铁路应用软件源代码分支管理研究

王 喆[1]，丁晓萌[2]，刘承亮[1]

[1]中国铁道科学研究院集团有限公司电子计算技术研究所 北京
[2]中国国家铁路集团有限公司科技和信息化部 北京

【摘要】目的 当前铁路应用软件研发存在研发水平参差不齐、效能落后、源代码版本管理不统一等问题。本文从引入 Git 版本管理系统的必要性入手，在充分对比了三种重要的分支管理模型的各自优缺点的基础上，分析了铁路软件研发的特点，并基于主干开发的分支管理模型设计了适应铁路软件研发团队的分支管理策略。该策略在国铁集团网络安全与信息化管理系统的研发过程中得到了应用和实践。

【关键词】版本管理；分支模型；Git；主干开发；代码审查

【基金项目】中国铁道科学研究院院基金重点课题（2020YJ005）

Research on Software Source Code BranchesManagement of Railway Application Software

Zhe Wang[1], Xiaomeng Ding[2], Chengliang Liu[1]

Institute of Computing Technologies, China Academic of Railway Science Corporation Limited, Beijing, China

【Abstract】Objective At present, there are some problems in the research and development of railway application software, such as uneven research and development level, backward efficiency, and inconsistent source code version management. Starting from the necessity of introducing Git version management system, this paper fully compares the advantages and disadvantages of three important branch management models, analyzes the characteristics of railway software R&D, and designs a branch management strategy suitable for railway software R&D team based on TrunkBasedDevelopment. This strategy has been applied and practiced in the development of Network Security and Information Management System of China Railway Group.

【Keywords】Version Control; Branch Models; Git; Trunk Based Development; Code Review

企业的数字化变革越来越依赖信息系统的建设。为了应对不断变化的市场需求，必须尽可能的提高企业应用软件的研发效率和软件质量。敏捷开发模式大大提高软件交付速度的同时，也给软件协同开发提出了更高的要求，通常使用 SVN 或 Git 来进行软件源代码版本管理和控制（VCS，Version ControlSystem）。版本控制的任务是对特定目标在不同时期表现出的状态进行记录和维护，根据实际应用背景选择合适的版本间的拓扑结构[1]。Git 和 SVN 的主要差别在于对待数据的方式。从概念上来说，SVN 将其存储的信息看作是一组基本文件和每个文件随时间逐步累积的差异也被称作基于差异的版本控制。Git 更像是把数据看作是对小型文件系统的一系列快照。每当你提交更新或保存项目状态时，Git 会对当时的全部文件创建一个快照并保存这个快照的索引。如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件[2]。VCS 系统最重要的特性之一就是源代码分支的管理。MartinFowler 在他的文章“Patterns for Managing Source Code Branches”中这样定义分支：是由代码仓库中的若干提交组成的提交序列[3]。在 Git 中，采用合适的分支管理模型，使得开发者相互独立工作，提升了团队软件开发的效率。

本研究分析了 Git 中常用的几种分支管理模型，并基于铁路软件研发的特点，制定了铁路应用软件源代码分支管理模型以及相关技术方案，结合持续

集成和持续部署，在提升软件研发效率的同时保证了软件产品的质量。

1 基于 Git 的分支管理模型

1.1 Git 简介

Git 是当前非常流行的分布式版本管理系统。它包含了四个工作区域：工作目录（Workspace）、暂存区（Index）、本地仓库（Repository）、远程仓库（Remote），见表 1。文件在这四个区域之间按照一定的规则和操作流转，转换关系如图 1 所示。

在 Git 中，对分支的操作包括了创建、删除、查看、合并、变基等。分支类型一般包括长期分支和短期分支。长期分支贯穿软件研发的全生命周期，保存了项目的最稳定代码。短期分支通常为完成特定目标创建，目标达成且其自身合并至长期分支后即删除。基于上述 Git 分支的基本操作和特点，衍生出了诸多分支管理模型，下文挑选几个有代表性的进行介绍。

1.2 Gitflow

Gitflow 是在 Git 出现后不久，由 Vincent Driessen 在 2010 年提出的。在此后的十多年里，Gitflow 成为了广受软件研发团队欢迎但同时也备受争议的分支管理模型[4]。Gitflow 的基本思想如图 2 所示。在 Gitflow 的模型里，最重要的两个分支是 master 分支和 develop 分支，他们都是长时分支。Master 分支上的代码随时可以部署到生产环境，develop 分支作为每日构建的集成分支，到达稳定状态时可以发布并 merge 回 master。

表 1 Git 工作区域介绍

工作区域	描述
Workspace	工作区，即创建的工程文件
Index	暂存区，提交代码、解决冲突的中转站
Repository	本地仓库，连接本地代码跟远程代码
Remote	远程仓库，即保存代码的服务器

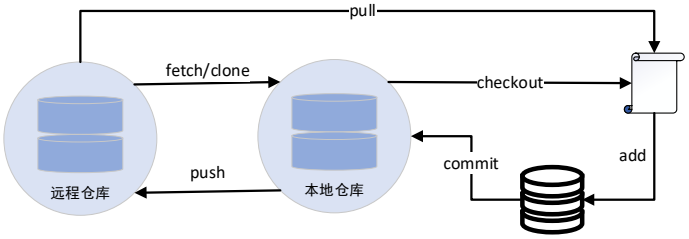

图 1 Git 中文件流转示意图

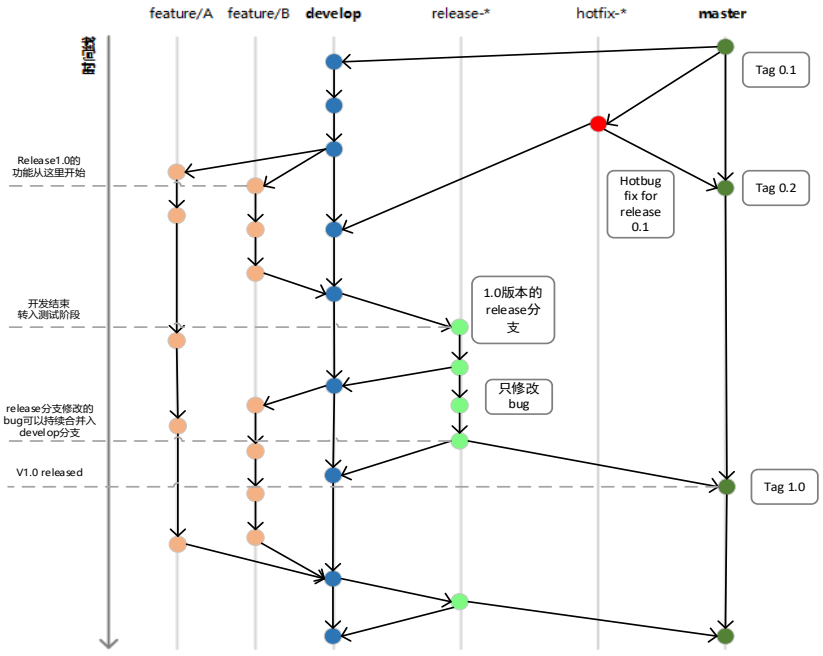

图 2 Gitflow 分支管理模型示意图

Gitflow 模型还包括了若干辅助分支（Supporting Branches），如 Feature 分支、Release 分支、Hotfix 分支。Feature 分支从 develop 分支创建，每个新特性都在独立的 feature 分支上进行开发，并在开发结束后 merge 回 develop 分支。Release 分支为每次发布准备，在这个分支上只进行缺陷修复，并在完成后 merge 回 master 和 develop 分支。Hotfix 分支用于快速修复，在修复完成后 merge 回 master 和 develop 分支。

1.3 GitHub Flow

GitHub 是一个面向开源及私有软件项目的托管平台，拥有超过千万的开发者用户。2011 年 GitHub 社区推出了基于其自身平台进行软件版本管理的分支模型即工作流 GitHubFlow。该模型主要包括如下 6 大原则：

（1）令 master 分支时常保持可以部署的状态

（2）开发新的功能需要从 master 分支创建新的功能分支，新分支名称要具有描述性

（3）在本地仓库分支中进行提交，并定期 push 分支代码至远程仓库的同名分支

（4）需要帮助、反馈，或者 branch 已经准备合并时，创建 Pull Request

（5）当其他开发者完成了代码审查并确认后，可以将本部分代码合并入 master 分支

（6）当代码与 master 分支合并后，立刻进行部署。

GitHub Flow 相比 Gitflow 来说较为轻量级，更加注重软件的持续部署，图 3 是 GitHubFlow 流程示意图。

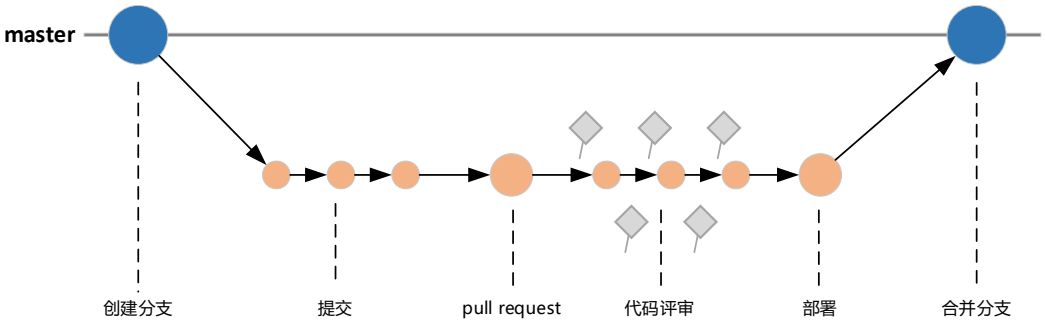

图 3 GitHubFlow 流程示意图

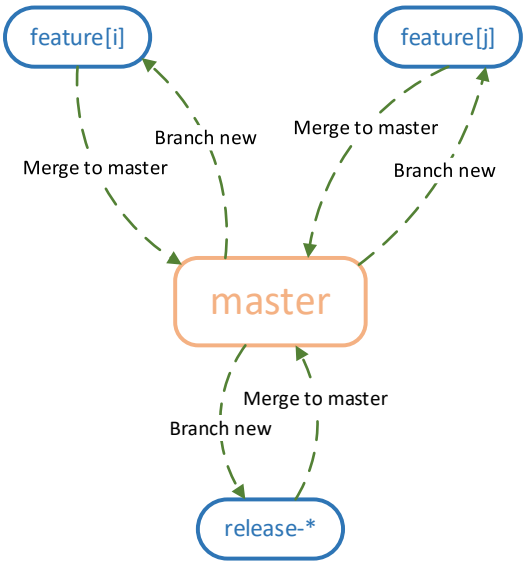

图 4 TBD 分支模型示意图

1.4 TBD

TBD（Trunk based Development），又被称作主干开发。该分支管理策略使得开发人员之间通过约定，向被指定为主干的分支提交代码，以此避免因为长期存在的多分支导致的开发压力。使用主干开发后，我们的代码库原则上就只能有一个 Trunk 分支即 master 分支了，所有新功能的提交也都提交到 master 分支上，保证每次提交后 master 分支都是可随时发布的状态。没有了分支的代码隔离，测试和解决冲突都变得简单，持续集成也变得稳定了许多，图 4 是 TBD 分支模型示意图。

由于只有一个主干分支，相对 Gitflow 和 GitHubFlow 简化了很多，易于流程的控制，也便于开发人员理解和操作。但是，由于其过于灵活，在实践中往往不同的团队采用不同的分支管理细节来适应自身团队特点。

1.5 分支模型比较

通过对三种分支管理模型的简要介绍，本文分析了三种分支模型的优缺点，如表 2 所示。

表 2 分支模型对比

	优点	缺点
Gitflow	流程清晰，覆盖面全 作为较早提出的分支模型，受众广泛 以 master 作为生产分支，面向单版本的线上产品迭代	复杂度较高，敏捷性较差 大部分工具默认将 master 分支设为默认分支，如果需要切换分支将导致流程繁琐 Hotfix 分支和 Release 分支设置繁琐，如开发人员疏忽将忘记合并回 develop 分支
GitHubFlow	流程简单，满足敏捷交付 无需频繁切换分支，在本地仓库开发，统一合并至 master 分支	对自动化测试要求较高，需要大量的单元、端到端和集成测试 模型过于简单，对于部署、发版和集成上存在着大量问题
TBD	频繁集成，每次集成冲突少，集成效率高 享受持续交付带来的好处 无需在分支间切换	工作在主干上的人员过多，将导致 bug 修复比较麻烦 为了保证不引入未完成功能，需要借助特性切换

依据通用分支管理模型各自的优缺点，企业应扬长避短，设计符合自身特点的分支管理模型。

2 铁路应用软件研发特点

铁路信息化建设从起步至今，经过了几十年的发展，建设了上千个铁路应用，覆盖战略决策、运输生产、经营开发、资源管理、建设管理、综合协同六大领域[7]，研发团队主要来自铁路内部企业，并承担了相关软件的迭代升级和后续研发工作。通过对上述软件研发团队的调研分析，总结铁路应用软件研发主要有如下几个特点：

（1）团队人员分散，水平参差不齐;

（2）研发效能相对落后;

（3）尚未普及 Git 版本管理;

综合以上分析，本研究认为铁路应用软件研发应采用 Git 作为版本管理系统，利用 Git 分布式代码管理的特点，解决位于不同地域的研发团队协同开发时代码提交和管理的难题。同时，结合 Git 中分支管理的便利性，选取简洁有效的分支管理模型（如 TBD）降低代码管理难度，并通过代码评审提升软件质量。

3 铁路应用软件源代码分支管理

3.1 分支类型

由于 Gitflow 相对复杂，对团队要求较高，GitHubFlow 需要以 GitHub 版本管理的依托，本研究采用相对简单、灵活的 Trunk Base 分支管理模型为基础建设铁路应用源代码分支管理策略。本策略中，代码分支包括 master、release 分支，feature 分支为可选。其中，master 分支为主分支，release 和 feature 分支为临时分支，阶段性创建并删除。除此之外，本研究还定义了关于分支模型的通用、缺陷修复、CI/CD、代码审查以及未完成功能处理等原则，共同构成源代码分支管理策略。图 5、图 6 分别是软件研发团队规模在 10 人及以下和 10 人以上时，采用本分支模型的基本策略。

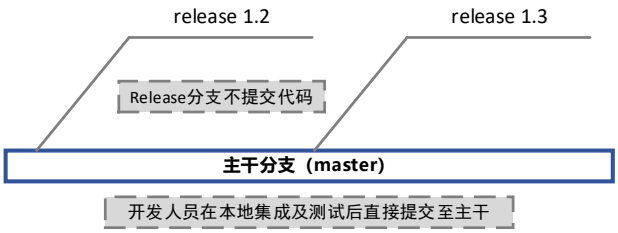

图 5 主干分支模型实现方案（10 人及以下团队）

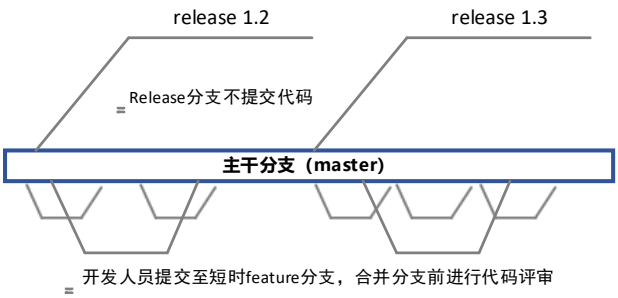

图 6 主干分支模型实现方案（10 人以上团队）

（1）Master 分支

master 分支作为主分支应是所有开发活动的核心分支，分支上存放的应是随时可供在生产环境中部署的代码。

（2）release 分支

当 master 分支上的代码已经包含了所有即将发

布的版本中所计划包含的软件功能，并且已通过所有测试时，应创建 release 分支：

①release 分支应由仓库管理员创建。

②release 分支上不应开展新功能的开发工作。

③release 分支创建时，应被及时赋予版本号，采用“release-版本号”命名规则。

④release 分支中不应包含不属于当前版本之外的软件功能。

⑤release 分支上应准备发布版本所需的各项说明信息（版本号、发布时间、编译时间等等）。

（3）Feature 分支

feature 分支可在开发一项新的软件功能的时候使用，应从 master 分支派生，由相关功能研发负责人创建分支。

①feature 分支不宜存在时间过长，功能开发、测试周期一般为两三天。

②feature 分支的命名应采用“feature-功能简述-研发人员姓名全拼”的规则，如“feature-login-zhangsan”。

③feature 分支代码可保存在开发者自己的代码库中而不强制提交到远程仓库里。

④feature 分支上的代码经过测试后，最终应合并回 master 分支，之后宜删除 feature 分支。

⑤feature 分支合并至 master 分支前，应首先开展代码审查工作，由代码审查员确认后再合并。

3.2 通用原则

通用原则定义了除分支类型之外最基础的原则，主要包括如下两条具体内容：

（1）在使用 git commit -m “[message]”提交代码时，message 应包括描述本次修改工作的主要描述，或者是项目管理平台中的任务编号。

（2）在执行分支合并前，应完成待合并分支的测试工作。

3.3 release 缺陷修复原则

按照 TBD 开发原则，一般不直接在 release 分支上提交代码。因此，当在 release 分支发现缺陷时，处理方法一般考虑两种情况：

（1）如果此时 master 分支还没有其他提交，可直接在 master 分支上修改缺陷然后合并至 release 分支；

（2）如果 master 分支已经有了提交，应提交 Cherry Pick 到 release 分支，如图 7 所示。

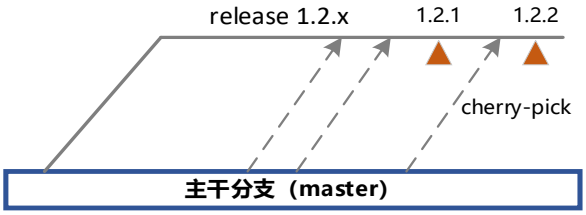

图 7 release 分支上修改 Bug 的方法

3.4 CI/CD 原则

持续集成（CI，Continuous Integration）是一种系统研发的迭代，团队中的开发人员通过一次又一次地将代码上传到代码仓库，进行集成测试和功能验证，从而实现项目与产品的迭代，不断地改善和解决问题。持续交付（CD，Continuous Delivery）是持续集成的一种补充及扩展，主要思想就是在自动化测试通过后，将软件及服务部署更新到生产环境[8]。为了充分发挥 CI/CD 的优势，在持续集成分支上应有如下原则：

（1）开发人员每日应至少合并一次代码至 master 分支，有利于基于主干开展持续集成和持续部署。

（2）开发人员在合并代码至 master 分支之前，应在本机执行完整的编译、单元测试、集成测试等。

3.5 代码审查原则

代码审查（Code Review）是软件开发过程中非常重要的环节，它不仅使得开发人员之间相互学习对方的长处和优点，更是增进代码质量的重要手段。在分支模型中，当涉及到分支合并时务必要开展代码审查工作，并且有如下原则：

（1）当有临时分支代码合并入 master 分支时，应首先开展代码审查工作。

（2）项目负责人宜在一天内完成分支合并代码的审查（包括代码编写质量、技术方案遵循性）并给出处理。

3.6 未完成功能处理原则

对于正在开发的功能，在将分支合并入 master 之前应加入功能特性切换（Feature Toggle）并将开关设置为关闭，避免将不成熟的功能引入发布版[9]；功能开发完毕并通过测试后，应将开关设置为打开；正式发布之前可移除开关功能，如图 8 所示。功能特性切换可以通过配置文件或者程序分支来实现。

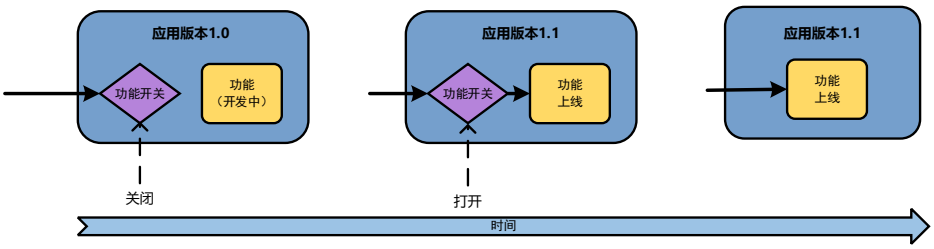

图 8 功能特性切换示意图

3.7 应用实践

上述分支管理原则在铁路网络安全与信息化管理系统研发过程中得到了实践检验。网信系统采用前后端分离架构，前后端代码分属不同的 Git 仓库中。研发团队包括前端研发 3 人，采用直接往 master 分支提交代码的方案；后端研发 8 人，成员由两地（北京、西安）3 部门组成，所以采用了带 feature 分支的方案。研发人员根据工作任务在本地仓库创建 feature 分支，定期提交至远程仓库，功能完成后申请与 master 分支合并。分支合并前由后端项目负责人完成代码评审。团队成员每天至少提交或者合并一次代码，在主分支开展持续集成、自动测试和自动部署，极大的提高了软件研发效率和质量。

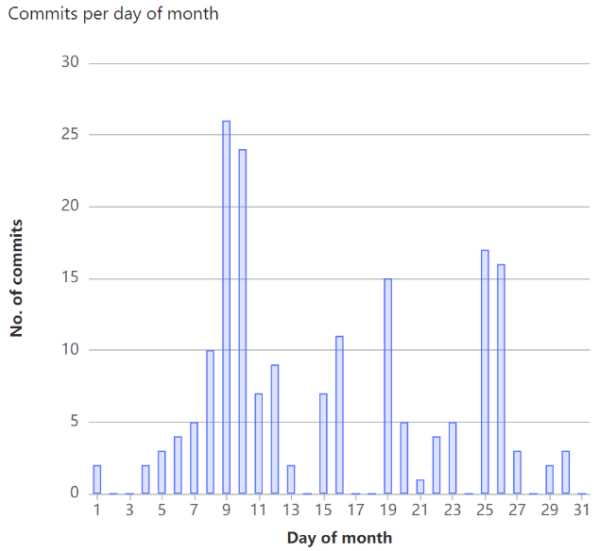

图 9 网信系统每日代码提交次数统计图（2021.2-2021.5）

4 结束语

在提出软件源代码分支管理必要性的前提下，简要介绍了 Git 版本管理系统，分析了目前比较流行的 Git 分支管理模型，包括 Git flow、GitHub Flow、TBD 三种。对铁路软件研发的特点进行了分析的基础上，有针对性的提出了适合铁路应用软件研发分支管理模型，详细介绍了该策略的分支类型、通用、缺陷修复、持续集成/部署、代码审查和未完成功能处理的原则。该分支管理模型已经在“国铁集团网络安全与信息化管理系统”的建设中进行了实践，取得了良好的效果。后续将努力在细节上对模型进行调整和完善，使其在更多项目中发挥效能提升作用。

参考文献

[1] 王真.版本控制工具在软件开发项目管理中的应用——以 GIT 为例[J].项目管理技术,2020,18(06):131-134.

[2] Scott Chacon.Pro Git 2nd Edition[M]. New York: Apress, 2014. 18-19.

[3] Martin Fowler.Patterns for Managing Source Code Branches [EB/OL]. https://martinfowler.com/articles/branching-patterns.html,2020-05.

[4] Vincent Driessen. [EB/OL]. https://nvie.com/posts/a-successful-git-branching-model.2020.03.05-2021.05.

[5] SCOTT CHACON. [EB/OL]. http://scottchacon.com/2011/08/31/github-flow.html.2011-08-31.

[6] Paul Hammant.[EB/OL]. Trunk-Based Development And Branch By Abstraction.2020-08-05

[7] 王喆,马小宁,邹丹,王沛然,孙思齐.基于铁路数据服务平台的铁路数据资产管理研究[J].铁路计算机应用,2021,30(03):23-26.

[8] 李正阳.系统研发的持续集成与持续交付技术的研究与实现[J].中小企业管理与科技(中旬刊),2021(03):177-180.

[9] Mahdavi HezavehRezvan,DremannJacob,Williams Laurie. Software development with feature toggles: practices used by practitioners[J]. Empirical Software Engineering, 2021, 26(1).

收稿日期：2021 年 6 月 12 日
出刊日期：2021 年 7 月 16 日
引用本文：王喆，丁晓萌，刘承亮，铁路应用软件源代码分支管理研究[J]. 国际计算机科学进展，2021，1(1):16-22
DOI: 10.12208/j.aics.20210006
检索信息：中国知网（CNKI Scholar）、万方数据（WANFANG DATA）、Google Scholar 等数据库收录期刊