1. **Overview**
2. Tại sao phải sử dụng GIT

* **Theo dõi các thay đổi đối với mã nguồn**: Git cho phép theo dõi các thay đổi đối với source code của mình, điều này có thể hữu ích để khôi phục các phiên bản cũ của mã, tìm lỗi và quản lý các thay đổi đối với mã.
* **Làm việc theo nhóm**: Git cho phép các nhà phát triển làm việc cùng nhau trên một dự án mà không làm xáo trộn công việc của nhau. Mỗi nhà phát triển có thể có nhánh riêng của mã và sau đó hợp nhất các thay đổi của họ vào nhánh chính khi họ sẵn sàng.
* **Chia sẻ mã**: Git giúp chia sẻ source code với người khác một cách dễ dàng. Bạn có thể tạo một kho lưu trữ Git công khai và chia sẻ liên kết đến kho lưu trữ với bất kỳ ai.
* **Lưu trữ mã**: Git cho phép lưu trữ mã của mình một cách an toàn. Source code được lưu trữ trên máy tính của bạn và cũng được lưu trữ trên một máy chủ từ xa. Điều này có nghĩa là bạn sẽ không bao giờ mất mã của mình, ngay cả khi máy tính của bạn bị hỏng.

1. Repository là gì?

Repository trong GitHub là một kho lưu trữ mã nguồn. Nó là một không gian lưu trữ có thể lưu trữ mã nguồn, tài liệu và các tài nguyên khác của mình. Repository có thể được chia sẻ với người khác, do đó bạn có thể cộng tác với những người khác trên dự án của mình.

Repository được tạo bằng Git, một hệ thống quản lý phiên bản phân tán. Điều này có nghĩa là bạn có thể làm việc trên mã của mình từ bất kỳ máy tính nào và các thay đổi của bạn sẽ được đồng bộ hóa với các máy tính khác.

1. Branch là gì?

Branch trong GitHub là một nhánh của mã nguồn trong một kho lưu trữ. Nó cho phép bạn làm việc trên các tính năng hoặc thay đổi khác nhau mà không ảnh hưởng đến mã chính. Khi bạn hoàn thành các thay đổi của mình, bạn có thể hợp nhất chúng vào mã chính.

Tạo mới 1 nhánh: git branch <branch name>.

Chuyển nhánh: git checkout <branch name>, git switch <branch name>.

Hợp nhất 1 nhánh vào nhánh chính: git merge < branch name to merge >.

1. Commit là gì?

Commit trong Git là một bản chụp nhanh của source code tại một thời điểm cụ thể. Khi bạn thực hiện một commit, bạn sẽ lưu trữ các thay đổi đối với mã nguồn của mình trong một bản ghi. Điều này cho phép bạn theo dõi các thay đổi đối với mã nguồn của mình, khôi phục các phiên bản cũ của mã và cộng tác với những người khác.

1. **Cách sử dụng**
2. **Git config**: là một lệnh được sử dụng để cấu hình Git. Nó cho phép bạn đặt các giá trị cho các biến cấu hình khác nhau, chẳng hạn như tên người dùng, email, màu sắc và các tùy chọn khác. Có 2 loại cấu hình git:

* Cấu hình toàn cục: Đây là cấu hình được áp dụng cho tất cả các kho lưu trữ Git trên máy tính của bạn. Bạn có thể đặt cấu hình toàn cục bằng cách sử dụng lệnh git config --global.
* Cấu hình cục bộ: Đây là cấu hình chỉ được áp dụng cho một kho lưu trữ Git cụ thể. Bạn có thể đặt cấu hình cục bộ bằng cách sử dụng lệnh git config --local.

Một số cấu hình Git phổ biến bao gồm:

* user.name: Tên người dùng của bạn.
* user.email: Email của bạn.
* color.ui: Cài đặt màu cho git.
* core.editor: Trình soạn thảo văn bản mặc định của bạn.
* diff.tool: Trình so sánh tệp mặc định của bạn.

Để xem tất cả các biến cấu hình Git, bạn có thể sử dụng lệnh: git config –list.

1. **Git clone**: là một lệnh được sử dụng để tạo một bản sao cục bộ của một kho lưu trữ Git từ xa. Điều này cho phép bạn làm việc trên mã nguồn của kho lưu trữ từ máy tính của bạn.

Clone 1 kho lưu trữ git sử dụng cú pháp: git clone <url>



Sau khi thực hiện lệnh clone, một thư mục mới sẽ được tạo trên máy tính của bạn với tên giống với tên của kho lưu trữ. Thư mục này sẽ chứa tất cả các tệp và thư mục trong kho lưu trữ.

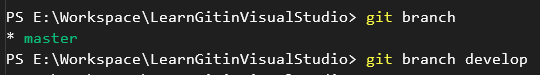
Bạn có thể bắt đầu làm việc trên mã nguồn của kho lưu trữ bằng cách chuyển đổi sang thư mục mới. Sau đó, bạn có thể sử dụng các lệnh Git khác để theo dõi các thay đổi đối với mã nguồn, cộng tác với những người khác và chia sẻ mã nguồn của bạn.

Một số tùy chọn có thể được sử dụng với lệnh git clone:

* --branch: Chọn một nhánh cụ thể để clone.
* --depth: Clone một số lịch sử của kho lưu trữ.
* --single-branch: Clone chỉ một nhánh duy nhất.
* --verbose: Hiển thị thêm thông tin khi clone
* --help: Hiển thị trợ giúp về lệnh

1. **Create branch**: cú pháp để tạo mới 1 branch trong git:

Git branch <branch name>



1. **Git checkout**: là một lệnh trong Git được sử dụng để chuyển đổi giữa các nhánh (branch) hoặc để lấy một tệp hoặc thư mục ra khỏi kho lưu trữ.

* git checkout <branch name>: chuyển đổi giữa các nhánh.



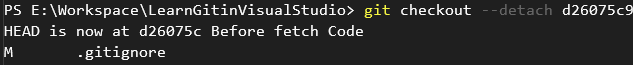
* git checkout <file or folder name>: lấy 1 tệp hoặc thư mục ra khỏi kho lưu trữ

Một số lệnh phổ biến với git checkout:

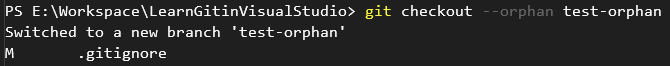
* -b: Tạo một nhánh mới và chuyển sang nó.



* --detach: Được sử dụng để tách HEAD khỏi nhánh hiện tại và chuyển sang một commit cụ thể. Điều này có nghĩa là bạn sẽ không còn ở trong bất kỳ nhánh nào và bạn sẽ có thể làm việc trực tiếp trên commit mà bạn đã chọn.



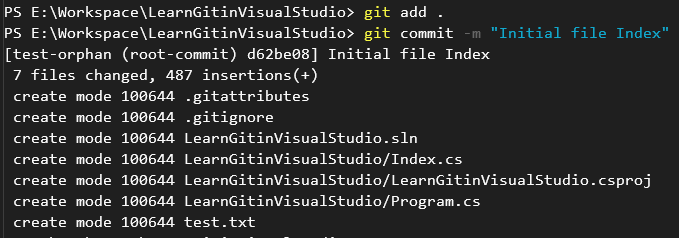
* --orphan: Được sử dụng để tạo một nhánh mới bị mồ côi từ trạng thái hiện tại của kho lưu trữ. Một nhánh bị mồ côi là một nhánh không có gốc chung với bất kỳ nhánh nào khác. Điều này có nghĩa là các thay đổi được thực hiện trên nhánh bị mồ côi sẽ không được phản ánh trên bất kỳ nhánh nào khác.



* --force: Được sử dụng để ghi đè bất kỳ thay đổi nào đã được thực hiện trong cây làm việc. Điều này có nghĩa là lệnh sẽ xóa bất kỳ thay đổi nào bạn đã thực hiện và thay thế chúng bằng các thay đổi từ kho lưu trữ.

1. **Git commit**: là một lệnh trong Git được sử dụng để lưu trữ các thay đổi đối với mã nguồn của bạn. Khi bạn thực hiện một commit, bạn sẽ lưu trữ các thay đổi đối với mã nguồn của mình trong một bản ghi. Điều này cho phép bạn theo dõi các thay đổi đối với mã nguồn của mình, khôi phục các phiên bản cũ của mã và cộng tác với những người khác

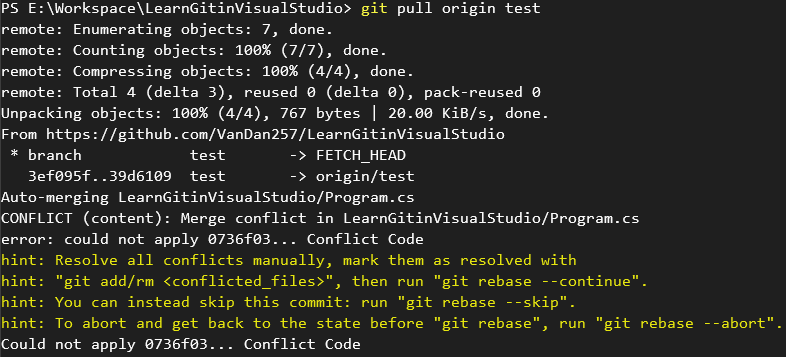
Cú pháp để thực hiện 1 commit: git commit –m <thông điệp>



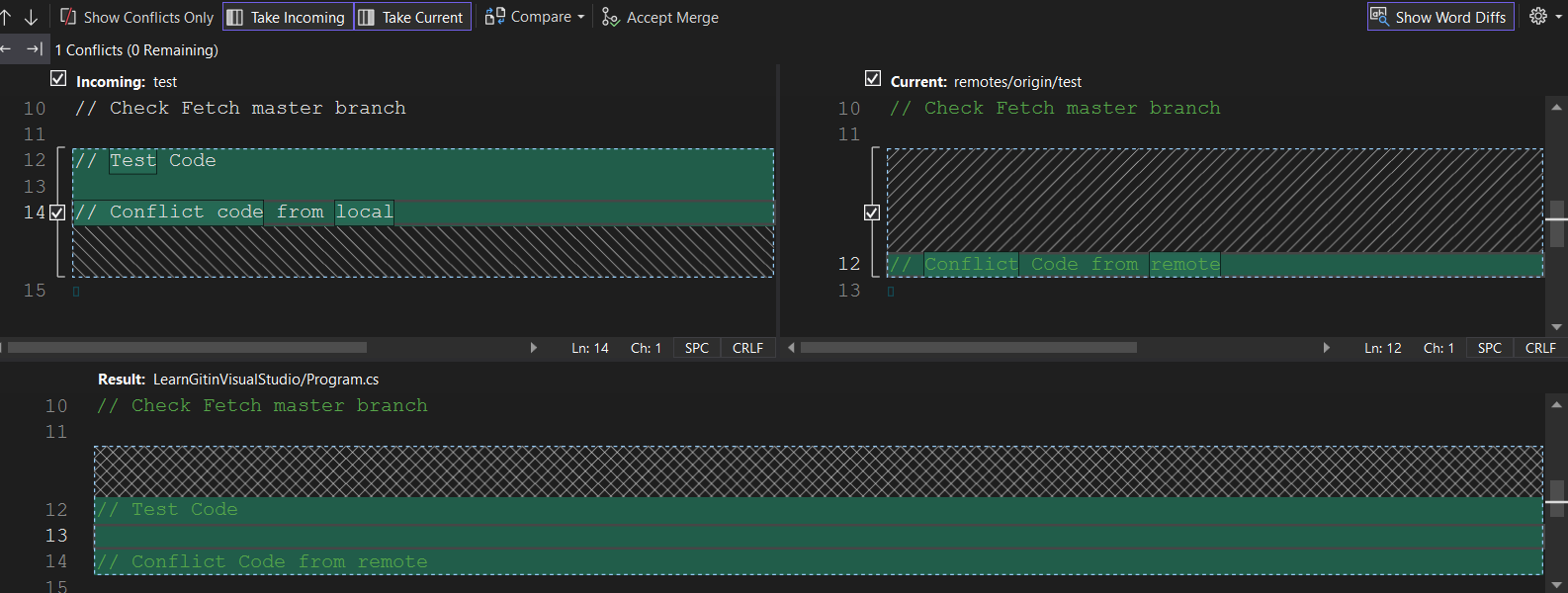
Khi thực hiện 1 commit, đầu tiên phải sử dụng lệnh git add để thêm các tệp hoặc thư mục sẽ được commit. Sau đó sử dụng lệnh git commit để thực hiện commit

1. **Git pull**: được sử dụng để lấy các thay đổi từ một nhánh khác và hợp nhất chúng vào nhánh hiện tại. Điều này cho phép bạn cộng tác với những người khác trên dự án của mình.

Cú pháp: git pull <remote> <branch name>.



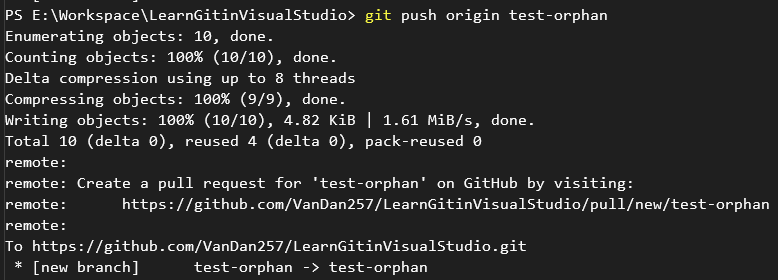
Xử lý xung đột:



Lệnh git pull sẽ lấy các thay đổi từ máy chủ từ xa và hợp nhất chúng vào nhánh hiện tại. Nếu có bất kỳ xung đột nào, lệnh git pull sẽ dừng lại và yêu cầu bạn giải quyết chúng trước khi tiếp tục.

1. **Git push**: được sử dụng để đẩy các thay đổi từ nhánh hiện tại lên một nhánh từ xa. Điều này cho phép bạn chia sẻ các thay đổi của mình với những người khác trên dự án của mình.

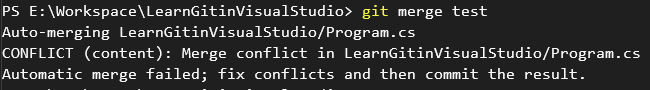
Cú pháp: git push <remote> <branch name>.



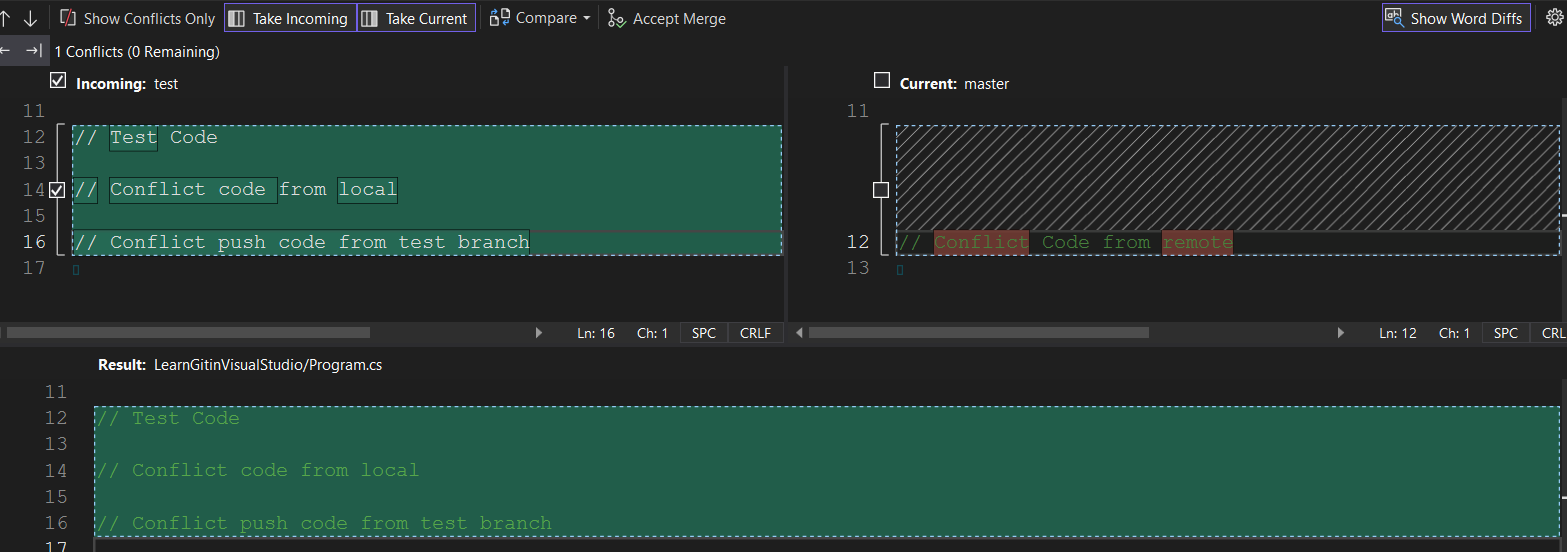
Lệnh git push sẽ đẩy các thay đổi từ máy tính của bạn lên máy chủ từ xa. Nếu có bất kỳ xung đột nào, lệnh git push sẽ dừng lại và yêu cầu bạn giải quyết chúng trước khi tiếp tục.

1. **Git merge**: là một lệnh trong Git được sử dụng để hợp nhất các thay đổi từ một nhánh khác vào nhánh hiện tại. Điều này cho phép bạn cộng tác với những người khác trên dự án của mình.

Cú pháp: git merge <branch name>.



Xử lý xung đột:



Lệnh git merge là một công cụ mạnh mẽ có thể được sử dụng để cộng tác với những người khác trên dự án của bạn. Tuy nhiên, điều quan trọng cần lưu ý là lệnh này có thể gây ra xung đột nếu các thay đổi từ hai nhánh chồng chéo lên nhau. Nếu điều này xảy ra, bạn sẽ cần giải quyết các xung đột trước khi tiếp tục.

Dưới đây là một số mẹo để giải quyết xung đột:

* So sánh các thay đổi từ hai nhánh.
* Xác định các thay đổi chồng chéo.
* Quyết định xem sẽ giữ các thay đổi từ nhánh nào.
* Sử dụng lệnh git add để thêm các thay đổi đã giải quyết vào hàng đợi để commit.
* Thực hiện một commit.

1. **Git revert**: là một lệnh trong Git được sử dụng để đảo ngược các thay đổi từ một commit. Điều này cho phép bạn khôi phục mã của mình về trạng thái trước khi thực hiện commit.

Cú pháp: git revert <commit-id>.

Code trước khi revert:

Lệnh git revert sẽ tạo một commit mới đảo ngược các thay đổi từ commit đã chỉ định. Commit mới sẽ có trạng thái là "reverted" và nó sẽ ghi chú các thay đổi được đảo ngược.

Lệnh git revert là một công cụ quan trọng trong Git. Nó cho phép bạn khôi phục mã của mình về trạng thái trước khi thực hiện commit.

* Đảo ngược các thay đổi từ commit cuối cùng: git revert HEAD.
* Đảo ngược các thay đổi từ một phạm vi commit: git revert <commit-id1> <commit-id2>.
* Đảo ngược các thay đổi từ 1 nhánh: git revert <branch name>.

Lệnh git revert là một công cụ mạnh mẽ có thể được sử dụng để khôi phục mã của mình về trạng thái trước khi thực hiện commit. Tuy nhiên, điều quan trọng cần lưu ý là lệnh này có thể gây ra xung đột nếu các thay đổi từ commit đã chỉ định chồng chéo lên nhau với các thay đổi trong mã hiện tại của bạn. Nếu điều này xảy ra, bạn sẽ cần giải quyết các xung đột trước khi tiếp tục.

1. **Git conflict**: là tình trạng xảy ra khi hai nhánh có những thay đổi trùng lặp nhau. Điều này có thể xảy ra khi bạn cố gắng hợp nhất (merge) hai nhánh hoặc khi bạn cố gắng thực hiện một thay đổi trong một tệp mà đã có một thay đổi khác trong tệp đó.

Để giải quyết conflict, bạn cần sửa đổi tệp theo cách sao cho các thay đổi từ hai nhánh không trùng lặp nhau. Sau khi đã sửa đổi tệp, bạn cần lưu tệp và sau đó thực hiện lại lệnh merge hoặc lệnh thực hiện thay đổi.

Một số mẹo để giải quyết conflict trong Git:

* Sử dụng lệnh git status để xem các tệp có conflict.
* Sửa đổi các tệp theo cách sao cho các thay đổi từ hai nhánh không trùng lặp nhau.
* Lưu tệp và sau đó thực hiện lại lệnh merge hoặc lệnh thực hiện thay đổi.
* Nếu bạn không thể giải quyết conflict, bạn có thể bỏ qua conflict bằng cách sử dụng lệnh git merge --ff-only hoặc git merge --ignore-conflicts.

1. **Gitignore:** là một tệp văn bản chứa danh sách các tệp hoặc thư mục mà bạn không muốn Git theo dõi. Gitignore thường được sử dụng để loại trừ các tệp tạm thời, tệp bản phát hành và tệp cấu hình khỏi kho lưu trữ.

Khi bạn đã thêm các tệp hoặc thư mục vào tệp gitignore, Git sẽ không còn theo dõi các tệp hoặc thư mục đó. Điều này có nghĩa là các tệp hoặc thư mục đó sẽ không được thêm vào kho lưu trữ khi bạn thực hiện một commit và chúng cũng sẽ không được tải xuống khi bạn kéo từ kho lưu trữ từ xa.