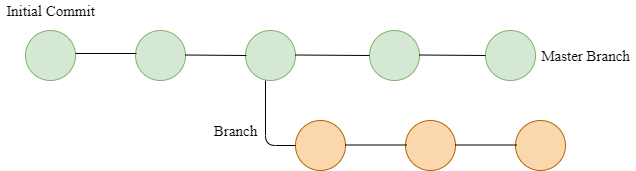
**Phần 6 Git Rebase**

Git rebase có thể tích hợp các thay đổi từ nhánh này sang nhánh khác bằng cách khắc phục các vấn đề mà chúng ta có thể gặp phải khi sử dụng lệnh git merge. Các thay đổi chúng ta sẽ thực hiện sẽ được ghi lại dưới dạng nhật ký hữu ích để xem lại nếu có bất kỳ lỗi nào xảy ra.

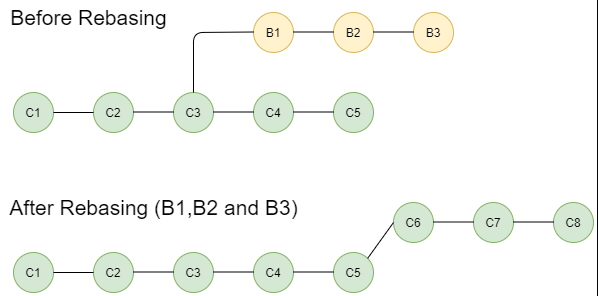
**Phân nhánh là gì?**

Phân nhánh có nghĩa là tách khỏi nhánh chính để bạn có thể làm việc riêng mà không ảnh hưởng đến mã chính và các nhà phát triển khác. Khi tạo [kho lưu trữ Git](https://www.geeksforgeeks.org/what-is-a-git-repository/) , theo mặc định, chúng ta được chỉ định [nhánh chính](https://www.geeksforgeeks.org/introduction-to-git-branch/) . Khi chúng ta bắt đầu thực hiện [các cam kết](https://www.geeksforgeeks.org/what-is-git-commit/) , nhánh chính này sẽ tiếp tục cập nhật và trỏ đến cam kết cuối cùng được thực hiện vào kho lưu trữ. Phân nhánh trong Git có thể hiển thị trong hình ảnh bên dưới.

****

**Git Rebase là gì?**

Rebase trong Git là quá trình tích hợp một loạt các commit trên một base tip khác. Nó lấy tất cả các commit của một nhánh và thêm chúng vào các commit của một nhánh mới. Git rebase trông như sau:

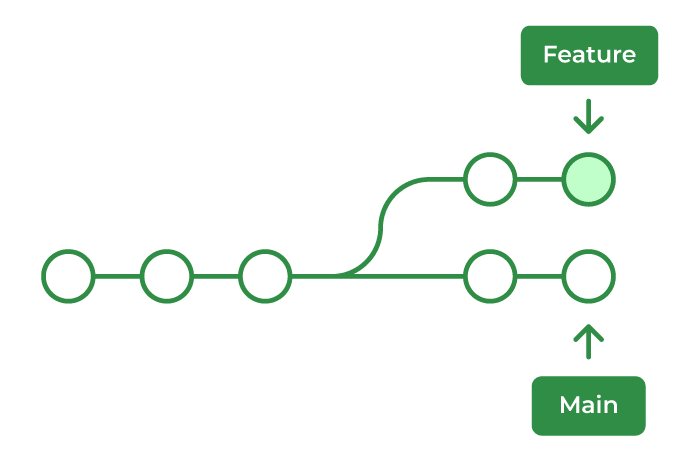


**git rebase [-i | --interactive] [tùy chọn] [--exec cmd] [--onto newbase | --keep-base] [ngược dòng [nhánh]]**

**Sử dụng Git Rebase**

Mục đích chính của rebasing là duy trì lịch sử dự án ngày càng thẳng và sạch hơn. Rebasing tạo ra lịch sử dự án tuyến tính hoàn hảo có thể theo dõi cam kết cuối cùng của tính năng cho đến khi bắt đầu dự án mà không cần phải fork. Điều này giúp bạn điều hướng dự án dễ dàng hơn.

Bạn có thể tích hợp nhánh tính năng vào nhánh chính theo hai cách. Cách đầu tiên là bằng cách hợp nhất trực tiếp vào nhánh chính hoặc trước tiên là rebase rồi hợp nhất. Sơ đồ bên dưới cho thấy Nếu bạn rebase nhánh tính năng trước, nó sẽ tạo điều kiện cho việc hợp nhất nhanh. Việc tích hợp các bản cập nhật ngược dòng vào kho lưu trữ cục bộ của bạn thường được thực hiện bằng cách rebase. Git merge khiến việc hợp nhất không cần thiết phải cam kết mỗi khi bạn muốn xem dự án đã tiến triển như thế nào khi bạn kéo các sửa đổi ngược dòng vào.



**Một số trường hợp sử dụng phổ biến của git rebase bao gồm:**

1. Duy trì lịch sử cam kết sạch và tuyến tính: Git rebase chủ yếu được sử dụng để duy trì lịch sử cam kết tuyến tính, trong đó các cam kết có liên quan đến cam kết đang tồn tại. Nó giúp dễ hiểu mã.
2. Cập nhật nhánh tính năng: Bằng cách rebase nhánh tính năng sẽ giúp chúng ta duy trì các bản cập nhật vì nó được tạo ra từ nhánh chính. Nhánh chính sẽ luôn được cập nhật.
3. Việc chuyển đổi nhánh tính năng có thể cập nhật nhánh này theo những thay đổi gần đây nhất trong nhánh chính nếu nhánh này được tạo từ nhánh chính (chẳng hạn như nhánh chính) và nhánh chính đó đã được cập nhật bằng các cam kết mới.
4. Giải quyết xung đột hợp nhất: Git rebase sẽ giúp chúng ta giải quyết [xung đột hợp nhất](https://www.geeksforgeeks.org/merge-conflicts-and-how-to-handle-them/) . Nó cho phép giải quyết xung đột ở mỗi giai đoạn, dẫn đến hợp nhất sạch hơn, bằng cách áp dụng từng cam kết từ nhánh được rebase riêng biệt.

**Git Standard so với Git Interactive Rebase**

Git rebase ở chế độ tương tác khi lệnh rebase chấp nhận đối số — I. Điều này có nghĩa là Tương tác. Nếu không có đối số nào, lệnh chạy ở chế độ Chuẩn. Để đạt được chế độ chuẩn rebase, chúng ta làm theo lệnh sau:

**git rebase master nhánh\_x**

trong đó branch\_x là nhánh chúng ta muốn chuyển đổi cơ sở

Lệnh trên tương đương với lệnh sau và sẽ tự động lấy các cam kết trong nhánh đang hoạt động hiện tại của bạn và áp dụng chúng vào đầu nhánh sẽ được đề cập:

**git rebase chủ**

Trong khi đó, trong Interactive rebase, bạn có thể thay đổi các commit riêng lẻ khi chúng được chuyển đến nhánh mới. Nó cung cấp cho bạn quyền kiểm soát hoàn toàn đối với lịch sử commit của nhánh. Để đạt được interactive rebase, chúng ta làm theo lệnh sau:

**git checkout nhánh\_x**

**git rebase -i master**

Lệnh này liệt kê tất cả các commit sắp được di chuyển và yêu cầu rebase tất cả các commit riêng lẻ rồi rebase chúng theo các lựa chọn bạn đã nhập. Điều này giúp bạn kiểm soát hoàn toàn giao diện của lịch sử dự án.

**Lệnh Git Rebase**

Sau đây là các lệnh Git rebase được sử dụng nhiều nhất:

1. git rebase master: Lệnh “git rebase master” có thể được sử dụng để biến mọi sửa đổi được tìm thấy trong nhánh master thành một phần của nhánh hiện tại.
2. git rebase –continue: Khi chúng ta rebase các nhánh, chúng ta sẽ gặp một số xung đột và vấn đề sau đó chúng ta cần giải quyết vấn đề. Sau khi giải quyết vấn đề, chúng ta lại tiếp tục các quy trình rebase để sử dụng “ git rebase –continue”.
3. git rebase –abort: Lệnh “Git rebase –abort” hủy lệnh rebase đang được thực hiện và khôi phục nhánh về trạng thái ban đầu.
4. git rebase –skip: Khi rebase các nhánh, chúng ta có thể gặp phải một số xung đột chưa được giải quyết để bỏ qua các cuộc chạm trán cụ thể, chúng ta sẽ sử dụng  “git rebase –skip”. Bỏ qua commit không phải là một thực hành tốt, nó sẽ làm hỏng codebase của bạn.
5. git rebase -I HEAD~3: Với sự trợ giúp của lệnh này, bạn có thể rebase tương tác ba commit gần đây nhất vào nhánh đang hoạt động. Bạn có thể chọn commit nào để rebase, thay đổi thông báo commit và squash hoặc chia commit trong trình chỉnh sửa tương tác được mở.

Hãy đảm bảo bạn đã sao lưu toàn bộ mã trước khi thực hiện lệnh git rebase; điều này sẽ cho phép bạn khôi phục mã cũ trong trường hợp có sự cố xảy ra.

Tùy chọn cấu hình trong Git Rebase

Chúng ta có thể tùy chỉnh hành vi của rebase theo yêu cầu của mình. Sau đây là một số tùy chọn cấu hình thường được sử dụng:

1. -I hoặc –interactive: Bằng cách sử dụng –interactive, chúng ta có thể thực hiện rebase một cách tương tác, trong đó chúng ta có thể chỉnh sửa hoặc commit. Bằng cách chọn tùy chọn này, trình soạn thảo văn bản sẽ mở ra, tại đó bạn có thể chọn commit nào để rebase. Bạn thậm chí có thể thay đổi thông báo commit hoặc kết hợp nhiều commit thành một commit duy nhất.
2. –onto <newbase>: Sử dụng tùy chọn này, một commit base mới được chỉ định cho quy trình rebase. Bạn có thể sử dụng nó để rebase nhiều commit vào một nhánh hoặc commit khác.
3. –skip: Nếu có tranh chấp phát sinh trong khi thực hiện thao tác rebase, tùy chọn này sẽ bỏ qua một commit. Nó được sử dụng để hướng dẫn Git bỏ qua một commit và thực hiện rebase.
4. –no-verify: Bất kỳ hook pre-commit nào đã được thiết lập trong kho lưu trữ đều bị bỏ qua bởi tùy chọn này. Nếu bạn cần commit nhanh mà không cần bắt đầu bất kỳ kiểm tra pre-commit nào, tùy chọn này có thể hữu ích.
5. –auto squash: Các cờ “fixup” hoặc “squash” trên bất kỳ commit nào được áp dụng ngay lập tức khi tùy chọn này được chọn. Khi sử dụng interactive rebase để xóa lịch sử commit, điều này có thể hữu ích.