**Phần 12 Chiến lược phân nhánh trong Git**

Các nhánh là các dòng công việc độc lập, bắt nguồn từ cơ sở mã gốc. Các nhà phát triển tạo các nhánh riêng biệt để làm việc độc lập trên các tính năng để các thay đổi từ các nhà phát triển khác không ảnh hưởng đến dòng công việc của một cá nhân. Các nhà phát triển có thể dễ dàng kéo các thay đổi từ các nhánh khác nhau và cũng hợp nhất mã của họ với nhánh chính. Điều này cho phép các nhà phát triển làm việc trên một cơ sở mã dễ dàng cộng tác hơn.

Chiến lược phân nhánh Git rất cần thiết cho việc quản lý mã hiệu quả và cộng tác trong các nhóm phát triển. Trong hướng dẫn toàn diện này, chúng ta sẽ đi sâu vào các chiến lược phân nhánh Git khác nhau, lợi ích của chúng, các bước triển khai và các biện pháp thực hành tốt nhất.

**Thuật ngữ chính**

* **Git Branch:** Phiên bản song song của mã trong [kho lưu trữ Git,](https://www.geeksforgeeks.org/what-is-a-git-repository/) cho phép phát triển và thử nghiệm riêng biệt.
* **Nhánh chính (trước đây là Nhánh chính):** Nhánh chính của kho lưu trữ Git nơi lưu trữ mã sẵn sàng để sản xuất.
* **Nhánh tính năng:** Một nhánh được tạo ra để làm việc trên một tính năng hoặc tác vụ cụ thể, tách biệt với nhánh chính.
* **Hợp nhất:** Quá trình kết hợp các thay đổi từ nhánh này sang nhánh khác.
* **Yêu cầu kéo (PR):** Yêu cầu do nhà phát triển đưa ra để hợp nhất các thay đổi của họ vào một nhánh khác, thường được sử dụng để đánh giá mã.
* **CI/CD Pipeline:** Pipeline tích hợp liên tục và [triển khai liên tục](https://www.geeksforgeeks.org/ci-cd-continuous-integration-and-continuous-delivery/) , tự động hóa quy trình xây dựng, thử nghiệm và triển khai các thay đổi mã.

**Chiến lược phân nhánh là gì?**

Chiến lược phân nhánh là chiến lược mà các nhóm phát triển phần mềm áp dụng để viết, hợp nhất và triển khai mã với sự trợ giúp của hệ thống kiểm soát phiên bản như Git. Nó đặt ra một bộ quy tắc hỗ trợ các nhà phát triển về cách thực hiện quy trình phát triển và tương tác với cơ sở mã được chia sẻ. Các chiến lược như thế này rất cần thiết vì chúng giúp giữ cho kho lưu trữ dự án được tổ chức, không có lỗi và tránh xung [đột hợp nhất](https://www.geeksforgeeks.org/git-merge-and-merge-conflict/) đáng sợ khi nhiều nhà phát triển đồng thời đẩy và kéo mã từ cùng một kho lưu trữ.

Gặp phải xung đột hợp nhất có thể cản trở việc phân phối mã nhanh chóng, do đó cản trở việc thiết lập và duy trì quy trình làm việc [DevOps](https://www.geeksforgeeks.org/how-devops-works/) hiệu quả. DevOps hướng đến mục tiêu tạo điều kiện cho quy trình phát hành nhanh chóng các thay đổi mã gia tăng. Do đó, việc triển khai chiến lược phân nhánh có cấu trúc có thể giảm bớt thách thức này, cho phép các nhà phát triển cộng tác liền mạch và giảm thiểu xung đột. Cách tiếp cận này thúc đẩy luồng công việc song song trong các nhóm, thúc đẩy phát hành nhanh hơn và giảm khả năng xảy ra xung đột thông qua quy trình được xác định rõ ràng để sửa đổi kiểm soát nguồn.

Chiến lược phân nhánh cung cấp các tính năng sau:

* Phát triển song song
* Năng suất được nâng cao nhờ sự hợp tác hiệu quả
* Phát hành tính năng có tổ chức và có cấu trúc
* Con đường rõ ràng cho quá trình phát triển phần mềm
* Môi trường không có lỗi mà không làm gián đoạn quy trình phát triển

**Từng bước thực hiện việc tạo nhánh**

Sau đây là các bước để tạo một nhánh:

**Bước 1: Tạo nhánh**

* Tạo một nhánh với tên bạn muốn chỉ định, ở đây chúng tôi đặt tên nhánh là “new-feature”.

git nhánh tính năng mới

**Bước 2: Điều hướng đến Chi nhánh**

* Bây giờ hãy điều hướng đến nhánh tính năng mới từ nhánh hiện tại bằng lệnh sau:

git checkout tính năng mới

( hoặc )

**Bước 3: Tạo và điều hướng nhánh tại một thời điểm**

* Lệnh sau chỉ giúp tạo nhánh và điều hướng đến nhánh.

git checkout -b tính năng mới

**Bước 4: Kiểm tra nhánh hiện tại**

* Thực hiện lệnh sau để kiểm tra nhánh hiện tại mà bạn đang sử dụng.

nhánh git

**Bước 5: Xóa nhánh**

Đảm bảo bạn có mặt tại nhánh bạn muốn xóa.

git branch -d <nhánh-cần-xóa>

**Các chiến lược phân nhánh Git phổ biến**

Sau đây là các chiến lược phân nhánh git phổ biến:

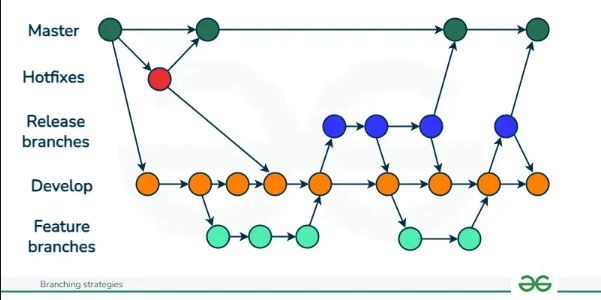
**Quy trình làm việc của Gitflow**

[GitFlow](https://www.geeksforgeeks.org/git-flow-vs-github-flow/) cho phép phát triển song song, nơi các nhà phát triển có thể làm việc riêng biệt trên các nhánh tính năng, nơi nhánh tính năng được tạo từ [nhánh chính](https://www.geeksforgeeks.org/git-origin-master/) . Sau khi hoàn tất các thay đổi, nhánh tính năng được hợp nhất với nhánh chính.

Các loại nhánh có thể có trong GitFlow là:

* **Master** – Dùng để phát hành sản phẩm
* **Phát triển** – Được sử dụng cho sự phát triển đang diễn ra
* **Phân nhánh tính năng** – phân nhánh từ nhánh phát triển để phát triển các tính năng mới.
* **Phát hành** – Hỗ trợ chuẩn bị bản phát hành sản xuất mới và sửa lỗi, thường được phân nhánh từ nhánh phát triển và đòi hỏi phải hợp nhất trở lại cả nhánh phát triển và nhánh chính.
* **Hotfix** – Các nhánh hotfix hỗ trợ giải quyết các lỗi đã phát hiện một cách nhanh chóng, cho phép các nhà phát triển tiếp tục công việc của họ trên nhánh phát triển trong khi vấn đề được giải quyết. Không giống như các nhánh phát hành, các nhánh hotfix được tạo từ nhánh chính dành riêng cho việc giải quyết lỗi quan trọng trong bản phát hành sản xuất.

Các nhánh Master và Develop là các nhánh chính và tồn tại trong suốt hành trình của phần mềm. Các nhánh khác về cơ bản là các nhánh hỗ trợ và tồn tại trong thời gian ngắn.



**Ưu điểm của Gitflow**

* Tạo điều kiện cho việc phát triển song song, đảm bảo tính ổn định của mã sản xuất trong khi các nhà phát triển làm việc trên các nhánh riêng biệt.
* Tổ chức công việc hiệu quả với các nhánh riêng biệt cho các mục đích cụ thể.
* Lý tưởng để quản lý nhiều phiên bản mã sản xuất.
* GitFlow hợp lý hóa quy trình quản lý phát hành, đẩy nhanh việc triển khai các tính năng mới và sửa lỗi.
* Bằng cách ủng hộ phát triển dựa trên tính năng thông qua các nhánh riêng lẻ, GitFlow thúc đẩy việc triển khai tính năng độc lập. Cách tiếp cận này cho phép hợp nhất liền mạch các tính năng vào cơ sở mã chính, giảm thiểu xung đột.
* GitFlow cung cấp quy trình được xác định rõ ràng để giải quyết lỗi và triển khai các bản sửa lỗi nhanh chóng, tạo điều kiện thuận lợi cho việc tích hợp nhanh chóng vào môi trường sản xuất.

**Nhược điểm của Gitflow**

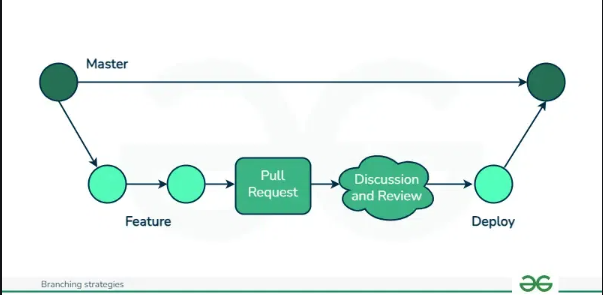
* Độ phức tạp tăng lên khi có thêm nhiều nhánh, có khả năng dẫn đến khó khăn trong quản lý.
* Việc hợp nhất các thay đổi từ nhánh phát triển vào nhánh chính đòi hỏi nhiều bước, làm tăng khả năng xảy ra lỗi và xung đột khi hợp nhất.
* Việc gỡ lỗi trở nên khó khăn do [lịch sử cam kết rộng rãi.](https://www.geeksforgeeks.org/git-changing-history/)
* Độ phức tạp của GitFlow có thể làm chậm quá trình phát triển và chu kỳ phát hành, khiến nó ít phù hợp cho việc tích hợp liên tục và phân phối liên tục.

**Dòng GitHub**

GitHub flow là một giải pháp thay thế đơn giản hơn cho GitFlow, ý tưởng dành cho các nhóm nhỏ hơn. [GitHub](https://www.geeksforgeeks.org/introduction-to-github/) flow chỉ có các nhánh tính năng bắt nguồn trực tiếp từ nhánh chính và được hợp nhất trở lại nhánh chính sau khi hoàn tất các thay đổi. Chúng không có nhánh phát hành. Khái niệm cơ bản của mô hình này xoay quanh việc duy trì mã chính trong điều kiện có thể triển khai nhất quán, do đó cho phép triển khai liền mạch các chu kỳ phát hành nhanh hơn, tích hợp liên tục và quy trình phân phối liên tục.

Các loại nhánh có thể có trong GitFlow là:

* **Master** – Quy trình làm việc GitHub Flow bắt đầu với nhánh master, chứa mã ổn định mới nhất được chuẩn bị để phát hành.
* **Tính năng** – Các nhà phát triển khởi tạo các nhánh tính năng từ nhánh chính để triển khai các tính năng mới hoặc giải quyết lỗi. Sau khi hoàn thành, nhánh tính năng được hợp nhất trở lại nhánh chính. Nếu xảy ra [xung đột hợp nhất](https://www.geeksforgeeks.org/merge-conflicts-and-how-to-handle-them/) , các nhà phát triển phải giải quyết xung đột đó trước khi hoàn tất hợp nhất.



**Ưu điểm của Github Flow**

* GitHub Flow nhấn mạnh vào tính năng phân nhánh nhanh chóng và hợp lý, chu kỳ sản xuất ngắn và phát hành thường xuyên, phù hợp với phương pháp Agile.
* Các nhóm có thể nhanh chóng xác định và giải quyết các vấn đề nhờ chiến lược tập trung vào vòng phản hồi nhanh.
* Việc kiểm tra và tự động hóa các thay đổi trên một nhánh duy nhất cho phép triển khai nhanh chóng và liên tục.
* GitHub Flow đặc biệt phù hợp với các nhóm nhỏ và ứng dụng web, nơi chỉ cần duy trì một phiên bản sản xuất duy nhất là đủ.

**Nhược điểm của Github Flow**

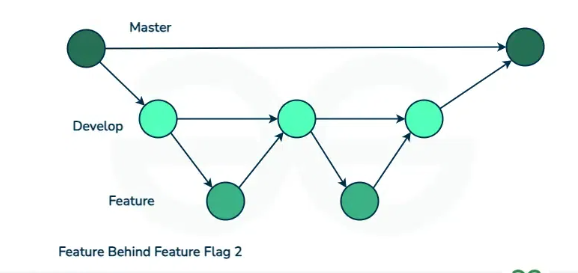
* GitHub Flow không lý tưởng để quản lý nhiều phiên bản của cơ sở mã.
* Việc thiếu các nhánh phát triển có thể dẫn đến mã sản xuất không ổn định nếu các thay đổi không được kiểm tra đúng cách trước khi hợp nhất.
* Nếu không có các nhánh phát triển riêng biệt, nhánh chính có thể trở nên lộn xộn, phục vụ cho cả mục đích sản xuất và phát triển.
* Khi các nhóm phát triển, xung đột hợp nhất có thể xảy ra thường xuyên hơn do mọi người hợp nhất các thay đổi vào cùng một nhánh. Thiếu minh bạch có thể làm trầm trọng thêm vấn đề này vì các nhà phát triển có thể không thấy những gì người khác đang làm.

**Luồng GitLab**

[GitLab](https://www.geeksforgeeks.org/bitbucket-vs-github-vs-gitlab/) flow cũng là một giải pháp thay thế cho GitFlow, được thiết kế để mạnh mẽ và có khả năng mở rộng hơn [GitHub](https://www.geeksforgeeks.org/introduction-to-github/) Flow. Được thiết kế cho các nhóm sử dụng GitLab, một trình quản lý kho lưu trữ Git dựa trên web, phương pháp này hợp lý hóa quá trình phát triển bằng cách tập trung vào một nhánh được bảo vệ, đơn độc, thường là nhánh chính. Tích hợp liên tục và thử nghiệm tự động tạo thành các yếu tố cốt lõi của GitLab Flow, đảm bảo tính ổn định của nhánh chính.

Các loại nhánh có thể có trong GitFlow là:

* **Bản chính:** Nhánh sản xuất chính chứa mã bản phát hành ổn định.
* **Phát triển:** Bao gồm các tính năng mới và sửa lỗi.
* **Tính năng:** Các nhà phát triển khởi tạo các nhánh tính năng từ nhánh phát triển để triển khai các tính năng mới hoặc giải quyết lỗi. Sau khi hoàn thành, họ tích hợp các thay đổi từ nhánh tính năng vào nhánh phát triển.
* **Phát hành:** Trước khi phát hành bản mới, một nhánh phát hành được tách ra khỏi nhánh phát triển. Nhánh phát hành này đóng vai trò là khu vực dàn dựng để tích hợp các tính năng mới và sửa lỗi dành cho bản phát hành sắp tới. Sau khi hoàn tất, các nhà phát triển sẽ hợp nhất các thay đổi từ nhánh phát hành vào cả nhánh phát triển và nhánh chính.



**Ưu điểm của Github Flow**

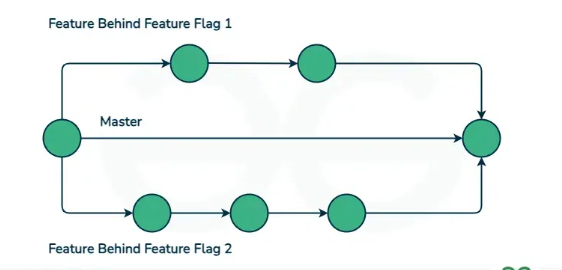
* GitLab Flow cung cấp chiến lược phân nhánh Git mạnh mẽ và có khả năng mở rộng, đặc biệt phù hợp với các nhóm và dự án lớn hơn.
* Cách tiếp cận này đảm bảo sự tách biệt rõ ràng giữa mã đang phát triển và mã sẵn sàng đưa vào sản xuất, giảm thiểu rủi ro vô tình thay đổi mã sản xuất.
* Với GitLab Flow, mỗi tính năng được phát triển trong nhánh riêng, thúc đẩy phát triển độc lập và giảm xung đột trong quá trình tích hợp vào cơ sở mã chính.
* Việc sử dụng các nhánh riêng biệt cho phép các nhà phát triển làm việc đồng thời trên các tính năng khác nhau, giúp phát triển tính năng nhanh hơn.

**Nhược điểm của Github Flow**

* GitLab Flow có thể gây ra nhiều thách thức do tính phức tạp của nó, đặc biệt là đối với các nhóm mới làm quen với Git.
* Việc sáp nhập các nhánh tính năng vào nhánh phát triển có thể dẫn đến xung đột vì các nhánh này có thể tách ra khỏi nhánh phát triển theo thời gian.
* Chiến lược GitLab Flow có thể làm chậm quá trình phát triển vì nó đòi hỏi phải hợp nhất các thay đổi vào nhánh phát triển trước khi phát hành. Điều này có thể gây ra vấn đề cho các nhóm yêu cầu phát hành nhanh các tính năng mới và sửa lỗi.

**Phát triển dựa trên Trunk**

Đây là chiến lược phân nhánh trong đó các nhà phát triển làm việc trên một nhánh "trunk" duy nhất, chủ yếu là nhánh chính và sử dụng cờ tính năng để cô lập các tính năng cho đến khi chúng sẵn sàng để phát hành. Nhánh chính này phải sẵn sàng để phát hành bất kỳ lúc nào. Không có nhánh bổ sung nào được tạo. Ý tưởng chính đằng sau chiến lược này là thực hiện các thay đổi nhỏ hơn thường xuyên hơn để tránh xung đột khi hợp nhất và mục tiêu là hạn chế các nhánh tồn tại lâu dài. Chiến lược này cho phép tích hợp và phân phối liên tục, khiến nó trở thành lựa chọn hấp dẫn cho các nhóm muốn phát hành bản cập nhật nhanh chóng và thường xuyên. Nó đặc biệt phù hợp với các dự án nhỏ hơn hoặc các nhóm đang tìm kiếm quy trình làm việc hợp lý.



**Ưu điểm của phát triển dựa trên Trunk**

* Phát triển dựa trên trunk giúp trunk được cập nhật liên tục, cho phép tích hợp liên tục các thay đổi về mã.
* Các nhà phát triển có thể thấy rõ hơn những thay đổi của nhau vì các cam kết được thực hiện trực tiếp vào trunk, thúc đẩy sự cộng tác và minh bạch.
* Không cần nhánh, khả năng [gặp phải xung đột hợp nhất](https://www.geeksforgeeks.org/merge-conflicts-and-how-to-handle-them/) hoặc "rắc rối hợp nhất" sẽ ít hơn vì các nhà phát triển thường xuyên đưa ra những thay đổi nhỏ, giúp đơn giản hóa việc giải quyết xung đột.
* Đường truyền chung luôn ở trạng thái có thể phát hành liên tục, cho phép phát hành nhanh hơn và ổn định hơn do công việc được tích hợp liên tục.

**Nhược điểm của phát triển dựa trên Trunk**

* Phát triển dựa trên trunk đòi hỏi rất nhiều quyền tự chủ và có thể gây khó khăn cho các nhà phát triển ít kinh nghiệm khi tương tác trực tiếp với trunk dùng chung, do đó, phương pháp này phù hợp với các nhà phát triển cao cấp.
* Phát triển dựa trên Trunk đòi hỏi mức độ kỷ luật cao và giao tiếp hiệu quả giữa các nhà phát triển để ngăn ngừa xung đột và đảm bảo phân lập đúng các tính năng mới.
* Khó quản lý đối với các nhóm lớn.
* Việc duy trì khả năng tương thích ngược với các phiên bản cũ hơn cũng có thể đặt ra thách thức.

**Chọn Chiến Lược Phân Nhánh Đúng Đắn**

Git cung cấp nhiều chiến lược phân nhánh, mỗi chiến lược phù hợp với các yêu cầu dự án và động lực nhóm khác nhau. Đối với người mới bắt đầu, nên bắt đầu với các phương pháp đơn giản hơn như GitHub Flow hoặc [phát triển dựa trên Trunk](https://www.geeksforgeeks.org/trunk-based-development-in-software-development/) , dần dần tiến tới các chiến lược phức tạp hơn khi cần. Đánh dấu tính năng cũng có thể giúp giảm nhu cầu phân nhánh quá mức. GitFlow có lợi cho các dự án yêu cầu kiểm soát truy cập nghiêm ngặt, đặc biệt là trong môi trường nguồn mở. Tuy nhiên, nó có thể không phù hợp với các hoạt động DevOps. Do đó, các nhóm tìm kiếm quy trình làm việc [Agile](https://www.geeksforgeeks.org/software-engineering-agile-software-development/) DevOps với sự hỗ trợ mạnh mẽ cho [tích hợp](https://www.geeksforgeeks.org/what-is-continuous-integration/) và phân phối liên tục có thể thấy GitHub Flow hoặc phát triển dựa trên Trunk phù hợp hơn. Cuối cùng, việc lựa chọn chiến lược phân nhánh phụ thuộc vào nhu cầu và mục tiêu cụ thể của dự án và nhóm.

| **Loại sản phẩm** | **Kích thước nhóm** | **Chiến lược áp dụng** |
| --- | --- | --- |
| Triển khai và phát hành liên tục | Bé nhỏ | GitHub Flow và TBD |
| Phiên bản phát hành theo lịch trình và định kỳ | Trung bình | GitFlow và GitLab Flow |
| Triển khai liên tục cho các sản phẩm tập trung vào chất lượng | Trung bình | Luồng GitLab |
| Sản phẩm có chu kỳ bảo trì dài | Lớn | GitFlow |