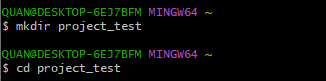
**CHƯƠNG V: CÁC KHÁI NIỆM CƠ BẢN TRONG GIT**

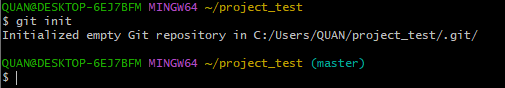
1. **KHỞI TẠO GIT REPOSITORY BAN ĐẦU**

Mục đích tạo ra một thư mục lưu trữ trong Git để chứa những dữ liệu cần thiết của 1 dự án, mỗi thành viên, đặc biệt là người quản trị phải thực hiện những tác động sau:

* **Bước 1:** Tạo thư mục cho repository của bạn trên máy tính.



* **Bước 2:** Khởi tạo Git repository trong thư mục bằng lệnh sau.



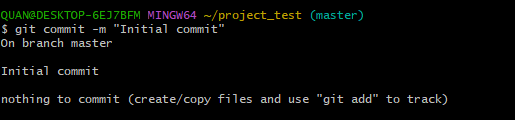
Lệnh này sẽ tạo 1 thư mục ẩn .git trong thư mục hiện tại để lưu lịch sử và thông tin về repository.

* **Bước 3:** Sau khi khởi tạo repository, ta có thể bắt đầu thêm các tệp và thư mục vào repository bằng lệnh git add. Nếu cần thêm tất cả các tệp và thư mục trong thư mục hiện tại vào repository, ta có thể sử dụng lệnh sau:



Dấu chấm (.) ở đây đại diện cho tất cả các tệp và thư mục trong thư mục hiện tại.

* **Bước 4:** Ta có thể tạo một commit mới để lưu trữ trạng thái hiện tại của repository bằng lệnh **git commit**. Lệnh này sẽ tạo ra một commit mới với các tệp và thư mục mà đã thêm vào repository ở bước trước đó. Nếu cần tạo commit với thông điệp "Initial commit", ta có thể sử dụng lệnh sau:



1. **COMMIT**

Trong Git, "commit" là một hành động để lưu trữ các thay đổi đã được thực hiện trên một repository vào một snapshot mới. Mỗi commit sẽ bao gồm một bản sao của tất cả các tệp tin trong thư mục làm việc của bạn tại thời điểm bạn thực hiện commit, cùng với một thông điệp để mô tả những thay đổi được thực hiện.

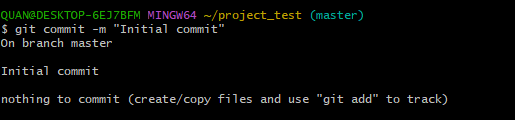
Một commit trong Git là một trạng thái của repository tại một thời điểm nhất định và được định danh bằng một mã hash duy nhất. Khi bạn tạo một commit, Git sẽ lưu trữ các thay đổi của bạn trên một nhánh (branch) riêng biệt, vì vậy bạn có thể quay lại trạng thái trước đó của repository nếu cần thiết.

Để tạo một commit, bạn cần thực hiện các bước sau:

* **Bước 1:** Sử dụng lệnh **git add** để thêm các tệp tin đã thay đổi vào staged area.



* **Bước 2**: Sử dụng lệnh **git commit** để tạo commit và ghi lại thông điệp mô tả các thay đổi đã được thực hiện.
* **Ví dụ:** Nếu bạn muốn tạo commit với thông điệp "Initial commit", bạn có thể sử dụng lệnh sau:



1. **XEM LỊCH SỬ COMMIT**
2. **Giới hạn thông tin đầu ra**

Ngoài các lựa chọn để định dạng đầu ra, **git log** còn nhận vào một số các lựa chọn khác cho mục đích giới hạn khác “-” là các lựa chọn cho phép bạn hiển thị một phần các commit. Bạn đã thấy một trong các tham số đó “-” đó là “-2”, cái mà dùng để hiện thị hai commit mới nhất. Thực tế bạn có thể dùng “-<n>”, trong đó n là số nguyên dương bất kỳ để hiển thị n commit mới nhất. Trong thực tế, bạn thường không sử dụng chúng, vì mặc định Git đã hiển thị đầu ra theo trang do vậy bạn chỉ xem được một trang lịch sử tại một thời điểm.

Tuy nhiên, tham số kiểu giới hạn theo thời gian như “--since” và --until khá hữu ích.

* **Ví dụ:** Lệnh này hiển thị các commit được thực hiện trong vòng hai tuần gần nhất:

**$ git log –since=2.weeks**

Lệnh này hoạt động được với rất nhiều định dạng - bạn có thể chỉ định một ngày cụ thể ("2008-01-15") hoặc tương đối như "2 years 1 day 3 minutes ago".

Cũng có thể lọc các commint thoả mãn một số tiêu chí nhất định.Tham số “--author” cho phép bạn lọc một tác giả nhất định, và tham số “--grep” cho phép bạn tìm kiếm các từ khoá trong thông điệp của commit. (Lưu ý là nếu như bạn muốn chỉ định tham số author và grep, bạn phải thêm vào “--all-match” bằng không lệnh đó sẽ chỉ tìm kiếm các commit thoả mãn một trong hai.)

Text

Description automatically generated

***Hình 5.1:* *Bảng liệt kê các lựa chọn trên và một số lựa chọn phổ biến khác***

* **Ví dụ:** Bạn muốn xem các commit đã thay đổi các tập tin thử nghiệm trong lịch sử mã nguồn của Git, được commit bởi Junio Hâmno trong tháng 10 năm 2008 mà chưa được tích hợp/gộp, bạn có thể chạy lệnh sau:

Text

Description automatically generated

Có gần 20,000 commit trong lịch sử mã nguồn của Git, lệnh này chỉ hiện thị 6 commit thoả mãn tiêu chí đặt ra.

1. **Hiển thị lịch sử trên giao diện**

Nếu bạn muốn sử dụng một công cụ đồ hoạ để trực quan hoá lịch sử commit, bạn có thể thử một chương trình Tcl/Tk có tên **gitk**  được xuất bản kèm với git. Gitk cơ bản là một công cụ git log trực quan, nó chấp nhận hầu hết các lựa chọn để lọc mà **git log** thường dùng. Nếu bạn gõ **gitk** trên thư mục của dự án, bạn sẽ thấy giống như hình bên dưới.

Graphical user interface, text, application

Description automatically generated

*Hình 5.2 Công cụ trực quan hóa lịch sử commit gitk*

1. **PHỤC HỒI**
2. **Thay đổ Commit cuối cùng**

Một trong những cách phục hồi phổ biến thường dùng khi commit quá sớm và có thể quên thêm vào đó một số tập tin. Muốn thực hiện lại commit đó, có thể chạy lệnh commit với tham số --amend :

**$ git commit –amend**

Trình soạn thảo văn bản xuất hiện để bạn thay đổi thông điệp của commit, nhưng nó đã chứa nội dung thông điệp của commit trước đó. Bạn có thể sửa nội dung như thường lệ, và nó sẽ được ghi đè lên commit trước đó.

* **Ví dụ:** Nếu như thực hiện xong commit và rồi sau đó mới nhận ra rằng đã quên tổ chức các thay đổi trong tập tin bạn muốn để thêm vào commit đó, có thể chạy lệnh sau:

Rectangle

Description automatically generated

Sau khi chạy ba lệnh này, kết quả cuối cùng cũng vẫn chỉ là một commit - commit thứ hai sẽ thay thế các kết quả của commit trước đó.

1. **Phục hồi tệp tin đã thay đổi**

Git status sẽ phục hồi lại những thay đổi trước đó- phục hồi nó lại trạng thái giống như sau khi thực hiện commit cuối hoặc như lúc mới đưa chúng vào thư mục làm việc. Trong thông báo đầu ra của ví dụ vừa rồi, khu vực tổ chức của chúng ta như sau:

Text

Description automatically generated

Nên cập nhật các phiên bản nâng cấp hơn để có thể sử dụng được những các chức năng có tính khả dụng cao hơn bằng cách sau:

Text

Description automatically generated

Bất kỳ thay đổi nào được thực hiện trên tập tin đó sẽ không còn nữa - bạn vừa mới sao chép một tập tin khác thay thế nó.

Bất cứ thứ gì đuợc commit vào Git luôn có thể phục hồi lại. Thậm chí cả các commit ở các nhánh đã bị xoá hoặc bị ghi đè bởi –amend.

1. **PHÂN NHÁNH**

Nhánh trong Git cũng giống như một cái cây có thân chính Master trong Git, từ nhánh chính này phát triển thành nhiều nhánh con, từ những nhánh con này đều phát triển “hoa”, “lá” và đều được gắn liền với nhánh chính. Các nhánh này đều có chung mục đích là làm cho cây của mục tiêu thêm phần sinh động và đẹp hơn.

Git lúc đầu có nhánh gốc là nhánh Master. Câu chuyện được đặt ra là một thành viên muốn thêm một tính năng cho dự án có sẵn, nhưng phần chỉnh sửa lại dễ ảnh hưởng đến dự án chính. Branch ra đời. Tính năng một nhánh Master gốc thành nhiều bản sao của cùng một Repository, cho phép người dùng chuyển đổi qua lại giữa các trạng thái và phiên bản khác nhau.

Từ những nhánh con đã phân tách, những người tham gia dự án có thể chỉnh sửa, thêm xóa chương trình của mình hợp lý, có thể kiểm tra quá trình hoạt động dự án ngay trên nhánh của mình mà không ảnh hưởng đến nhóm chính. Khi mọi sự hoàn tất người tham gia dự án có thể gộp nhánh của mình với nhánh chính cùng những tính năng mới để hoàn thiện dự án.

A picture containing diagram

Description automatically generated

*Hình 5.3: Quy trình phân chia nhánh*

* 1. **Làm việc với nhánh**
     1. **Tạo nhánh**
* **Bước 1: Kiểm tra nhánh**

Graphical user interface

Description automatically generated with medium confidence

Chỉ có một nhánh Master (nhánh gốc)

* **Bước 2: Phân nhánh**

Dùng lệnh git branch +<Tên nhánh>



* **Bước 3: Kiểm tra nhánh đã tạo**

Graphical user interface

Description automatically generated with low confidence

Như vậy nhánh b1 đã tạo thành công.

* + 1. **Phân nhánh**
* **Bước 1: Dùng lệnh git checkout+<tên nhánh chuyển>**

A picture containing graphical user interface

Description automatically generated

* **Bước 2: Kiểm tra nhánh người dùng đang truy cập**

Dùng lệnh **git branch**

A picture containing text

Description automatically generated

Dấu sao trước tên nhánh cho biết vị trí nhánh hiện tại

* + 1. **Gộp nhánh**
* **Bước 1: Chuyển nhánh**

Chuyển sang bước b1

A picture containing graphical user interface

Description automatically generated

* **Bước 2: Tạo file text có tên “nhanh.txt”**

A picture containing graphical user interface

Description automatically generated

Tiến hành nhập thông tin file text sau khi nhấn enter tạo file text. Để lưu lại và thoát nhấn tổ hợp phím ctrl+D. Kết quả được hiển thị như hình dưới:

Graphical user interface, text, application

Description automatically generated

* **Bước 3: Thêm file “nhanh.txt” vào nhánh b1**



* **Bước 4: Ghi chú**

Text

Description automatically generated

* **Bước 5: Kiểm tra**

Lệnh ls dùng kiểm tra file hiện có nhánh b1

Text

Description automatically generated

Kiểm tra nhánh Master

Graphical user interface, text

Description automatically generated with medium confidence

* **Bước 6: Tiến hành gộp nhánh b1 vào nhánh Master**

Đầu tiên chuyển sang nhánh Master

Text

Description automatically generated

Tiến hành gộp nhánh b1 bằng lệnh git merge đối với b1

Text

Description automatically generated

Tiếp theo kiểm tra các file nhánh Master

Graphical user interface, text

Description automatically generated

Đã xuất hiện tên file “nhanh.txt” là file vừa tạo bên nhánh b1.

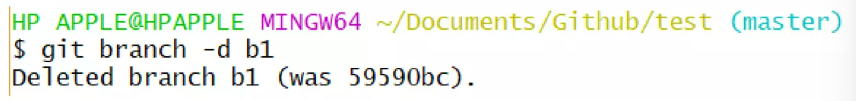
* + 1. **Xóa nhánh**
* **Bước 1: Chuyển sang nhánh Master**

Text

Description automatically generated

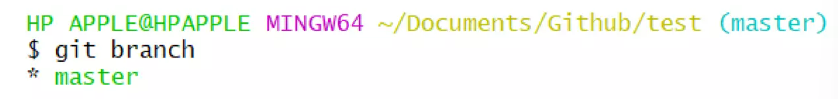
* **Bước 2: Xóa nhánh b1**

Dùng lệnh git branch -d+<tên nhánh xóa>



* **Bước 3: Kiểm tra**

Dùng lệnh git branch để xem nhánh hiện có



Chỉ còn nhánh gốc là Master.

* 1. **Quản lý các nhánh**
* Để xem commit mới nhất trên từng nhánh, bạn có thể chạy lệnh git branch -v :

Text

Description automatically generated

* Để biết nhánh nào đã được tích hợp vào nhánh hiện tại, bạn có thể sử dụng git branch –merged:

Shape, rectangle

Description automatically generated

* Để xem cách nhánh chứa các công việc/thay đổi chưa được tích hợp vào, bạn có thể chạy lệnh git branch –no-merged:



* Lệnh này lại hiện thị các nhánh khác. Bởi vì chúng bao gồm các công việc mà bạn chưa tích hợp vào, xóa nó đi bằng lệnh git branch -d sẽ báo lỗi:

Text

Description automatically generated

1. **MERGE**
   1. **Lệnh “GIT MERGE”**

Trong Git, hợp nhất(merge) là một thủ tục để kết nối lịch sử đã phân nhánh. Nó kết hợp hai hoặc nhiều lịch sử với nhau. Lệnh git merge tạo điều kiện cho bạn lấy dữ liệu được tạo bởi git branch và tích hợp chúng vào một nhánh duy nhất. Git merge sẽ liên kết một loạt các commit thành một lịch sử thống nhất.

Lệnh **git merge** được sử dụng để gộp các nhánh.

Cú pháp cho lệnh hợp nhất git như sau: **Git merge <query>**

* + - **Để hợp nhất commit được chỉ định với nhánh hiện đang hoạt động:**

**Git merge < commit >**

Lệnh trên sẽ hợp nhất commit được chỉ định với nhánh hiện đang hoạt động. Bạn cũng có thể hợp nhất commit đã chỉ định với một nhánh cụ thể bằng cách chuyển tên nhánh vào <commit>.

* + - **Để hợp nhất các commit vào nhánh chính:**

Sử dụng lệnh log để tìm id commit cụ thể

**Git log**

Để hợp nhất các commit vào nhánh chính, hãy chuyển sang nhánh chính

**Git merge master**

* Git merge nhánh:

Git cho phép gộp toàn bộ nhánh này vào nhánh khác. Giả sử bạn đã thực hiện nhiều thay đổi trên một nhánh và muốn hợp nhất tất cả các thay đổi đó cùng một lúc. Git cho phép bạn làm như vậy.

* 1. **Xung đột khi git merge**

Khi hai nhánh đang cố gắng hợp nhất và cả hai đều được chỉnh sửa cùng một lúc và trong cùng một file, Git sẽ không thể xác định phiên bản nào sẽ thực hiện để thay đổi. Tình huống như vậy được gọi là xung đột hợp nhất. Nếu tình huống như vậy xảy ra, nó sẽ dừng ngay trước khi hợp nhất commit để bạn có thể giải quyết xung đột theo cách thủ công.

Diagram

Description automatically generated

Hãy cùng hiểu nó bằng **ví dụ** sau.

Giả sử kho lưu trữ từ xa của tôi đã được sao chép bởi hai thành viên trong nhóm của tôi là **user1** và **user2** . Người dùng1 đã thực hiện các thay đổi như bên dưới trong file chỉ mục dự án của tôi.

Text

Description automatically generated

Cập nhật nó trong kho lưu trữ cục bộ với sự trợ giúp của lệnh git add.

Text

Description automatically generated

Bây giờ hãy commit các thay đổi và cập nhật nó với kho lưu trữ từ xa. Xem kết quả bên dưới:

Text

Description automatically generated

Bây giờ, kho lưu trữ từ xa của tôi sẽ trông như thế này:

Graphical user interface, text, application

Description automatically generated

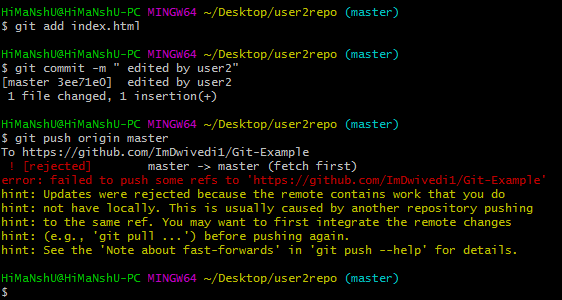
Nó sẽ hiển thị trạng thái của file như được chỉnh sửa bởi ai và khi nào.

Bây giờ, cùng lúc, **user2** cũng cập nhật file chỉ mục như sau.

Graphical user interface, text

Description automatically generated

Người user2 đã thêm và commit các thay đổi trong kho lưu trữ cục bộ. Nhưng khi anh ta cố gắng đẩy nó đến máy chủ từ xa, nó sẽ phát ra lỗi. Xem kết quả bên dưới:



Trong kết quả đầu ra ở trên, máy chủ biết rằng file đã được cập nhật và không được hợp nhất với các nhánh khác. Vì vậy, yêu cầu push đã bị máy chủ từ xa từ chối. Nó sẽ đưa ra một thông báo lỗi như **[từ chối] không thể đẩy một số giới thiệu đến <URL từ xa>** . Nó sẽ đề xuất bạn pull kho lưu trữ trước trước khi push. Xem lệnh dưới đây:

Text

Description automatically generated

Trong đầu ra đã cho, lệnh git rebase được sử dụng để kéo kho lưu trữ từ URL từ xa. Tại đây, nó sẽ hiển thị thông báo lỗi như **xung đột hợp nhất trong <tên file>** .

* 1. **Giải quyết xung đột**

Để giải quyết xung đột, cần biết xung đột có xảy ra hay không và tại sao lại xảy ra. Lệnh công cụ hợp nhất Git được sử dụng để giải quyết xung đột. Lệnh hợp nhất được sử dụng như sau: **Git mergetool**

Trong kho lưu trữ của tôi, nó sẽ dẫn đến:

Graphical user interface

Description automatically generated

Kết quả trên hiển thị trạng thái của file bị xung đột. Để giải quyết xung đột, hãy vào chế độ insert chỉ bằng cách nhấn phím **I** và thực hiện các thay đổi như bạn muốn. Nhấn **phím Esc** để thoát ra từ chế độ chèn. Nhập: **w!**ở cuối trình chỉnh sửa để lưu và thoát các thay đổi. Để chấp nhận các thay đổi, hãy sử dụng lệnh rebase. Nó sẽ được sử dụng như sau:

**git rebase –continue**

Do đó, xung đột đã được giải quyết. Xem kết quả bên dưới:

Text

Description automatically generated

Trong kết quả đầu ra ở trên, xung đột đã được giải quyết và kho lưu trữ cục bộ được đồng bộ hóa với kho lưu trữ từ xa.

1. **ĐÁNH DẤU – TAG**
   1. **Liệt kê tag**

Chỉ cần gõ **git tag**

Shape, rectangle

Description automatically generated with medium confidence

Lệnh này sẽ liệt kê các tag được sắp xếp theo thứ tự bảng chứ cái

Có thể tìm kiếm một tag sử dụng mẫu (pattern). Ví dụ, trong kho chứa mã nguồn của Git có chứa hơn 240 tag. Nếu như bạn chỉ quan tâm đến các tag thuộc dải 1.4.2, bạn có thể chạy lệnh sau:

* 1. **Thêm tag mới**

Git sử dụng hai loại tag chính:

* + - * Lightweight (hạng nhẹ): một nhánh mà không có sự thay đổi - nó chỉ trỏ đến một commit nào đó.
      * Annotated (chú thích): được lưu trữ như là những đối tượng đầy đủ trong cơ sở dữ liệu của Git.
    1. **Amotated tags**

Tạo một tag chú thích (annnotated) trong Git rất đơn giản. Cách dễ nhất là sử dụng -a khi bạn chạy lệnh tag:

Text

Description automatically generated

Tham số -m được sử dụng để truyền vào nội dung/thông điệp cho tag.

Bạn có thể xem được thông tin của tag cùng với commit được tag bằng cách sử dụng lệnh git show:

Text

Description automatically generated

* + 1. **Signed Tags**

Bạn cũng có thể ký các tag của bạn sử dụng GPG, giải sử bạn có một private key. Tất cả những gì bạn cần phải làm là sử dụng -s thay vì -a:

Text

Description automatically generated

Nếu bạn chạy lệnh git show trên tag đó, bạn có thể thấy được chữ ký GPG của bạn được đính kèm theo nó:

Text

Description automatically generated

* + 1. **Lightweight Tags**

Một cách khác để tag các commit là sử dụng lightweight tag. Cơ bản nó là mã băm của một commit được lưu lại vào trong một tập tin - ngoài ra không còn thông tin nào khác. Để tạo một lightweight tag, bạn không sử dụng -a, -s hay -m:

Shape, rectangle

Description automatically generated

Show commit:

Text

Description automatically generated