一次合并（解决潜在的冲突），做回归测试，然后打tag再发布。

9. Release

According to our current R&D process, we can only determine the content to be released before the weekly version is released, and we can say that a version has been determined. We need to merge the features of these versions (to resolve potential conflicts) and do regression testing. , and then tag it before posting.

9.1 Create Version Branch 创建版本分支

9.1.1冲突的解决

如果冲突解决的方法如下：

9.1.1 Conflict Resolution

If the conflict is resolved as follows:

9.2回归测试

直接将version分支部署在staging环境，即可开始回归测试

9.2 Regression testing

Deploy the version branch directly in the staging environment to start regression testing

9.3合并release发布

- 将version分支提MR到release分支，记得MR时要带上这次版本所有的Jira号
- 根据公司的审计要求，我们发布必须打tag来发布，合并到release分支后，打tag就可以了。

$ git tag tag-20210105-v1

9.3 Merge release release
Bring the version branch to MR to the release branch, remember to bring all the Jira numbers of this version when MR
According to the company's audit requirements, we must tag to release the release. After merging into the release branch, tagging is enough.

$ git tag tag-20210105-v1

9.3.1冲突的解决

这个时候如果有冲突（很少发生），说明在version分支创建之后，release分支有了更新，这个时候需要

- 将release分支合并到该version分支
- 解决冲突
- 重新测试该version分支
- 测试通过后，该version分支MR到release分支，进行打tag发布

  9.3.1 Conflict Resolution

  If there is a conflict at this time (rarely), it means that after the version branch is created, the release branch has been updated.

  Merge the release branch into the version branch
  conflict resolution
  Retest the version branch
  After the test is passed, the version branch MR goes to the release branch for tag release

9.4基线(release)分支的重置

由于我们的开发分支都是以release为基础的，当一个版本发布后，release分支已经发生改变（当对于当时的开发分支），因此，我们需要将release分支合并到我们现在开发的分支，我们称为基线分支重置。

该过程会由DMS系统的工具来辅助，当release分支变更并且发布到线上：

- DMS会提MR到保护分支（test/uat/master)，需要开发进行合并处理
- DMS会在客户端githooks做检测，提醒开发人员重置基线分支

- feature分支MR到保护分支（test/uat/master)时，提醒开发人员重置基线分支

  9.4 Resetting the baseline (release) branch

  Since our development branches are all based on release, when a version is released, the release branch has changed (for the development branch at that time), therefore, we need to merge the release branch into our current development branch, we call it Reset for the baseline branch.

  The process will be assisted by the tools of the DMS system, when the release branch is changed and published online:

  DMS will mention MR to the protection branch (test/uat/master), which needs to be merged for development
  DMS will perform detection on client-side githooks and remind developers to reset the baseline branch
  When the feature branch MR reaches the protection branch (test/uat/master), remind the developer to reset the baseline branch

9.5发布失败 Fail to release

- 如果发布失败，应用做回滚之后，代码也要记得做回滚
- 发布失败，要周知到所有人：合作方（互相有依赖）、团队（中间可能有人拉了分支），
- 继续发布：先剔除有bug的分支再做发布，修复有bug的分支后再发布
- 要保证线上的代码和git的代码一致9.5 Failed to publish
  If the release fails, after the application is rolled back, the code must also remember to roll back
  If the release fails, it is necessary to know everyone: partners (depending on each other), team (someone may have pulled a branch in the middle),
  Continue to publish: remove the branch with bugs before publishing, fix the branch with bugs before publishing
  To ensure that the online code is consistent with the git code

10.额外的流程1-Live bug修复

我们线上（Live）出了bug，我们需要做修复。模式与feature开发基本类似，只是我们分支变为了bugfix来区分我们的内容，后续工作流程和feature开发一致。

10. Additional Process 1-Live bug fixes

We have a bug online (Live) and we need to fix it. The mode is basically similar to feature development, except that our branch becomes a bugfix to distinguish our content, and the subsequent workflow is consistent with feature development.

11.额外的流程2-hotfix

当线上（Live）出了严重的问题，需要紧急修复，我们称为hotfix。

- 我们从release拉一个hotfix分支，修改
- 该hotfix分支在staging环境做验证
- 合并到release分支进行发布

  11. Additional process 2-hotfix

  When there is a serious problem online (Live) that needs to be repaired urgently, we call it a hotfix.

  We pull a hotfix branch from release, modify
  The hotfix branch is verified in the staging environment
  Merge into release branch for release

12.额外的流程3-adhoc

业务需要紧急处理问题或者需求，需要紧急上线，我们称为adhoc。

- 我们从release拉一个adhoc分支，修改
- 该adhoc分支在staging环境做验证
- 合并到release分支进行发布

  12.additional process 3-adhoc

  The business needs to deal with problems or needs urgently, and needs to be launched urgently. We call it adhoc.

  We pull an adhoc branch from release, modify
  The adhoc branch is verified in the staging environment
  Merge into release branch for release

13.额外的流程4-发布回滚

1.发布回滚到某个tag后，如果可以紧急修复问题，可以走hotfix流程，但是如果无法快速恢复，需要先将release和其他相关分支的代码回滚到对应tag的改动，并且需要重新验证。

13. Additional Process 4 - Release Rollback

1. After the release is rolled back to a tag, if the problem can be urgently repaired, you can go through the hotfix process, but if it cannot be recovered quickly, you need to roll back the code of the release and other related branches to the changes of the corresponding tag, and need to re-verify .

14.冲突的解决

1.代码冲突解决原则：找到和你冲突的人去解决，不能单方面臆想解决。

2.冲突应该越早解决越好

1. 解决代码冲突过程需要注意的是，不能够随意拉取merge的目标分支来解决，需要考虑拉取目标分支的代码是否包含了其他代码，确保只有和你可以一起发布的代码，才可以做拉取操作，常见的比如test、uat（你不能够去拉取test/uat的代码，因为有别的功能代码在里面，你的分支将会被污染）。

4.解决冲突后，需要人工确认是否存在代码错误，有时候一个文件会有多个冲突，容易遗漏，可以使用IDE 或者工具或者git merge 全局看一下是否存在 <<< 这些或者一些代码报错高亮的地方。

解决冲突后代码需要重新跑一下自测/mock测试，毕竟代码变动过的都有可能出错（比如代码覆盖，或者说就消失了）。

5.冲突不应该是常态的，如果经常出现冲突，要思考几个问题：

1. 是否有人分支存在不规范操作，一般这种是大范围的冲突。
2. 是否代码文件代码耦合严重，需要做适当的拆分。

14. Conflict Resolution

1. Code conflict resolution principle: Find the person who conflicts with you to resolve it, and cannot unilaterally imagine a solution.

2. Conflicts should be resolved as soon as possible

In the process of resolving code conflicts, it should be noted that you cannot arbitrarily pull the target branch of the merge to solve the problem. You need to consider whether the code of the target branch to be pulled contains other code, and ensure that only the code that can be released together with you can be pulled. Operations, such as test and uat (you cannot pull the code of test/uat, because there are other functional codes in it, your branch will be polluted).

4. After resolving conflicts, you need to manually confirm whether there are code errors. Sometimes a file will have multiple conflicts, which are easy to be missed. You can use IDE or tools or git merge to see if there are <<< these or some code errors are highlighted The place.

After resolving the conflict, the code needs to re-run the self-test/mock test. After all, the code that has been changed may make mistakes (such as code coverage, or it will disappear).

5. Conflicts should not be the norm. If conflicts occur frequently, consider several questions:

Whether there are irregular operations in someone's branch, generally this is a large-scale conflict.
Whether the code file code is seriously coupled and needs to be split appropriately.

15.分支的管理和维护

对于发到线上的feature、hotfix、adhoc，bugfix分支，我们3个月定期清理。如果需要保留，可以做成保护分支。

对于发到线上的个人开发分支、版本分支，我们定期1个月清理。

对于上线的feature分支，不允许再进行修改。

15. Branch management and maintenance

For the feature, hotfix, adhoc, and bugfix branches sent to the online, we clean up regularly for 3 months. If you need to keep it, you can make a protected branch.

For personal development branches and version branches that are posted online, we regularly clean up every one month.

For the online feature branch, no further modification is allowed

五、注意事项 Matters need attention

- version分支也属于保护分支
- 所有的保护分支只能Merge
- 所有的保护分支（release除外）不能回合到开发分支（feature，hotfix，bugfix），
- 目前有些项目有xx国家环境，stable环境，业务项目组自行维护处理，可以与DMS系统集成做一些release分支重建的工作。
- The version branch is also a protected branch
- All protected branches can only be merged
- All protected branches (except release) cannot go back to development branches (feature, hotfix, bugfix),
- At present, some projects have xx country environment and stable environment. The business project team maintains and handles it by itself, and can integrate with the DMS system to do some release branch reconstruction work.