链滴

开发流程和 git 使用笔记

作者：LeqiangBian

原文链接：https://ld246.com/article/1566906442926

来源网站：链滴

技术评审流程

基于RFC的

在XDF开发做重要功能点之前一般都会有一个技术评审流程，在开发重要的功能点之前，请发起Request For Comment (RFC)进行讨论:

1. 将 template.md复制到新文件并填写详细信息。提交这个RFC在你自己的repository fork。
2. 提交一个merge request (MR)请求到main repository。每个RFC有自己的MR请求，不要将RFC与其他文件修改关联。这个RFC MR的名称格式为RFC: <RFC名称>。
3. 相关人员通过PR review。在通过PR view之后，PR发起者把RFC的 status改成in-progress，更新RFC MR，合并MR到main repository。
4. 实现RFC描述的功能的issue名称格式为 RFC IMPL: <<RFC名称>>。如果已经完成了RFC中描述功能，将MR的status从in-progress更改为completed。

如果在第2步中没有办法进行MR操作，必须在master分支上撰写RFC，也可以通过issue来进行RFC的讨论。关联的issue名称为RFC: <RFC名称>。

RFC内容:

- 必要的数据库表设计
- 必要的性能指标

RFC状态:

- draft:
- in-progress:
- completed:

Refernces:

- CockroachDB RFC

基于石墨文档的

通过石墨文档的comment机制进行review

编码开发流程

Gitflow
template.md

说明：

1. 以上是简化后的Gitflow；
2. 须对于关键分支在GitLab上设置“分支保护”，设置“Allowed to push”为“No one”；
3. 在Gitlab上用Merge Request过程进行Code Review，须由非本人接受merge操作。

References:

- Git Feature Branch Workflow
- Gitflow Workflow
- [CockroachDB Contributing Guide]
(https://github.com/cockroachdb/cockroach/blob/master/CONTRIBUTING.md)
- [Kubernetes Developer Guide]
(https://github.com/kubernetes/community/tree/master/contributors/devel)
- grpc-java Contributing Guide

配置Git

保证你的系统上有Git Bash Shell。Windows系统从https://git-scm.com/download/win 上下载安装文件。MAC系统可以使用Brew进行安装。

在Windows的Git Bash Shell运行git config -l | grep autocrlf，运行结果应该是core.autocrlf=true。如果不是运行git config --global core.autocrlf true。

用新东方的企邮做如下配置，例如你的企邮是liumin@ff.cn，那么运行以下命令：

```
git config --global user.name "Min Liu"
git config --global user.email liumin@ff.cn
```

推荐和Git一起使用Beyond Compare。在Windows上安装完Beyond Compare 4之后，在Git Bash运行

以下命令：

```
git config --global diff.tool bc3
git config --global difftool.bc3.path "c:/Program Files (x86)/Beyond Compare 4/bcomp.exe"
git config --global difftool.prompt false
```

difftool.bc3.path可能需要根据你安装Beyond Compare的位置调整一下。

git difftool会使用配置好的Beyond Compare进行文本比较。

Git commit message format:

```
Short (72 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Further paragraphs come after blank lines.

  - Bullet points are okay, too

  - Typically a hyphen or asterisk is used for the bullet, preceded by a
    single space, with blank lines in between, but conventions vary here
```

关于如何写commit message，参见[Writing good commit messages](https://github.com/erlang/otp/wiki/Writing-good-commit-messages)。

编码规范

我们遵循Google Style Guides:

- Google Java Style Guide
- Google JavaScript Style Guide
- Shell Style Guide
- Google Python Style Guide
- Google HTML/CSS Style Guide
- Google Objective-C Style Guide
- Google C++ Style Guide

-Google TypeScript style

JSON的命名规范遵循JavaScript的，参见Camel case: defined。

其他各个模块可以根据自己的需要在此基础之上进行细化。

使用UNIX风格的UTF-8编码的文本文件，缩进采用两个空格。源代码的注释和git commit message量使

用英文，除非是要用中文解释业务。

详细的gitworkflow

先把master repository上的代码clone到本地，把origin改成upstream

```
git clone git@gitlab.xxx.com:demo.git
git remote rename origin upstream
```

在Gitlab上从main repository做一个fork repository, 在本地的git目录用以下命令加上fork remote：

```
git remote add fork git@gitlab.xxx.com:demo.git
```

在本地的git上建立一个branch，例如git checkout -b dev-leqiang

```
# 显示所有分支
git branch -a

# 切换分支和创建分支
git checkout -b branch_name tag_name

# 删除本地分支
git branch -d dev-leqiang

# 强制删除
git branch -D dev-leqiang

# 删除远程分支
git push origin --delete dev-leqiang

# 强制提交
git push -f origin

# 多个git commit合并成一个commit
git rebase -i HEAD~x

# rebase处理完冲突继续操作
git rebase --continue

# 提交单个commit到fork repository
git push <fork repostory> dev-leqiang

# 用以下命令和upstream的master保持同步
git fetch upstream master
git rebase upstream/master

# 查看当前分支树历史提交
git log

# git 回滚到某个版本，不保留修改
git reset --hard 5375a685e8cc163454c7e57087ffb6df41716fd3
```

```
# git 回滚到某个版本，保留修改
git reset --soft <HASH>

# 把所有没有提交的修改暂存到stash里面
git stash
git stash pop 恢复
```