Git 分支管理

基本介绍

Git Flow，**GitHub Flow** 和 **GitLab Flow** 是三种常见的 Git 工作流程，它们各自有不同的特点和适用场景。

1. Gitflow:
   Gitflow 是一种非常流行的工作流程模型，适用于大型软件项目。它基于 Git 版本控制系统，并**采用两个永久的分支：master 和 develop。master 分支用于发布稳定的版本**，而 develop 分支用于开发新的功能。除了这两个永久分支外，Gitflow 工作流还包括用于开发新功能的 feature 分支、修复 bug 的 hotfix 分支和预发布版本的 release 分支。
2. GitHub Flow:
   GitHub Flow 是一种简单且灵活的工作流程模型，特别适用于小型团队。它基于 GitHub 平台，并鼓励频繁的提交和部署。在 GitHub Flow 中，**只有一个主分支，即 master 分支，所有功能和修复都在这个分支上进行。**开发人员创建新的分支（通常以功能或问题的名称命名），并在该分支上进行开发和提交。一旦功能完成并通过了代码审查，就可以将其合并到主分支上。
3. GitLab Flow:
   GitLab Flow 是基于 GitLab 平台的工作流模型，结合了 Gitflow 和 GitHub Flow 的优点。它鼓励使用 master 分支来表示可用于生产的稳定代码，使用 feature 分支来开发新功能，使用 bug 分支来修复问题。与 Gitflow 不同的是，GitLab Flow 采用仅包含代码修改的分支，而不是完全分支。这样可以减少分支的数量和复杂性。

优缺点：

1. **Git Flow**[1][2][3]:
   - **优点**：清晰可控，为管理具有预定发布周期的大型项目提供了一个健壮的框架[1][2]。
   - **缺点**：相对复杂，需要同时维护两个长期分支（master 和 develop）。大多数工具都将 master 当作默认分支，但开发是在 develop 分支进行的，这导致经常要切换分支，非常烦人[2][3]。
2. **GitHub Flow**[4][5][6]:

    - **优点**：简单，适合代码部署非常频繁的团队和项目[4]。它的核心优势在于其流程带来的自动化可能性，能够做到其它流程无法实现的检查过程，并极大简化开发团队的体力劳动[5]。
    - **缺点**：因为只有一条 master 分支，万一代码合入后，由于某些因素 master 分支不能立刻发布，就会导致最终发布的版本和计划不同[7]。
3. **GitLab Flow**[8910]：
    - **优点**：GitLab Flow 是 Git Flow 和 GitHub Flow 的综合，既有适应不同开发环境的弹性，又有单一主分支的简单和便利[10]。
    - **缺点**：暂未找到明确的缺点，但需要注意的是，任何工作流程都需要根据团队的具体情况进行适当的调整和优化。

以上是 AI 生成。

参考链接：

1. 高效团队的gitlab flow最佳实践 - JadePeng - 博客园
2. 团队如何选择合适的Git分支策略？ - DevOps在路上 - 博客园
3. git flow 實戰經驗談 part1 - 別再讓 gitflow 拖累團隊的開發速度

分支管理

以上三种工作流都是常见的 git 分支管理方案，不过跟目前的分支管理由一定的出入，这里会根据实际情况重新确定管理方案。

- `master` 分支是唯一稳定的上游分支，开发人员不应该直接在 `master` 分支中进行代码的提交，版本记录以 `tag` 的形式进行， `tag` 标识里的版本号应该和迭代发布记录、线上对应版本保持一致；
- feature 分支是功能开发分支，该分支在开发时从 master 分支切出，如果该分支在进行多人协作的的话，各个开发人员会从该分支切出属于自己的开发分支，完成开发工作后合并到 `feature` 分支，整个 `feature` 分支完成开发、测试、修复后会进行发布（如果 master 分支有更新的话，需要先同步 `master` 分支），在发布结束会将 feature 分支合并到 master 分支并且打上相应的 tag；
- `hotfix` 分支是线上问题修复分支，该分支在线上出现问题时从 `master` 分支切出，完成测试、修复后进行发布，然后合并到 `master` 分支并且打上相应的 `tag` ；
- 分支名字应该具有语义化，比如说：
    - `feat-xxx-xxx` ： 功能开发

- `fix-xxx-xxx`：开发过程中的问题修复
- `hotfix-xxx-xxx`：线上问题修复
- `refactor-xxx-xxx`：代码重构

分支合并

- `git merge`：Git merge 的作用是将一个分支的更改合并到另一个分支，并创建一个新的合并提交。这意味着在目标分支上会生成一个新的合并提交，保留了源分支的完整历史记录。这种方式适合于公共分支，能够清晰地展示每次合并的历史记录，但可能会导致历史记录比较杂乱。
- `git rebase`：Git rebase 的作用是将一个分支的更改整合到另一个分支，并重新设置提交的基准点，使得提交历史变得更加线性。通过 rebase，可以将源分支的提交整合成一个新的提交，然后将这个新的提交应用到目标分支上。这样可以保持提交历史的整洁和线性，但可能会丢失一些合并的上下文信息。

总结：

1. Git merge 适合于公共分支的合并
2. Git rebase 则适合于私有分支的整合

参考链接：

1. https://joyohub.com/2020/04/06/git-rebase/
2. https://morningspace.github.io/tech/git-merge-stories-1/