支 `check` 出一个另外一个分支的历史来了解 `git rebase` 的工作原理。

* 首先，我们从主分支上 `check` 了一个 `feature1` 分支，并且在 `feature1` 分支中添加了一些功能，然后进行了 `commit` 提交；
* 接着，从 `feature1` 分支再 `check` 出 `feature2` ，然后对 `feature2` 分支也进行一些更改；
* 最后，回到 `feature1` 分支并提交更多更改；

整个交互的流程图如下：

假如我们需要将 `feature2` 分支的更改合并到发布的 `main` 分支中，并且不希望包含 `feature1` 分支的更改，可以使用下面的指令：

```
git rebase --onto main feature1 feature2
```

该指令用于重新定位 `feature2` 分支的基础，使其基于 `main` 分支，而不是 `feature1` 分支。这是通过将 `feature2` 分支上的提交应用到 `main` 分支之上来实现的。

指令执行结果如下：

代码审查工具：可以使用代码审查工具（如 GitHub Pull Requests）来审查合并分支前的提交，这样即使使用 git merge 也能清晰地了解每个特性的提交历史。
3. 混合使用

专题分支使用 `rebase`：在个人开发的专题分支上使用 `git rebase`，保持历史清晰。

整合分支使用 `merge`：在将专题分支合并到主分支时使用 `git merge`，保留上下文信息。
4. 命名和注释

良好的提交信息：无论使用哪种策略，确保提交信息清晰、有意义，可以帮助理解历史。

合并提交注释：在使用 `git merge` 时，可以在合并提交中详细描述合并的内容和目的，保留上下文信息。

总结
==

`git merge` 和 `git rebase` 都是 git 比较重要的指令，两个指令并没有绝对的好，也没有绝对的不好，平时使用时一定要注意每个指令的优缺点以及团队的抉择。

关于 git的更详细分析，参考往期文章：[美团一面：Git 是如何工作的？（推荐阅读）](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=MzIwNDAyOTI2Nw==&mid=2247491037&idx=1&sn=2351e0e54dcbf16a2575e353ed41734d&chksm=96c731e1a1b0b8f73027d20dc84e60760a195d46e106fcbddf98a945d125c453990f805a2769&token=108909832&lang=zh_CN#rd”)

学习交流
====

如果你觉得文章有帮助，请帮忙点个赞呗，公众号：`猿java`，持续输出硬核文章。

原文链接: https://juejin.cn/post/7392513610948640806