**Tìm hiểu về GIT**



MỤC LỤC

[**I.** **Giới thiệu tổng quan về GIT** 3](#_Toc179750378)

[**1.Sơ lược về lịch sử** 3](#_Toc179750379)

[**2.Cơ bản về GIT** 3](#_Toc179750380)

[**II. Cài đặt GIT trên window** 6](#_Toc179750381)

[**III. Các khái niệm cơ bản về Git** 7](#_Toc179750382)

[**1. Repository (Kho lưu trữ)** 7](#_Toc179750383)

[**2. Commit** 8](#_Toc179750384)

[**3. Branch (Nhánh)** 8](#_Toc179750385)

[**4. Merge (Gộp nhánh)** 9](#_Toc179750386)

[**5. Clone** 9](#_Toc179750387)

[**6. Pull và Push** 9](#_Toc179750388)

[**7. Conflict (Xung đột)** 10](#_Toc179750389)

[**IV. Một số lệnh Git cơ bản** 10](#_Toc179750390)

[**1. git init** 10](#_Toc179750391)

[**2. git clone <URL>** 10](#_Toc179750392)

[**3. git status** 11](#_Toc179750393)

[**4. git add <file>** 11](#_Toc179750394)

[**5. git commit -m** 12](#_Toc179750395)

[**6. git branch** 12](#_Toc179750396)

[**7. git checkout <branch>** 12](#_Toc179750397)

[**8. git merge <branch>** 12](#_Toc179750398)

[**9. git pull** 13](#_Toc179750399)

[**10. git push** 13](#_Toc179750400)

[**V. Các dịch vụ hỗ trợ GIT phổ biến** 13](#_Toc179750401)

[**1.GitHub** 13](#_Toc179750402)

[**2.GitLab** 14](#_Toc179750403)

[**3.Bitbucket** 14](#_Toc179750404)

1. **Giới thiệu tổng quan về GIT**
2. **Sơ lược về lịch sử**

Năm 2002, dự án nhân Linux bắt đầu sử dụng một hệ thống quản lý phiên bản phân tán (DVCS) độc quyền có tên là BitKeeper.

Đến năm 2005, mối quan hệ giữa cộng đồng phát triển nhân Linux và công ty thương mại phát triển BitKeeper đã đổ vỡ.

Điều này đã thúc đẩy cộng đồng phát triển Linux tạo ra công cụ riêng của họ dựa trên những bài học rút ra khi sử dụng BitKeeper.

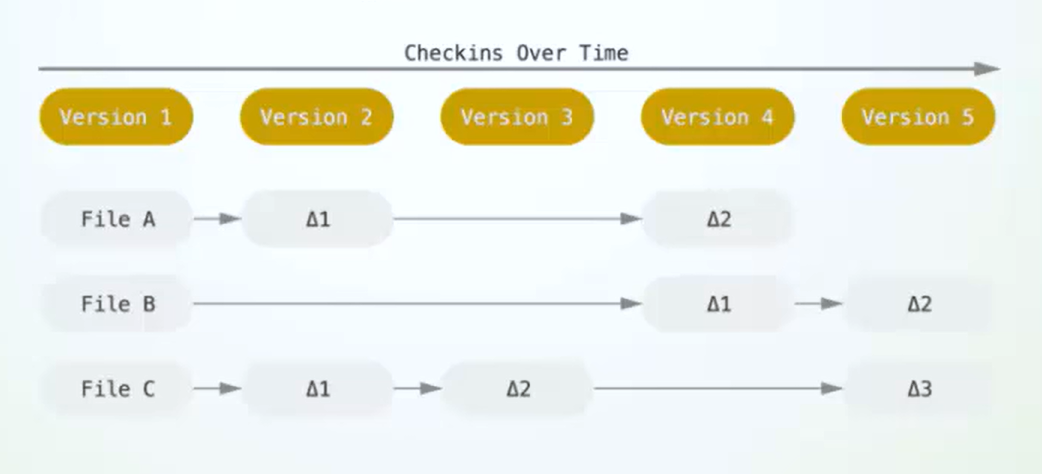
Một số mục tiêu của hệ thống mới bao gồm:

* Tốc độ
* Thiết kế đơn giản
* Hỗ trợ mạnh mẽ cho phát triển phi tuyến tính (hàng nghìn nhánh song song)
* Hoàn toàn phân tán
* Có thể xử lý hiệu quả các dự án lớn như nhân Linux (về tốc độ và kích thước dữ liệu)

1. **Cơ bản về GIT**
   1. **Snapshots, Not Differences**

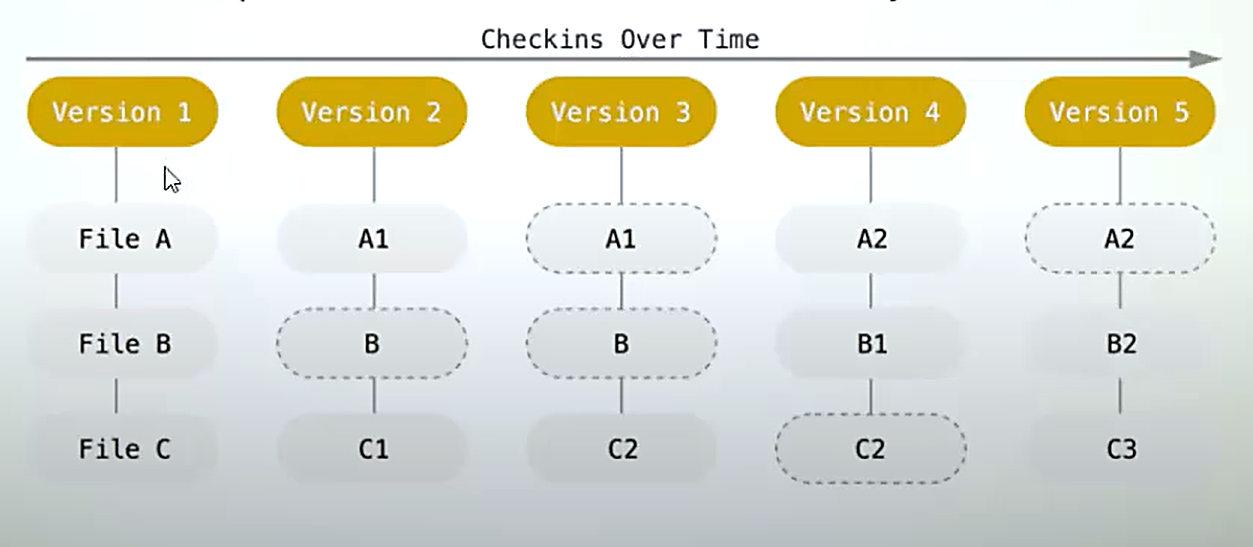
Sự khác biệt lớn nhất giữa Git và các hệ thống quản lý phiên bản khác (như Subversion và những hệ thống tương tự) nằm ở cách Git xử lý dữ liệu.

Trong khi hầu hết các hệ thống khác lưu trữ thông tin dưới dạng danh sách các thay đổi trên từng tệp, thì những hệ thống này (CVS, Subversion, Perforce, Bazaar, v.v.) nhìn nhận dữ liệu như một tập hợp các tệp cùng với lịch sử thay đổi của từng tệp theo thời gian.



Git coi dữ liệu của mình giống như một tập hợp các “snapshot” (ảnh chụp) của một hệ thống tệp thu nhỏ.

Đây là sự khác biệt quan trọng giữa Git và hầu hết các hệ thống quản lý phiên bản khác.



* 1. **Nearly Every Operation Is Local**

Trong Git, hầu hết các thao tác chỉ yêu cầu sử dụng các tệp và tài nguyên cục bộ, mà không cần phải truy cập thông tin từ các máy tính khác trong mạng.

Với việc lưu trữ toàn bộ lịch sử của dự án ngay trên ổ đĩa cục bộ, các thao tác trong Git thường diễn ra gần như tức thì, mang lại trải nghiệm mượt mà và hiệu quả cho người sử dụng.

* 1. **Git Has Integrity**

Mọi dữ liệu trong Git đều được kiểm tra và tính toán mã băm trước khi lưu trữ, và sau đó được tham chiếu bằng mã băm đó.

Điều này có nghĩa là không thể thay đổi nội dung của bất kỳ tệp hoặc thư mục nào mà Git không phát hiện được.

Bạn sẽ không bao giờ mất thông tin trong quá trình truyền tải hay gặp phải lỗi tệp mà không có sự phát hiện từ Git.

Cơ chế mà Git sử dụng để thực hiện việc kiểm tra mã băm này được gọi là mã băm SHA-1. Đây là một chuỗi dài 40 ký tự, bao gồm các ký tự thập lục phân (0-9 và a-f), được tính toán dựa trên nội dung của tệp hoặc cấu trúc thư mục trong Git.

* 1. **Git Generally Only Adds Data**

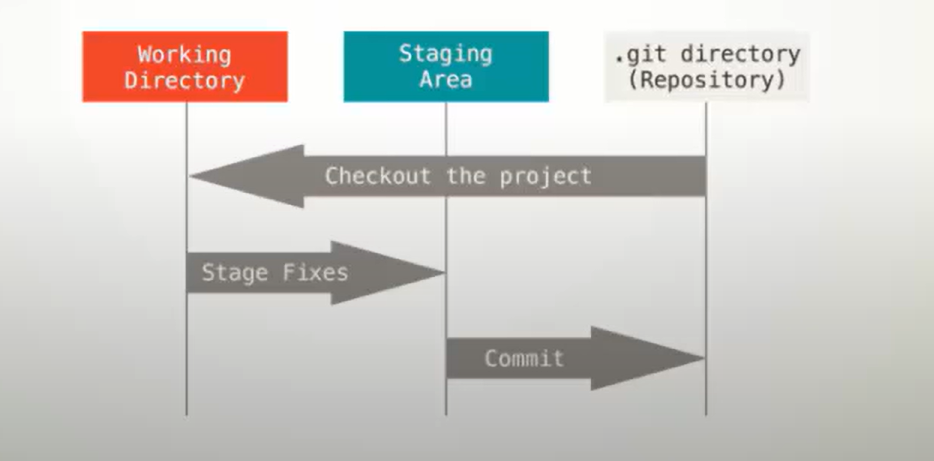
Trong Git, hầu hết các thao tác mà người dùng thực hiện đều nhằm mục đích bổ sung dữ liệu vào cơ sở dữ liệu của hệ thống. Một khi người dùng cam kết một snapshot vào Git, khả năng mất mát dữ liệu trở nên rất khó khăn, đặc biệt là khi họ thường xuyên đẩy kho lưu trữ của mình lên một repository khác.

Điều này không chỉ đảm bảo tính toàn vẹn của các phiên bản dự án mà còn tạo ra một cơ chế khôi phục dữ liệu hiệu quả khi cần thiết. Sự an toàn và bảo mật này là một trong những lý do khiến Git trở thành một công cụ quản lý phiên bản được ưa chuộng trong cộng đồng lập trình viên.

* 1. **The Three States**

Điều cần ghi nhớ chính về Git, nếu bạn muốn quá trình học tập của mình diễn ra một cách suôn sẻ, là Git có ba trạng thái chính mà các tệp có thể tồn tại:

* + cam kết (committed)
* chỉnh sửa (modified)
* chuẩn bị(staged).



Thư mục Git là nơi Git lưu trữ metadata và cơ sở dữ liệu đối tượng cho dự án của bạn. Đây là phần quan trọng nhất của Git và là những gì được sao chép khi bạn nhân bản (clone) một kho lưu trữ từ máy tính khác.

Thư mục làm việc (working directory) là một phiên bản của dự án, trong đó các tệp được trích xuất từ cơ sở dữ liệu nén trong thư mục Git và được lưu trữ trên đĩa để bạn có thể sử dụng hoặc chỉnh sửa.

Khu vực staging (staging area) là một tệp, thường nằm trong thư mục Git của bạn, lưu trữ thông tin về những thay đổi nào sẽ được đưa vào cam kết (commit) tiếp theo. Khu vực này đôi khi được gọi là "index", nhưng cũng thường được gọi là khu vực staging.

Quy trình làm việc cơ bản trong Git diễn ra như sau:

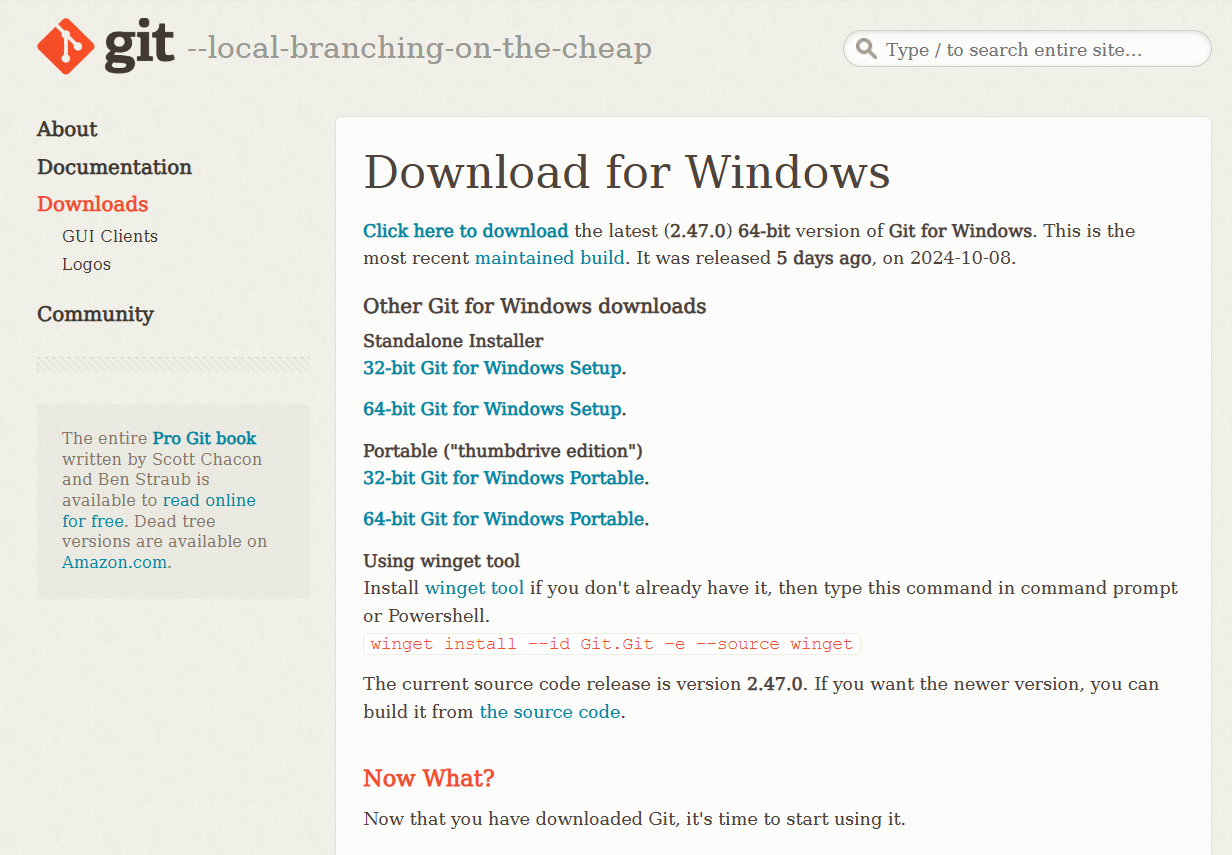
*1. Chỉnh sửa các tệp trong thư mục làm việc.*

*2. Chuẩn bị (stage) các tệp, thêm các snapshot của chúng vào khu vực staging.*

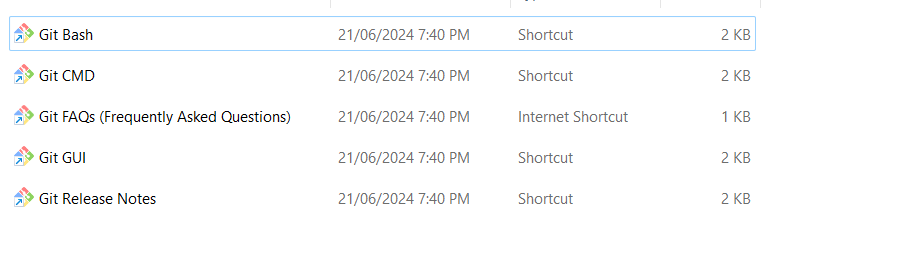
*3. Thực hiện một cam kết (commit), quá trình này lấy các tệp như chúng đang ở trong khu vực staging và lưu trữ snapshot đó vĩnh viễn vào thư mục Git của bạn.*

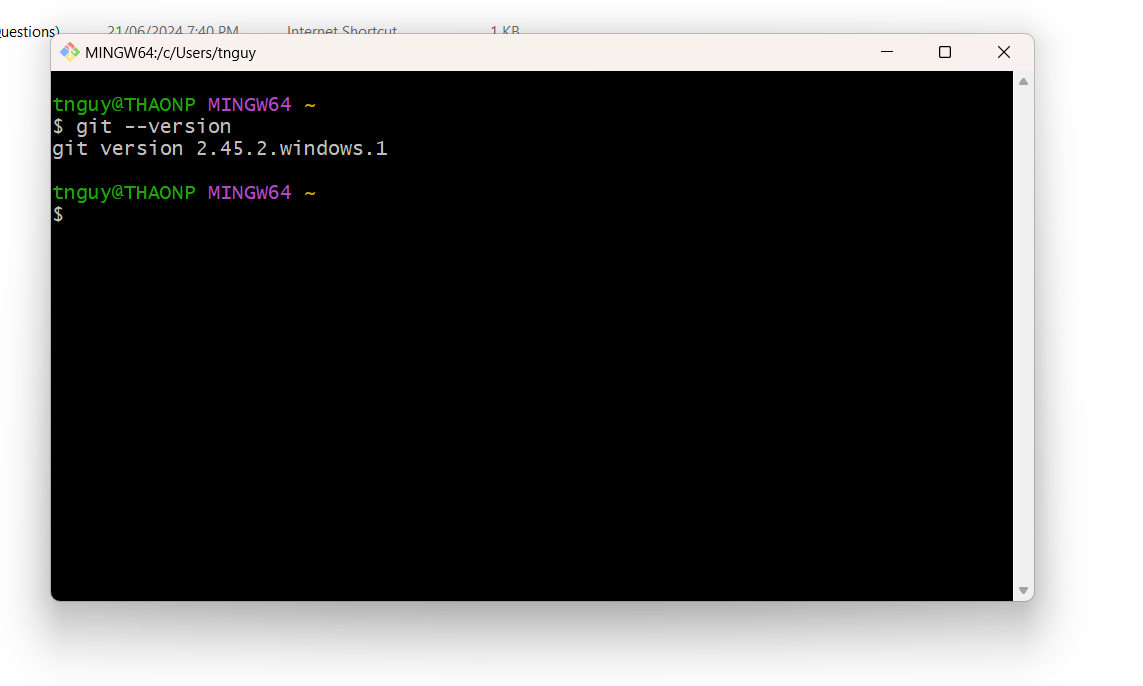
# **II. Cài đặt GIT trên window**

Link dowload: [Git - Downloading Package (git-scm.com)](https://git-scm.com/downloads/win)



Sau khi tải và cài đặt xong mở ứng dụng Git Bash để sử dụng

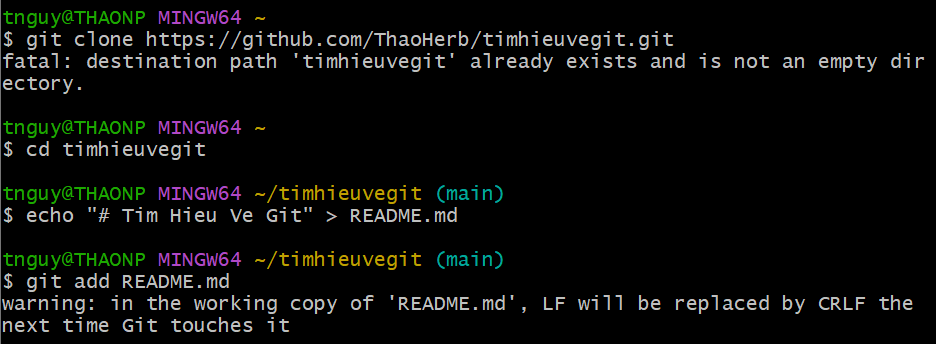




# **III. Các khái niệm cơ bản về Git**

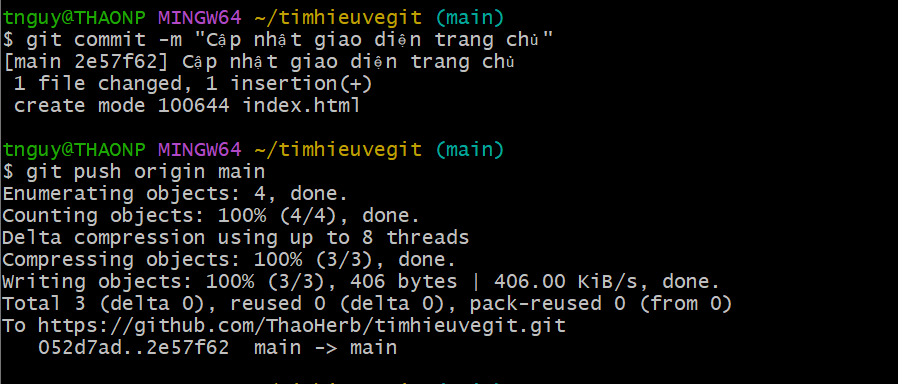
## **1. Repository (Kho lưu trữ)**

* + Khái niệm: Repository là nơi lưu trữ toàn bộ mã nguồn và lịch sử thay đổi của một dự án. Nó có thể bao gồm mã nguồn, tài liệu, và các tệp khác liên quan đến dự án.
  + Phân loại:
  + Local Repository: Đây là phiên bản của repository lưu trữ trên máy tính cá nhân của bạn. Bạn có thể làm việc offline mà không cần kết nối Internet.
  + Remote Repository: Đây là phiên bản của repository lưu trữ trực tuyến, cho phép nhiều người cùng làm việc và truy cập từ bất kỳ đâu. Một số nền tảng phổ biến cho remote repository bao gồm GitHub, GitLab, và Bitbucket.
  + Ví dụ
  + Khi bạn tạo một dự án mới bằng cách sử dụng lệnh git init, Git sẽ tạo một thư mục .git trong thư mục dự án của bạn, đó chính là local repository.
  + Có thể clone một repository từ GitHub bằng lệnh



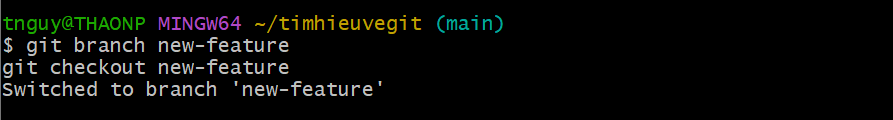
## **2. Commit**

* + Khái niệm: Commit là hành động lưu lại những thay đổi trong repository. Mỗi commit bao gồm một bức ảnh chụp (snapshot) của trạng thái mã nguồn tại thời điểm đó.
  + Đặc điểm:
  + Mỗi commit có một mã hash duy nhất, thường là một chuỗi 40 ký tự, giúp nhận diện và theo dõi các thay đổi cụ thể trong mã nguồn.
  + Thông điệp (message) kèm theo commit nên rõ ràng và mô tả ngắn gọn về các thay đổi đã thực hiện.
* Ví dụ:



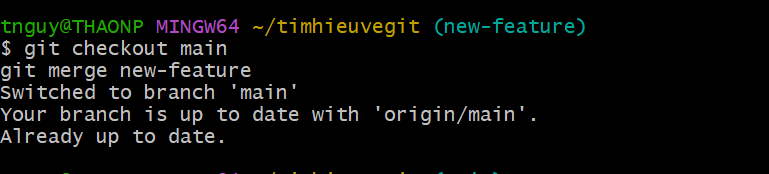
## **3. Branch (Nhánh)**

* + Khái niệm: Branch là một đường dẫn phát triển độc lập, cho phép bạn làm việc trên nhiều tính năng hoặc sửa lỗi mà không làm ảnh hưởng đến mã nguồn chính.
  + Nhánh chính: Mặc định, nhánh chính trong Git thường được gọi là main hoặc master.
  + Lợi ích: Việc sử dụng nhánh giúp quản lý quy trình phát triển một cách hiệu quả hơn, cho phép các nhà phát triển thử nghiệm mà không làm gián đoạn mã nguồn chính.
  + Ví dụ : tạo một nhánh mới là new-feature và chuyển sang làm việc trên nhánh đó.



## **4. Merge (Gộp nhánh)**

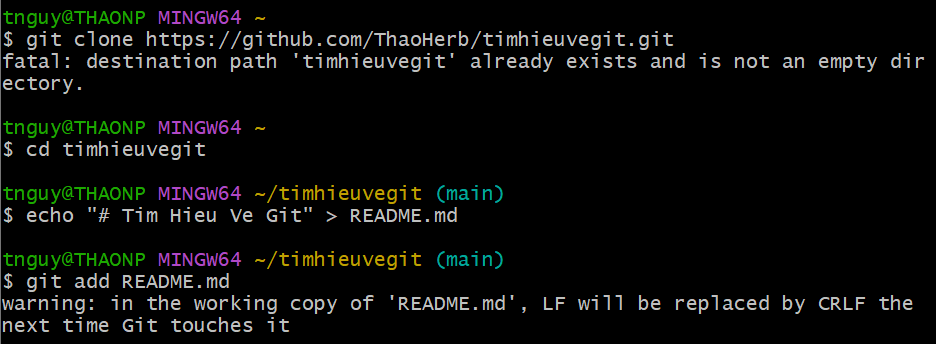
* + Khái niệm: Merge là quá trình kết hợp các thay đổi từ một nhánh khác vào nhánh hiện tại.
  + Quy trình: Sau khi hoàn thành công việc trên nhánh phụ (ví dụ: một tính năng mới), bạn có thể thực hiện merge để đưa các thay đổi vào nhánh chính bằng lệnh:



* + Xử lý xung đột: Trong quá trình merge, nếu có sự khác biệt giữa các thay đổi trên các nhánh, Git sẽ phát hiện và yêu cầu bạn giải quyết xung đột trước khi tiếp tục.

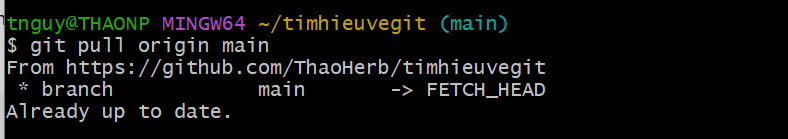
## **5. Clone**

* + Khái niệm: Clone là quá trình tạo một bản sao đầy đủ của repository từ remote về máy tính cục bộ để bạn có thể làm việc.
  + Cách sử dụng: Lệnh clone sẽ tải về toàn bộ mã nguồn, lịch sử commit và các nhánh có sẵn trong repository.
  + Ví dụ:

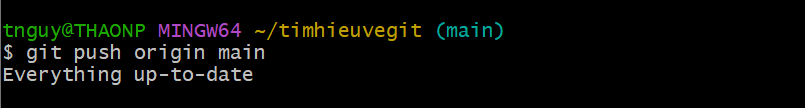


## **6. Pull và Push**

* + Pull:
* Khái niệm: Lệnh pull được sử dụng để tải về các thay đổi mới nhất từ remote repository về máy tính cục bộ. Điều này giúp đảm bảo rằng bạn luôn làm việc với phiên bản mới nhất của mã nguồn.
  + Cách sử dụng:



* + Push:
  + Khái niệm: Lệnh push cho phép bạn gửi các thay đổi đã commit từ máy tính cục bộ lên remote repository. Điều này giúp chia sẻ công việc của bạn với những người khác.
  + Cách sử dụng:



## **7. Conflict (Xung đột)**

* + Khái niệm: Xung đột xảy ra khi có các thay đổi khác nhau đối với cùng một đoạn mã từ các nhánh khác nhau. Khi bạn thực hiện merge hoặc pull, Git sẽ phát hiện các xung đột và yêu cầu bạn giải quyết trước khi tiếp tục.
  + Quy trình xử lý xung đột:

*1. Git sẽ đánh dấu những phần mã có xung đột trong tệp tin.*

*2. Bạn cần mở tệp đó, xem xét và quyết định giữ thay đổi nào.*

*3. Sau khi giải quyết xung đột, bạn cần thêm tệp đã chỉnh sửa vào staging area và thực hiện commit để hoàn tất quá trình.*

# **IV. Một số lệnh Git cơ bản**

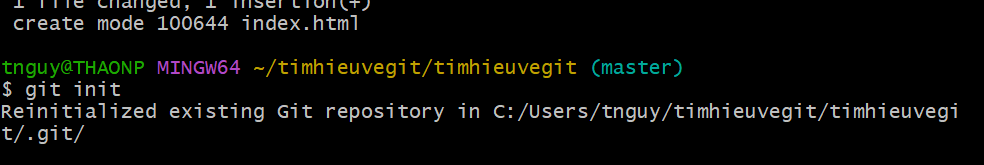
## **1. git init**

* + Mô tả: Tạo một repository Git mới trong thư mục hiện tại. Lệnh này sẽ tạo ra một thư mục con .git, nơi Git lưu trữ tất cả các thông tin và lịch sử của dự án.
  + Cách sử dụng:

mkdir my-project

cd my-project

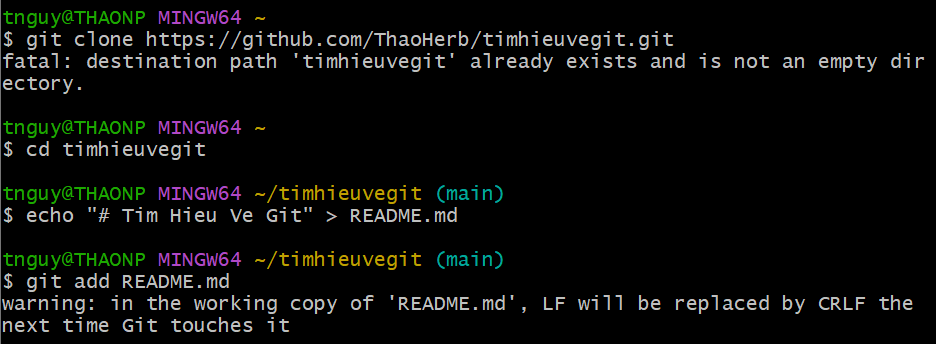
git init



## **2. git clone <URL>**

* + Mô tả: Sao chép một repository từ xa về máy cục bộ. Lệnh này sẽ tạo một bản sao đầy đủ của repository bao gồm toàn bộ lịch sử và các nhánh.
  + Cách sử dụng:

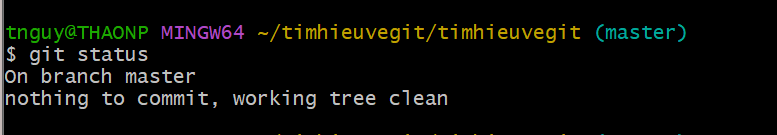
git clone https://github.com/username/repository.git



## **3. git status**

* Mô tả: Kiểm tra trạng thái hiện tại của repository. Lệnh này cho biết những thay đổi nào đã được thực hiện nhưng chưa được commit, cũng như tình trạng của các tệp.
* Cách sử dụng:

git status

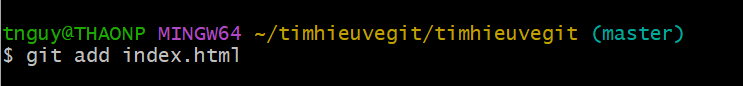


## **4. git add <file>**

* + Mô tả: Thêm tệp vào khu vực tạm thời (staging area) trước khi thực hiện commit. Bạn có thể thêm từng tệp hoặc tất cả tệp đã thay đổi.
  + Cách sử dụng:

git add index.html

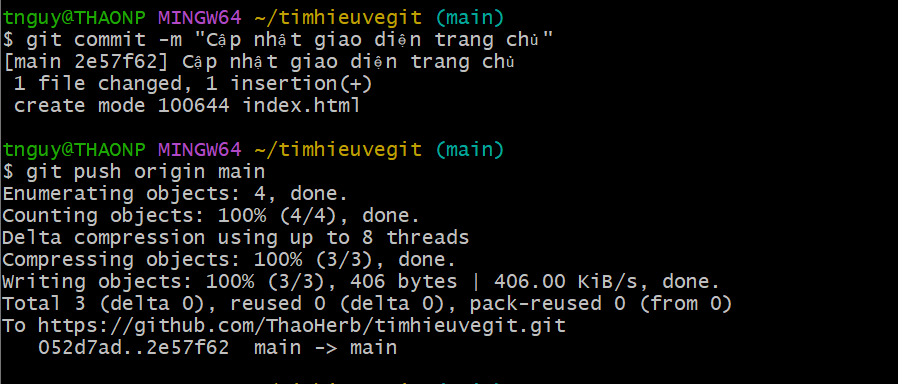
Hoặc thêm tất cả tệp: git add .



## **5. git commit -m**

* + Mô tả: Lưu các thay đổi trong repository với một thông điệp mô tả. Thông điệp này nên rõ ràng để người khác dễ hiểu về các thay đổi được thực hiện.
  + Cách sử dụng:

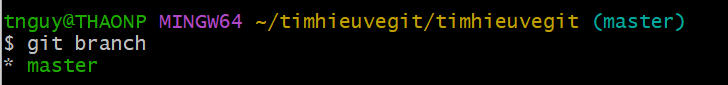
git commit -m "Sửa lỗi hiển thị trên trang chủ"



## **6. git branch**

* + Mô tả: Hiển thị danh sách các nhánh trong repository. Nhánh hiện tại sẽ được đánh dấu bằng dấu sao (\*).
  + Cách sử dụng:

git branch

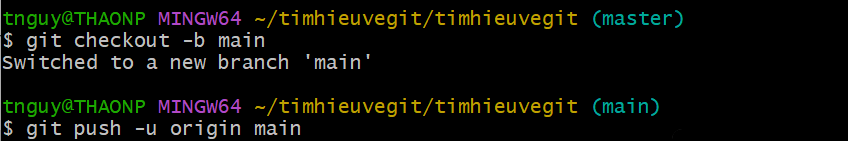


## **7. git checkout <branch>**

* + Mô tả: Chuyển sang một nhánh khác. Nếu nhánh không tồn tại, bạn có thể tạo nhánh mới bằng cách sử dụng thêm tùy chọn -b.
  + Cách sử dụng:

git checkout feature/new-feature

Hoặc tạo và chuyển sang nhánh mới: git checkout -b feature/new-feature

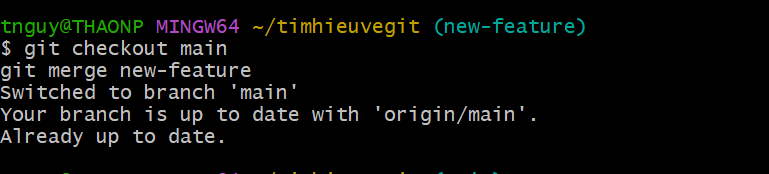


## **8. git merge <branch>**

* + Mô tả: Gộp một nhánh vào nhánh hiện tại. Lệnh này sẽ lấy tất cả các thay đổi từ nhánh được chỉ định và hợp nhất chúng vào nhánh bạn đang làm việc.
  + Cách sử dụng:

git checkout main

git merge feature/new-feature



## **9. git pull**

* + Mô tả: Tải các thay đổi từ repository trực tuyến về và tự động hợp nhất chúng vào nhánh hiện tại. Lệnh này tương đương với việc thực hiện git fetch và git merge.
  + Cách sử dụng:

git pull origin main

## **10. git push**

* + Mô tả: Đẩy các thay đổi từ repository cục bộ lên repository trực tuyến. Điều này giúp cập nhật các thay đổi mà bạn đã thực hiện cho những người khác cùng xem.
  + Cách sử dụng:

git push origin main

# **V. Các dịch vụ hỗ trợ GIT phổ biến**

1. **GitHub**



* Nền tảng phổ biến nhất để lưu trữ mã nguồn trực tuyến. GitHub cung cấp nhiều tính năng hữu ích như:
* Cộng tác: Dễ dàng làm việc cùng nhau trên cùng một dự án.
* Issue Tracking: Quản lý các vấn đề và yêu cầu.
* Pull Requests: Đề xuất các thay đổi và nhận phản hồi từ nhóm.
* CI/CD: Tích hợp các công cụ phát triển liên tục và triển khai liên tục.
* Ví dụ: Tạo một repository mới trên GitHub và thực hiện pull request để hợp nhất các thay đổi từ nhánh khác vào nhánh chính.

1. **GitLab**



* Tương tự GitHub nhưng cũng hỗ trợ tự lưu trữ (self-hosted). GitLab cung cấp nhiều tính năng mạnh mẽ cho DevOps như:
* CI/CD: Tích hợp sẵn tính năng CI/CD để tự động hóa quy trình phát triển.
* Quản lý dự án: Công cụ quản lý dự án mạnh mẽ tích hợp trực tiếp.
* Ví dụ: Sử dụng GitLab CI để tự động kiểm tra và triển khai mã mỗi khi có thay đổi.

1. **Bitbucket**



* + Dịch vụ lưu trữ mã nguồn tương tự GitHub và GitLab. Bitbucket thường được ưa chuộng trong các nhóm làm việc trong môi trường doanh nghiệp. Nó hỗ trợ cả Git và Mercurial.
  + Tính năng nổi bật:
* Quản lý nhánh mạnh mẽ: Dễ dàng quản lý và theo dõi các nhánh khác nhau.
* Tích hợp với Jira: Dễ dàng theo dõi vấn đề và yêu cầu liên quan đến phát triển phần mềm.
  + Ví dụ: Tạo các pull request trong Bitbucket để xem xét và hợp nhất các thay đổi vào nhánh chính.