# BỘ THÔNG TIN VÀ TRUYỀN THÔNG

# HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

-----


BÀI TIỂU LUẬN

## THIẾT KẾ ĐIỆN TỬ TIÊN TIẾN

ĐỀ TÀI

## QUẢN LÝ PHIÊN BẢN TRONG THIẾT KẾ (GIT)

**Giảng viên hướng dẫn:** **Thầy Lương Công Duẩn**

**Sinh viên thực hiện:** **Bùi Trung Trà – B19DCDT243**

**Lê Công Năm – B19DCDT158**

**Nguyễn Đức Quân – B19DCDT178**

**Nguyễn Thành Liêm – b19DCDT128**

**Hồ Văn Thành Minh – B19DCDT142**

**Hà Nội, 2023**

# MỤC LỤC

# DANH MỤC HÌNH ẢNH

# LỜI CẢM ƠN

Đầu tiên, xin được gửi lời cảm ơn đến Học viện Công nghệ Bưu chính Viễn thông và Khoa Kỹ thuật Điện tử 1 đã tạo ra môi trường rèn luyện, trau dồi kiến thức, kinh nghiệm tốt và hiệu quả để nhóm có cơ hội phát triển và được cung cấp các hành trang quý giá cho chuyên môn nói riêng và cuộc sống nói chung.

Xin trân trọng cảm ơn thầy Lương Công Duẩn đã tận tâm giúp đỡ nhóm thực hiện đề tài "Quản lý phiên bản trong thiết kế (Git)". Với những kiến thức và sự hướng dẫn tận tình, chi tiết của thầy đã giúp đỡ nhóm đề tài rất nhiều điều từ phong cách làm việc chuyên nghiệp đến những kiến thức chuyên môn từ cơ bản đến chuyên sâu.

Do còn hạn chế về kinh nghiệm nghiên cứu, nên đề tài của nhóm không thể tránh khỏi những sai sót, rất mong nhận được những ý kiến đóng góp của thầy để đề tài được hoàn thiện hơn.

Cuối cùng xin kính chúc thầy và gia đình luôn thành công và hạnh phúc.

Trân trọng!

Hà Nội, tháng 4 năm 2023.

Nhóm 5

# CHƯƠNG I: GIỚI THIỆU

## 1. QUẢN LÝ PHIÊN BẢN SỬ DỤNG GIT

### 1.1. Giới thiệu về quản lý phiên bản sử dụng Git

Quản lý phiên bản là một khía cạnh quan trọng trong quá trình phát triển phần mềm. Git là một trong những công cụ quản lý phiên bản phổ biến nhất hiện nay, được sử dụng rộng rãi trong cộng đồng phát triển phần mềm. Git được phát triển bởi Linus Torvalds vào năm 2005 như là một công cụ quản lý mã nguồn cho dự án Linux. Sau đó, nó đã trở thành một công cụ quản lý phiên bản độc lập và có thể được sử dụng cho các dự án phần mềm khác.

### 1.2. Tầm quan trọng của Git trong quản lý phiên bản

Git giúp cho quá trình quản lý phiên bản trở nên dễ dàng và hiệu quả hơn. Nó cho phép các nhà phát triển phần mềm theo dõi, quản lý và kiểm soát các thay đổi của mã nguồn một cách chặt chẽ, đồng thời cho phép các phiên bản khác nhau của sản phẩm tồn tại đồng thời.

Git là một trong những Hệ thống Quản lý Phiên bản Phân tán, vốn được phát triển nhằm quản lý mã nguồn (source code) hữu hiệu của Linux.

Trên Git, có thể lưu trạng thái của file khi có nhu cầu dưới dạng lịch sử cập nhật. Vì thế, có thể đưa file đã chỉnh sửa một lần về trạng thái cũ hay có thể hiển thị sự khác biệt ở nơi chỉnh sửa.

Thêm nữa, khi định ghi đè (overwrite) lên file mới nhất đã chỉnh sửa của người khác bằng file đã chỉnh sửa dựa trên file cũ, thì khi đăng (upload) lên server sẽ hiện ra cảnh cáo. Vì thế, sẽ không xảy ra thất bại về việc đã ghi đè lên nội dung chỉnh sửa của người khác mà không hề hay biết.

## 2. GIT

### 2.1. Sơ lược về Git

Git là một đại diện tiêu biểu của hệ thống quản lý phiên bản phân tán DVCS mã nguồn mở, một trong những dạng hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay.

Git có những đặc điểm sau:

- Có hướng tiếp cận mới so với các hệ thống Source Control khác như SVN hay CVS truyền thống.
- Có nhiệm vụ theo dõi những thay đổi, chỉnh sửa trong Source Code của người dùng vào mội thời điểm và đồng bộ những Source Code do họ chỉnh sửa lên Server cùng đồng nghiệp.

Ngoài ra, Git còn có thể chạy trên nhiều hệ điều hành khác nhau như Linux, Windows, MacOS,…

### 2.2. Lịch sử phát triển Git

Git được tạo ra bởi Linus Torvalds vào năm 2005. Lúc đó, dự án Linux đang sử dụng một hệ thống quản lý phiên bản tên là BitKeeper để quản lý mã nguồn của mình. Tuy nhiên, khi quyền sử dụng BitKeeper của dự án Linux bị thu hồi, Linus Torvalds quyết định tạo ra một hệ thống quản lý phiên bản của riêng mình.

Ban đầu, Git được sử dụng cho việc phát triển và quản lý mã nguồn của dự án Linux. Tuy nhiên, sau đó Git đã được sử dụng rộng rãi trong cộng đồng phát triển phần mềm và trở thành một trong những hệ thống quản lý phiên bản phổ biến nhất trên thế giới.

Hiện nay, Git được sử dụng trong các dự án phát triển phần mềm lớn như Android, jQuery, Ruby on Rails, và nhiều dự án phần mềm mã nguồn mở khác trên toàn cầu. Git cũng được tích hợp vào nhiều dịch vụ lưu trữ mã nguồn như GitHub, GitLab, Bitbucket, để giúp các nhà phát triển làm việc và quản lý mã nguồn một cách dễ dàng.

### 2.3. Ưu điểm và nhược điểm của Git

Dựa vào những đặc tính và sự phổ biến của Git đối với người dùng, nhất là lập trình viên, ta sẽ đưa ra những lợi ích khi sử dụng Git. Tuy nhiên, một công cụ hay phần mềm nào cũng đều có lợi và có hại. Vì vậy chúng ta sẽ tìm hiểu ưu và nhược điểm của nó.

| Ưu điểm | Nhược điểm |
| --- | --- |
| - Hệ thống quản lý phiên bản phân tán (DVCS) với tính năng nhánh và định nghĩa linh hoạt.<br>- Tốc độ xử lý nhanh hơn so với nhiều hệ thống quản lý phiên bản khác.<br>- Khả năng xử lý các xung đột đến từ sự thay đổi dữ liệu giữa các nhánh một cách tốt hơn.<br>- Tính bảo mật cao với các tính năng chống phá hoại, sao chép dự phòng và mã hóa.<br>- Sử dụng dễ dàng và hỗ trợ đa nền tảng. | - Học cú pháp Git ban đầu có thể khó khăn đối với người mới bắt đầu.<br>- Quản lý phiên bản phân tán có thể dẫn đến việc tăng kích thước repository nhanh hơn so với hệ thống quản lý phiên bản tập trung.<br>- Không có tính năng giải quyết xung đột tự động hoàn toàn, yêu cầu sự can thiệp của người dùng.<br>- Không thể xóa các commit trước đó một cách dễ dàng mà không ảnh hưởng đến lịch sử phiên bản.<br>- Có thể gặp phải các vấn đề liên quan đến hiệu suất nếu repository quá lớn hoặc có quá nhiều branch. |

# CHƯƠNG II: CÀI ĐẶT GIT

## 1. CÀI ĐẶT GIT TRÊN WINDOWS

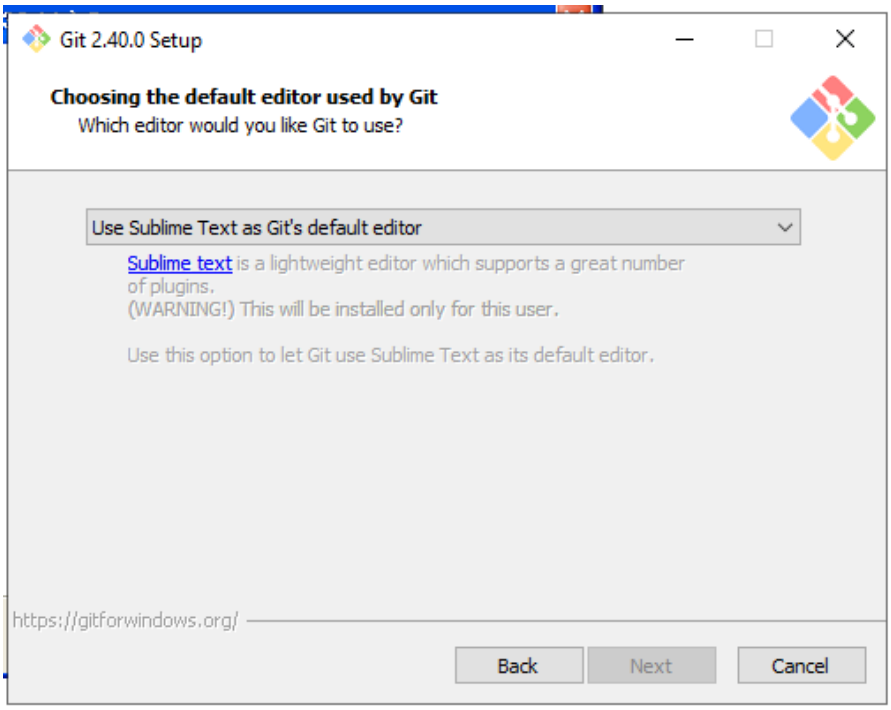
Windows vốn là hệ điều hành quen thuộc với đại đa số người dùng.

- Bước 1: Truy cập vào đường dẫn: **https://gitforwindows.org/** và sau đó ấn vào chữ Download.


*Hình 1: Cài đặt Git trên Windows.*

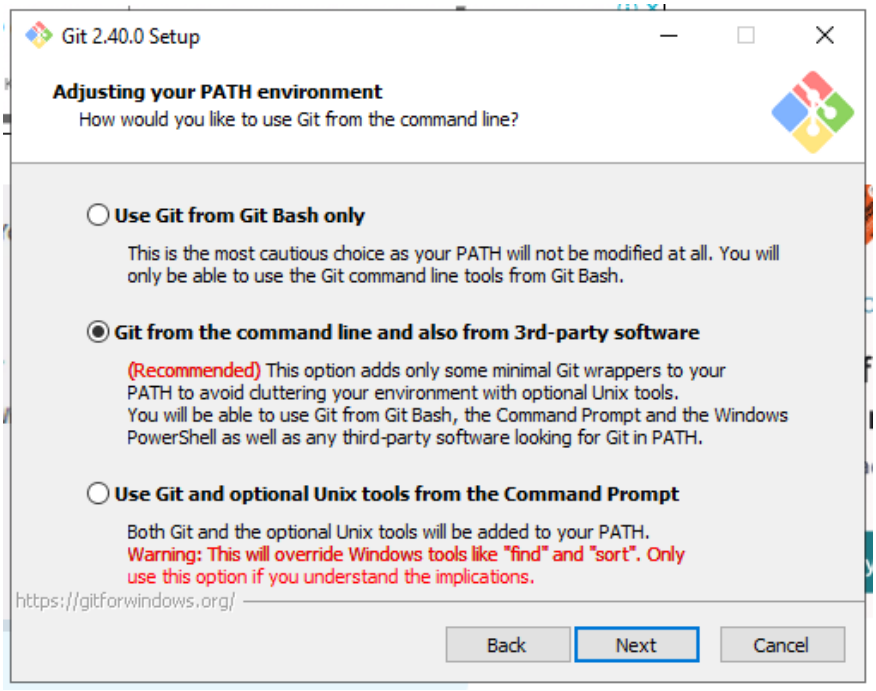
- Bước 2: Sau khi đã tải xuống thành công, click vào file để tiến hành cài đặt.
- Bước 3: Cài đặt bình thường như các phần mềm khác, về các options thì dựa vào yêu cầu của mỗi người để có những lựa chọn phù hợp.

- Bước 4: Cài đặt Git Editor cho Windows, có thể là các text editor như Vin, Notepad++, Sublime Text,…



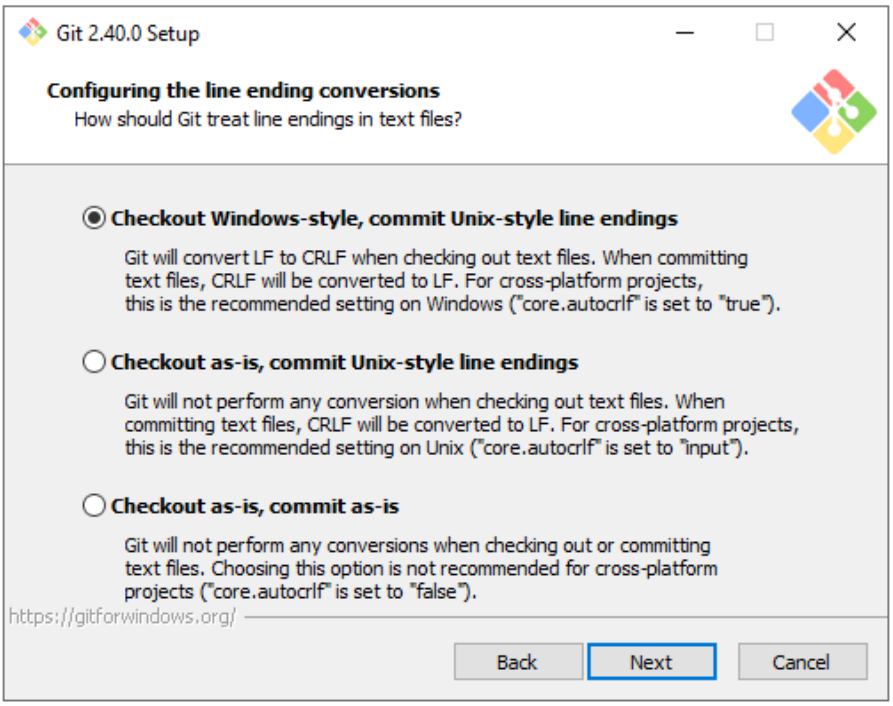

*Hình 2: Cài đặt Git trên Windows.*

- Bước 5: Lựa chọn PATH môi trường phù hợp với nhu cầu.



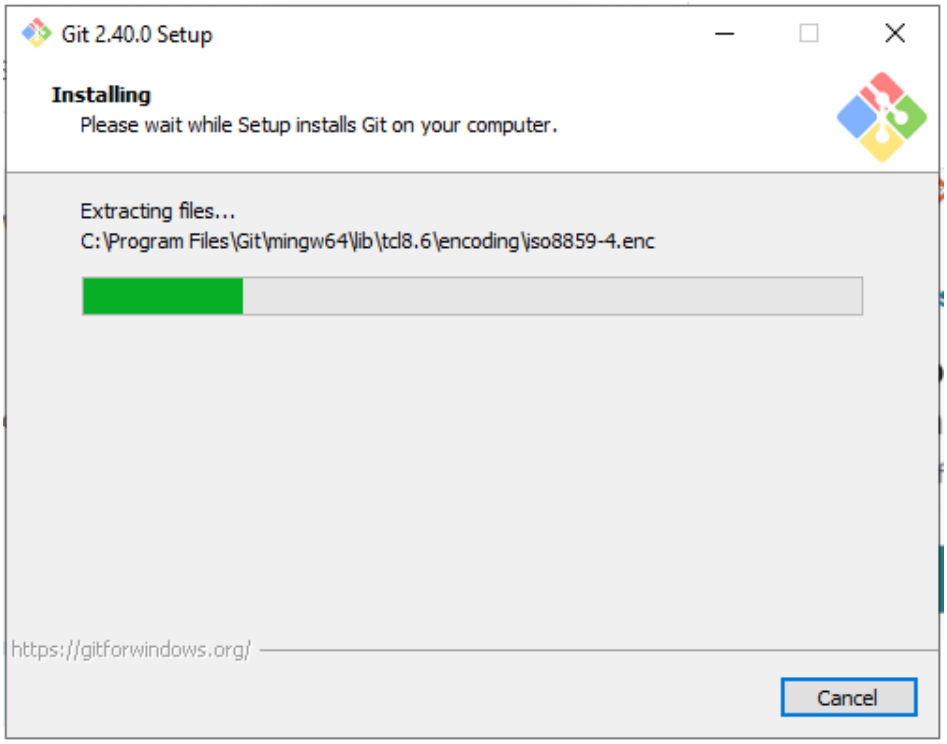

*Hình 3: Cài đặt Git trên Windows.*

- Bước 6: Chọn phong cách dòng lệnh.



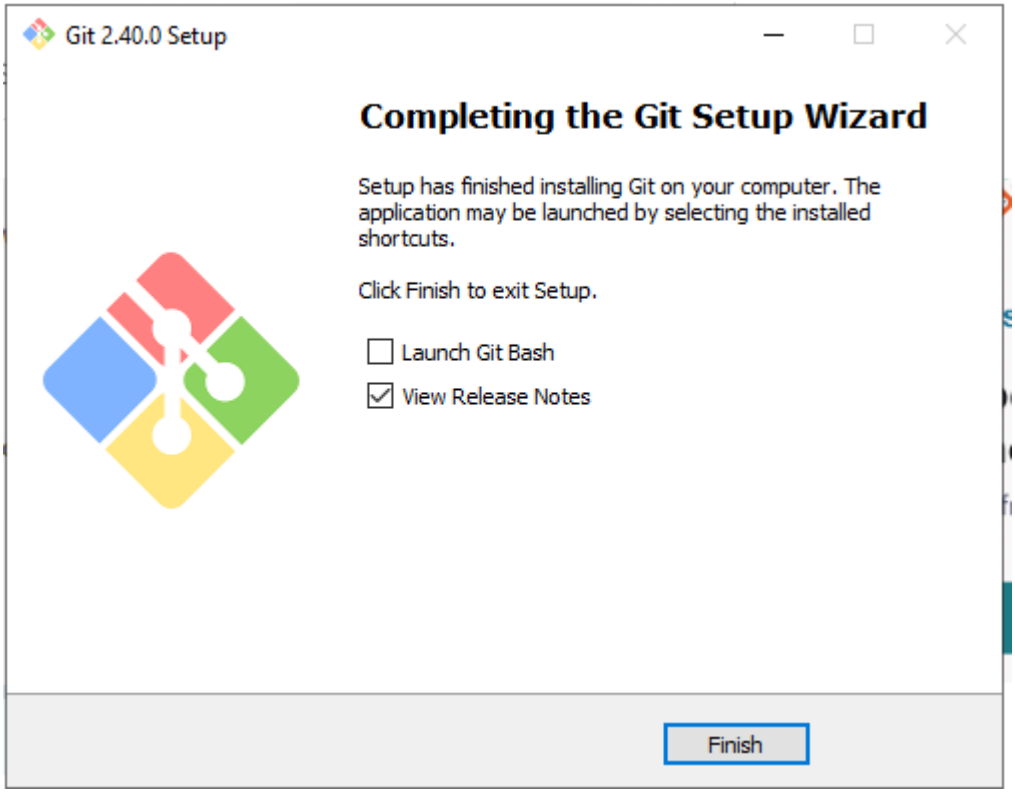

*Hình 4: Cài đặt Git trên Windows.*

- Bước 7: Bấm Install để cài đặt.




*Hình 5: Cài đặt Git trên Windows.*

- Bước 8: Kết thúc sau khi đã cài đặt thành công.




*Hình 6: Cài đặt Git trên Windows.*

## 2. CÀI ĐẶT GIT TRÊN LINUX

Hệ điều hành Linux hỗ trợ Git rất tốt, đặc biệt là với công cụ Terminal mặc định của Linux, giúp Git phát huy được tối đa sức mạnh vốn có. Git sẽ có nhiều cách cài đặt khác nhau dựa trên từng Distro từ nhân Linux. Người dùng truy cập vào đường dẫn: https://git-scm.com/download/linux và làm theo đường dẫn sau.

- **Bước 1:** Mở Terminal và gõ 2 lệnh sau.

```
Sudo apt-get update
Sudo apt-get install git
```

*Hình 7: Cài đặt Git trên Linux.*

- **Bước 2:** Gõ lệnh '**git-version**' để kiểm tra đã cài đặt thành công hay chưa.

```
student@nathanlee:~$ git --version
git version 2.20.1
```

*Hình 8: Cài đặt Git trên Linux.*

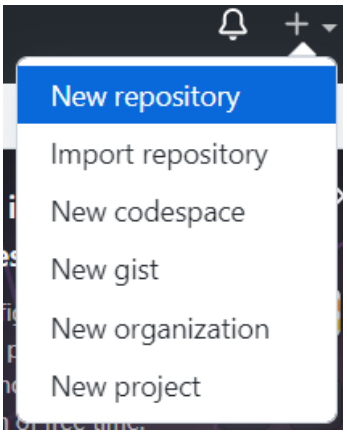
# CHƯƠNG III: CẤU HÌNH

## 1. CẤU HÌNH GITHUB

### 1.1. Tạo Github Repository

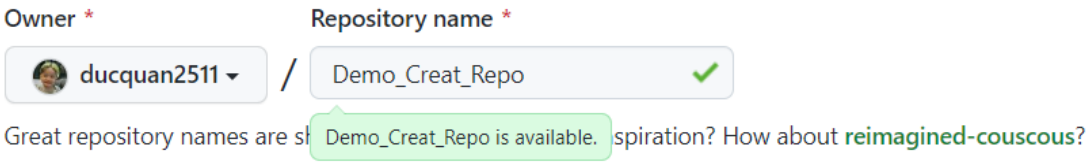
Để làm việc, trước hết cần tạo một kho lưu trữ mã nguồn lệnh cho dự án, việc này về bản chất cũng giống như thao tác trên các dòng lệnh Git. Ở đây, người dùng sẽ làm qua giao diện với các cú nhấp chuột. Các bước tạo Repo ban đầu được mô tả như sau:

- **Bước 1:** Người dùng bấm vào ký hiệu dấu cộng been cạnh ảnh đại diện của tài khoản và chọn New Repository, Github sẽ chuyển qua trang khởi tạo Repo mới.




*Hình 9: Khởi tạo Github Repository.*

- **Bước 2:** Đặt tên Repo cũng như miêu tả dự án cần thiết.

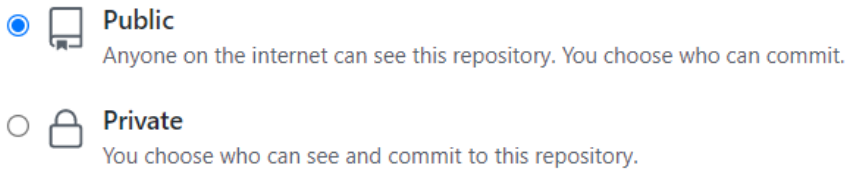



*Hình 10: Đặt tên cho Repository.*


*Hình 11: Mô tả Repository dự án.*

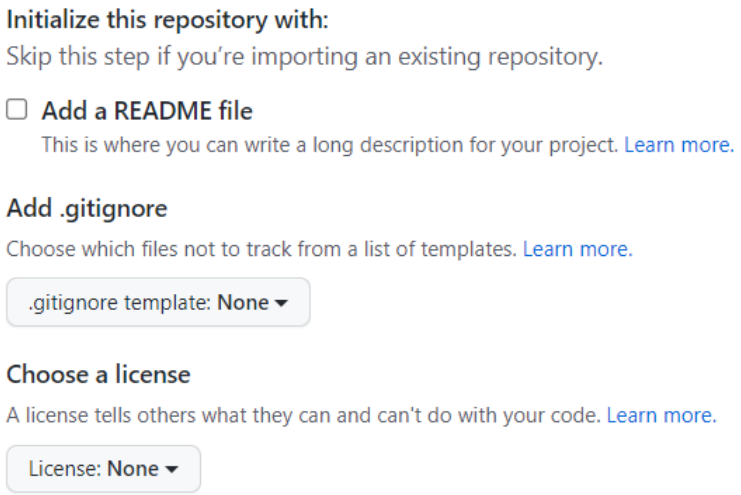
- **Bước 3:** Chọn một trong hai tùy chọn do Github đưa ra. Hai tùy chọn có nội dung như sau:

- Hai chế độ phân quyền là Public và Private. Public là chế độ mặc định, cho phép bất cứu ai cũng có thể xem được Repo. Nếu người tạo không muốn công khai có thể chuyển sang chế độ Private.



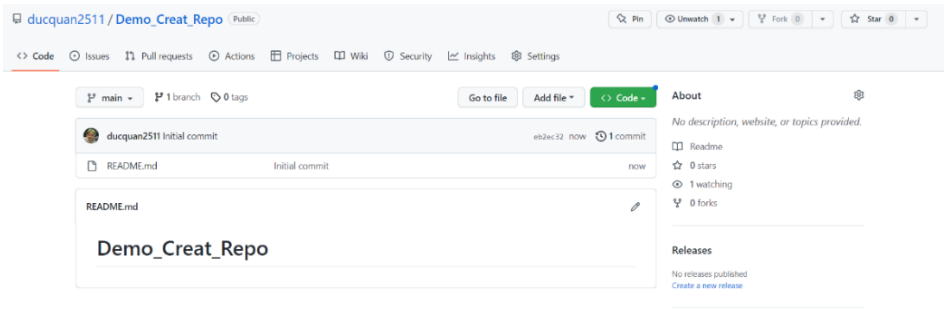

*Hình 12: Hai chế độ phân quyền của Repository trên Github.*

- Tập tin README giới thiệu Repo kèm một tệp tin .gitgnore. Đây là một template có sẵn trong Github và cho mọi người tùy chọn.



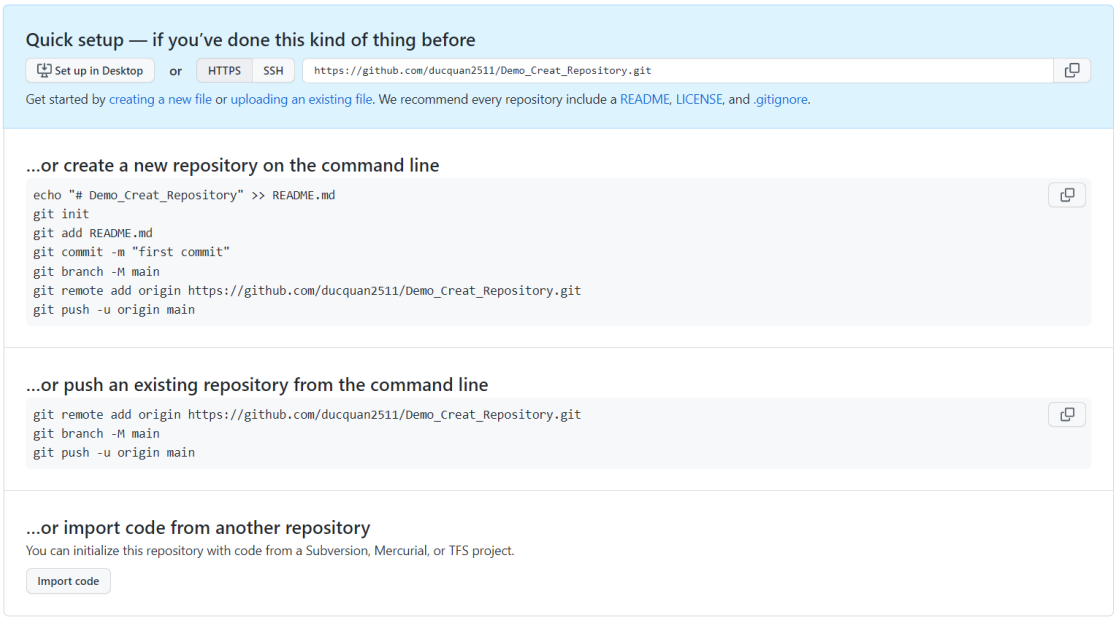

*Hình 13: Template tùy chọn Repository Github.*

- **Bước 4:** Bấm nút Creat repository tạo Repo, Github sẽ chuyển trang Repo đã hoàn thành.




*Hình 14: Repository sau khi khởi tạo.*

Trong trưởng hợp người dùng không thêm tùy chọn, Github sẽ mặc định đây là người dùng mới với dịch vụ nó sẽ chuyển đến một trang hướng dẫn cho người dùng này.




*Hình 15: Trang hướng dẫn dành cho người dùng.*

### 1.2. Xóa Github Repository

Nếu có nhu cầu xóa Repo thì yêu cầu tài khoản người dùng phải có đặc quyền hoặc là Admin cho một Repo của tổ chức. Các bước xóa Repo trên Github rất đơn giản nhưng có một vài điều cần lưu ý là tất cả các tệp đính kèm cùng phân quyền nhóm dự án đều bị xóa bỏ.

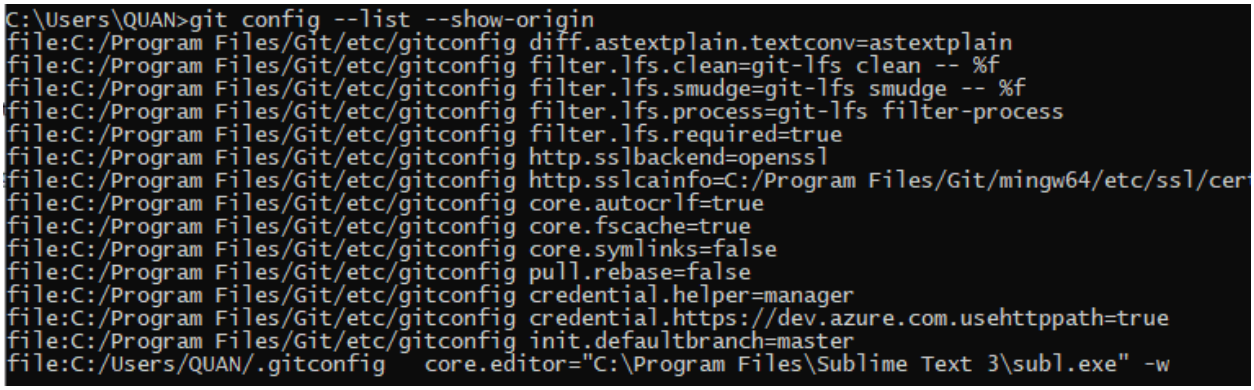
Người dùng có thể khôi phục lại hành động xóa Repo trong 90 ngày.

## 2. CẤU HÌNH GIT CƠ BẢN

### 2.1. Thiết lập ban đầu

Điều đầu tiên cần làm sau khi cài đặt git là cấu hình chứng thực, thiết lập các tùy chọn ban đầu phù hợp với cá nhân người sử dụng.

Git có một số công cụ dòng lệnh là git config để người dùng tuy chỉnh các giá trị để git có giao diện và vận hành như họ muốn. Muốn xác định người dùng cần gõ lệnh git config '**–list –show-origin**', nó sẽ hiện thị file config được lưu trên đường dẫn nào, cũng như các giá trị đã được cấu hình mặc định bởi Git.




*Hình 16: Xem thông số mặc định.*

### 2.2. Danh tính của bạn

Thông tin đầu tiên cần phải cấu hình trước khi sử dụng Git là tên và địa chỉ email của bạn. Git sẽ sử dụng hai thông tin này để gắn vào mỗi lệnh commit (mục đích để biết ai đã thực hiện commit).

Nên cấu hình danh tính của bạn ở mức global, và chỉ làm một lần. Git sẽ sử dụng thông tin này cho tất cả các thao tác bạn làm trên hệ thống. Nếu muốn sử dụng tên và email khác cho một dự án riêng, thì thực hiện thêm việc cấu hình danh tính tại mức local cho chính dự án đó.

Ví dụ: *$ git config --global user.name "Nam Cong"*
*$ git config --global user.email langyentrach@gmail.com*

Mở **.gitconfig** để kiểm tra thông tin vừa cấu hình:
*[user]*
*name = Nam Cong*
*email = langyentrach@gmail.com*

### 2.3. Kiểm tra cài đặt

Git sẽ đọc một giá trị đến từ các file khác nhau, người dùng sẽ gõ lệnh '**git config –list**' để liệt kê các thiết lập mà Git tìm thấy.

```
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
core.editor="C:\Program Files\Sublime Text 3\subl.exe" -w
gui.recentrepo=C:/Users/QUAN/namcong
user.name=Nam cong
user.email=langyentrach@gmail.com
```

*Hình 17: Xem thiết lập.*

# CHƯƠNG IV: CÁC LỆNH CƠ BẢN TRONG GIT

## 1. GIT INIT

- **Git Init:** Khởi tạo một kho lưu trữ Git mới trong thư mục hiện tại.
- Khi sử dụng câu lệnh git init, Git sẽ tạo ra một thư mục ẩn có tên .git trong thư mục hiện tại của bạn. Thư mục này sẽ chứa toàn bộ các thông tin cần thiết để quản lý phiên bản của dự án của bạn, bao gồm các commit, nhánh, và các thiết lập khác.
- **Ví dụ:** Để khởi tạo một kho lưu trữ Git mới cho một dự án đang được lưu trữ trong thư mục "my_project", bạn có thể chạy lệnh sau:

```
QUAN@DESKTOP-6EJ7BFM MINGW64 ~
$ cd my_project

QUAN@DESKTOP-6EJ7BFM MINGW64 ~/my_project
$ git init
Initialized empty Git repository in C:/Users/QUAN/my_project/.git/
```

*Hình 18: Git init.*

## 2. GIT CLONE

- **Git clone:** Sao chép một kho lưu trữ Git từ một kho lưu trữ trực tuyến sang máy tính cá nhân của bạn.
- Câu lệnh git clone được sử dụng để sao chép một kho lưu trữ Git từ một kho lưu trữ trực tuyến, chẳng hạn như GitHub hoặc Bitbucket, sang máy tính cá nhân của bạn. Khi thực thi lệnh, Git sẽ tải về toàn bộ lịch sử của kho lưu trữ đó và tạo ra một bản sao của nó trên máy tính của bạn.
- **Ví dụ:** Để sao chép một kho lưu trữ Git từ GitHub sang máy tính của bạn, bạn có thể chạy lệnh sau trong dòng lệnh:

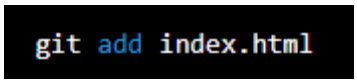
```
QUAN@DESKTOP-6EJ7BFM MINGW64 ~/my_project (master)
$ git clone https://github.com/username/repo.git
```

*Hình 19: Git clone.*

Trong đó, https://github.com/username/repo.git là đường dẫn đến kho lưu trữ trên GitHub mà bạn muốn sao chép. Git sẽ tải về toàn bộ lịch sử của kho lưu trữ đó và tạo ra một bản sao của nó trên máy tính của bạn.

## 3. GIT ADD

- **Git Add:** Thêm các tệp đã sửa đổi hoặc tạo mới vào mục theo dõi của Git.
- Câu lệnh git add được sử dụng để thêm các tệp đã sửa đổi hoặc tạo mới vào mục theo dõi của Git. Khi thực thi lệnh, Git sẽ đánh dấu các tệp đã được thêm vào mục theo dõi và chuẩn bị cho việc tạo commit.
- **Ví dụ:** Để thêm tệp "index.html" vào mục theo dõi của Git, bạn có thể chạy lệnh sau trong dòng lệnh:




*Hình 20: Git add.*

Nếu bạn muốn thêm tất cả các tệp đã sửa đổi hoặc tạo mới trong thư mục hiện tại vào mục theo dõi của Git, bạn có thể chạy lệnh sau:


*Hình 21: Git add.*

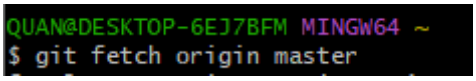
## 4. GIT COMMIT

- **Git Commit:** Tạo một "commit" mới để lưu trữ các thay đổi vào kho lưu trữ.
- Các câu lệnh **Git Commit**:
  - **Git commit:** Tạo một commit mới với các thay đổi đã được đưa vào staged area, và hiển thị trình soạn thảo để nhập thông điệp commit.
  - **Git commit -m “message”:** Tạo một commit mới và nhập thông điệp commit trực tiếp trong lệnh.
  - **Gid commit -a:** Tạo một commit mới với các thay đổi đã được thêm và commit các tệp tin đã được chỉnh sửa hoặc đã bị xóa mà không cần sử dụng lệnh git add.
  - **Git commit --amend:** Chỉnh sửa commit cuối cùng. Lệnh này cho phép bạn thay đổi thông điệp commit hoặc thay đổi các thay đổi đã được thêm vào commit cuối cùng.

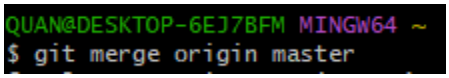
- **Git commit -s:** Tạo một commit ký số với chữ ký GPG. Lệnh này được sử dụng khi bạn muốn xác minh tính toàn vẹn của commit và đảm bảo rằng nó chưa bị sửa đổi.
- **Git commit --allow-empty:** Tạo một commit trống, không có thay đổi nào được thêm vào. Lệnh này có thể hữu ích khi bạn muốn thêm một commit để đánh dấu một sự kiện quan trọng, nhưng không có thay đổi nào cần được thêm.
- **Git commit --squash:** Tạo một commit mới và gộp nó vào commit trước đó. Lệnh này cho phép bạn gộp nhiều commit thành một commit lớn hơn, dễ đọc hơn và dễ hiểu hơn.

## 5. GIT PULL

- **Git Pull:** Đẩy các commit mới lên một kho lưu trữ trực tuyến, chẳng hạn như GitHub hoặc Bitbucket. Trong đó, **"pull"** là một hành động để lấy các thay đổi từ một repository khác và đồng bộ hóa chúng với repository hiện tại. Khi bạn sử dụng lệnh pull, Git sẽ tự động tìm kiếm các thay đổi mới nhất trên repository khác và cập nhật chúng vào repository hiện tại. Quá trình này sẽ tự động thực hiện merge giữa các thay đổi đó với các thay đổi đã tồn tại trong repository hiện tại, nếu có.
- **Git pull** thực ra là viết tắt của **"git pull origin master"**. Trong đó:
  - **Origin** là tên của kho chứa từ xa (hay remote repository).
  - **Master** là tên của nhánh trên kho chứa từ xa. Một kho chứa có thể có nhiều nhánh khác nhau.
- Để liệt kê các kho chứa từ xa bạn có thể dùng câu lệnh sau: **git remote -v**
- Để liệt kê các nhánh của các kho chứa: **git branch –a**
- Về bản chất khi chạy câu lệnh **git pull origin master** thực sự là bạn đang sử dụng hai câu lệnh phía sau:




*Hình 22: Git fetch.*




*Hình 23: Git merge.*

- Câu lệnh **"git fetch origin master"** sẽ truy vấn thông tin của kho chứa từ xa trên máy chủ remote và sau đó kéo về máy local những thay đổi này. Tiếp đó câu lệnh trên sẽ thực hiện việc so sánh những thay đổi mới kéo về máy local và hiển thị thông tin.
- Câu lệnh **"git merge orign master"** sẽ gộp những thay đổi mới kéo về (dùng câu lệnh **git fetch** ở trên) từ máy chủ từ xa với nhánh hiện tại trên máy local.

## 6. GIT PUSH

- **Git Push:** Lấy các commit mới từ kho lưu trữ trực tuyến và áp dụng chúng vào nhánh hiện tại. Trong đó, **"push"** được sử dụng để đẩy các thay đổi từ repository hiện tại lên repository khác. Thường thì push được sử dụng khi bạn đã hoàn thành một số thay đổi trên local repository của mình và muốn chia sẻ những thay đổi này với các thành viên khác trong nhóm hoặc lưu trữ trên một remote server. Khi sử dụng lệnh push, Git sẽ đẩy các thay đổi được commit từ local repository lên repository khác, đồng thời cập nhật các thay đổi trên các nhánh của repository hiện tại.
- Sau khi tạo commit bằng câu lệnh **git commit** thì thực tế commit của bạn mới chỉ được lưu lại ở máy local. Để những thay đổi này được đấy lên máy chủ từ xa của Git (hay remote repo) thì bạn cần sử dụng câu lệnh **git push** sau: **$ git push origin master**

## 7. GIT BRANCH

- **Git Branch:** Xem danh sách các nhánh hiện có và tạo ra một nhánh mới.
- Để xem danh sách các nhánh hiện có trong kho lưu trữ, chúng ta có thể chạy lệnh sau: **git branch**. Dòng lệnh này sẽ hiển thị danh sách các nhánh hiện có, với dấu sao (*) nhằm chỉ ra nhánh đang được sử dụng.
- Để tạo ra một nhánh mới, chúng ta có thể sử dụng lệnh sau: **demo_project**.
- Sau khi tạo ra nhánh mới, ta có thể chuyển sang nhánh đó bằng câu lệnh: **git checkout**.

## 8. GIT MERGE

- **Git Merge:** Kết hợp các thay đổi từ một nhánh khác vào nhánh hiện tại.
- Câu lệnh **git merge** trong Git được sử dụng để kết hợp các thay đổi từ một nhánh khác vào nhánh hiện tại. Thông thường, bạn sử dụng git merge để hợp nhất các thay đổi từ một nhánh đặc biệt (ví dụ nhánh feature hoặc release) vào nhánh chính (ví dụ nhánh master).
- Để kết hợp các thay đổi từ một nhánh khác vào nhánh hiện tại, ta có thể sử dụng lệnh sau: "**git merge <ten-nhanh-khac>"**. Trong đó **<ten-nhanh-khac>** là tên của nhánh khác mà ta muốn hợp nhất vào nhánh hiện tại. Lệnh trên sẽ kết hợp tất cả các thay đổi từ nhánh khác vào nhánh hiện tại và tạo ra một commit mới.
- Nếu có xung đột giữa các thay đổi trên hai nhánh, Git sẽ hiển thị thông báo và yêu cầu chúng ta giải quyết xung đột trước khi tiếp tục kết hợp.
- **Ví dụ:** Để kết hợp các thay đổi từ nhánh feature-branch vào nhánh master, ta có thể chạy lệnh sau:

```
git checkout master
git merge feature-branch
```

*Hình 24: Git merge.*

Lệnh trên sẽ chuyển sang nhánh master và kết hợp tất cả các thay đổi từ nhánh feature-branch vào nhánh master.

## 9. GIT CHECKOUT

- **Git Checkout:** Chuyển đổi giữa các nhánh hoặc khôi phục các tệp đã bị xóa.
- Để chuyển đổi giữa các nhánh, bạn có thể sử dụng lệnh sau: **git checkout <ten-nhanh>**. Trong đó, **<ten-nhanh>** là tên của nhánh mà bạn muốn chuyển sang. Lệnh trên sẽ chuyển đổi kho lưu trữ sang nhánh đó và cập nhật các tệp trong thư mục làm việc để phản ánh trạng thái của nhánh đó.

## 10.GIT STATUS

- Câu lệnh git status trong Git được sử dụng để xem trạng thái hiện tại của mục theo dõi của Git. Nó cho phép bạn biết được các tệp nào đã thay đổi, đã được thêm vào hoặc đã được xóa khỏi kho lưu trữ của bạn.
- Khi bạn chạy lệnh git status, Git sẽ trả về một danh sách các tệp đã được sửa đổi, đã được thêm vào hoặc đã bị xóa khỏi kho lưu trữ của bạn. Nó cũng sẽ cho bạn biết trạng thái của các tệp này, chẳng hạn như đã được sửa đổi nhưng chưa được đánh dấu để được tạo commit, đã được đánh dấu để được tạo commit hoặc đã được xóa khỏi kho lưu trữ của bạn.
- **Ví dụ:** Nếu bạn thêm một tệp mới vào kho lưu trữ của bạn, khi bạn chạy lệnh git status, bạn sẽ thấy tên của tệp đó được hiển thị với trạng thái untracked. Nghĩa là Git chưa bắt đầu theo dõi tệp này và nó chưa được đánh dấu để được tạo commit.

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_file.txt

nothing added to commit but untracked files present (use "git add" to track)
```

*Hình 25: Git status.*

Nếu bạn sửa đổi một tệp đã được theo dõi và chưa được đánh dấu để được tạo commit, khi bạn chạy lệnh git status, bạn sẽ thấy tên của tệp đó được hiển thị với trạng thái modified. Nghĩa là tệp đã được sửa đổi và đang chờ để được đánh dấu để được tạo commit.

```
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified_file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

*Hình 26: Git status.*

Nếu bạn đã thêm một tệp và đánh dấu nó để được tạo commit, khi bạn chạy lệnh git status, bạn sẽ thấy tên của tệp đó được hiển thị với trạng thái changes to be committed. Nghĩa là tệp đã được đánh dấu để được tạo commit và sẽ được bao gồm trong commit tiếp theo.

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new_file.txt
```

*Hình 27: Git status.*

# 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.

```
QUAN@DESKTOP-6EJ7BFM MINGW64 ~
$ mkdir project_test

QUAN@DESKTOP-6EJ7BFM MINGW64 ~
$ cd project_test
```

*Hình 28: Tạo thư mục cho repository của bạn trên máy tính.*

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

```
QUAN@DESKTOP-6EJ7BFM MINGW64 ~/project_test
$ git init
Initialized empty Git repository in C:/Users/QUAN/project_test/.git/

QUAN@DESKTOP-6EJ7BFM MINGW64 ~/project_test (master)
$
```

*Hình 29: Khởi tạo Git repository trong thư mục.*

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

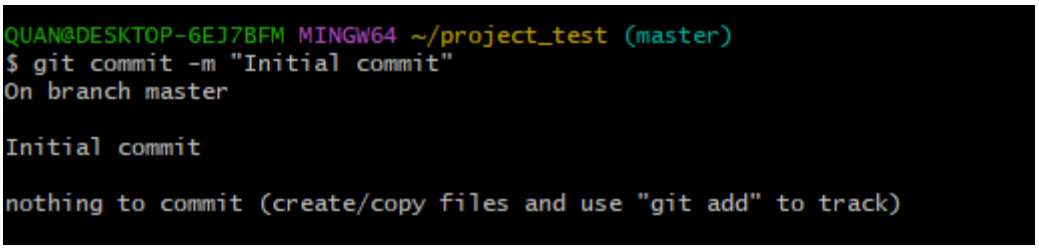
- **Bước 3:** Sau khi khởi tạo repository, ta có thể bắt đầu thêm các tệp và thư mục vào repository bằng lệnh git add. Nếu cần thêm tất cả các tệp và thư mục trong thư mục hiện tại vào repository, ta có thể sử dụng lệnh sau:

```
QUAN@DESKTOP-6EJ7BFM MINGW64 ~/project_test (master)
$ git add .
```

*Hình 30: Git add.*

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:




*Hình 31: Git commit.*

## 2. 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.

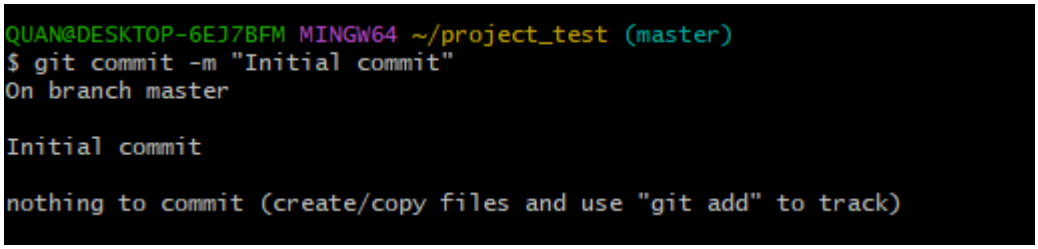
Một commit trong Git là một trạng thái của repository tại một thời điểm nhất định và được định danh bằng một mã hash duy nhất. Khi bạn tạo một commit, Git sẽ lưu trữ các thay đổi của bạn trên một nhánh (branch) riêng biệt, vì vậy bạn có thể quay lại trạng thái trước đó của repository nếu cần thiết.

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.


*Hình 32: 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:




*Hình 33: Git commit.*

## 3. XEM LỊCH SỬ COMMIT

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.)

| Lựa chọn | Mô tả |
|---|---|
| `-(n)` | Chỉ hiển thị n commit mới nhất |
| `--since, --after` | Giới hạn các commit được thực hiện sau ngày nhất định. |
| `--until, --before` | Giới hạn các commit được thực hiện trước ngày nhất định. |
| `--author` | Chỉ hiện thị các commit mà tên tác giả thoả mãn điều kiện nhất định. |
| `--committer` | Chỉ hiện thị các commit mà tên người commit thoả mãn điều kiện nhất định. |

*Hình 34: Bảng liệt kê các lựa chọn.*

- **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:

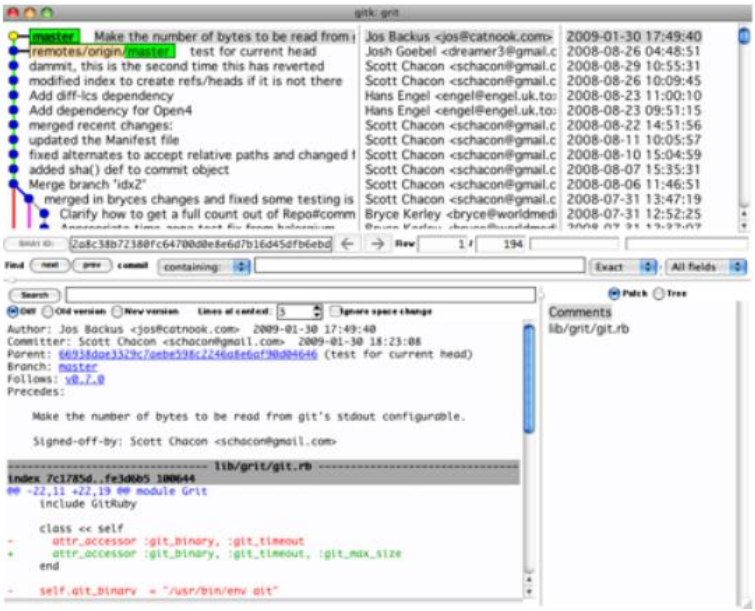
```
$ git log --pretty="%h - %s" --author=gitster --since="2008-10-01" \
   --before="2008-11-01" --no-merges -- t/
5610e3b - Fix testcase failure when extended attribute
acd3b9e - Enhance hold_lock_file_for_{update,append}()
f563754 - demonstrate breakage of detached checkout wi
d1a43f2 - reset --hard/read-tree --reset -u: remove un
51a94af - Fix "checkout --track -b newbranch" on detac
b0ad11e - pull: allow "git pull origin $something:$cur
```

*Hình 35: Git log.*

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.

## 4. HIỂN THỊ LỊCH SỬ TRÊN GIAO DỊCH

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.




*Hình 36: Công cụ trực quan hóa lịch sử commit git.*

## 5. PHỤC HỒI

### 5.1. Thay đổi 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:

```
$ git commit -m 'initial commit'
$ git add forgotten_file
$ git commit --amend
```

*Hình 37: Thay đổi commit cuối cùng.*

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.

### 5.2. 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:

```
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   benchmarks.rb
#
```

*Hình 38: Phục hồi tệp tin đã thay đổi.*

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:

```
$ git checkout -- benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#
```

*Hình 39: Phục hồi tệp tin đã thay đổi.*

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.

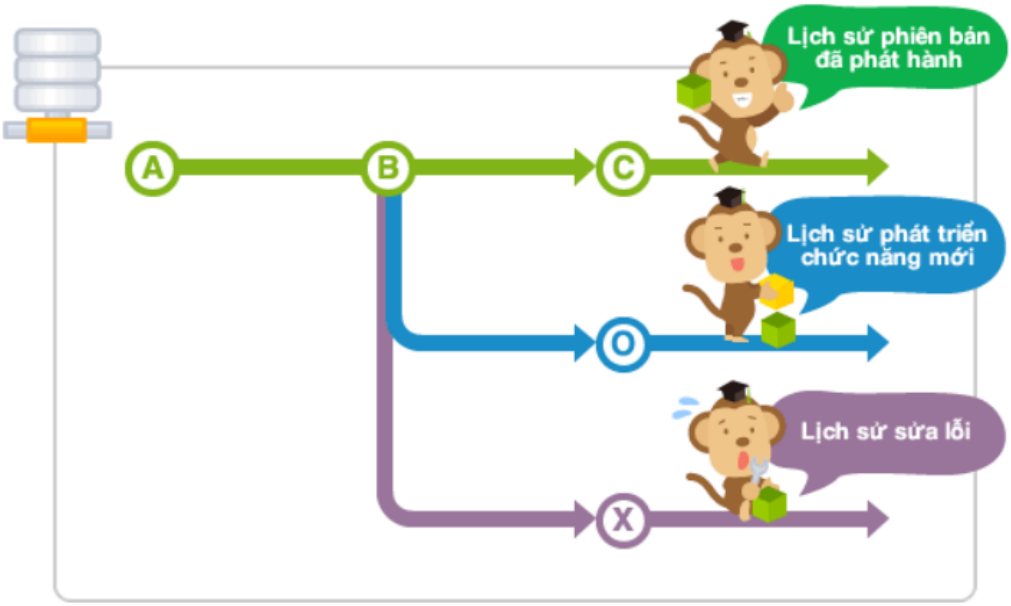
Bất cứ thứ gì được commit vào Git luôn có thể phục hồi lại. Thậm chí cả các commit ở các nhánh đã bị xoá hoặc bị ghi đè bởi –amend.

## 6. 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.




*Hình 40: Quy trình phân chia nhánh.*

## 6.1. Làm việc với nhánh

### 6.1.1. Tạo nhánh

- **Bước 1: Kiểm tra nhánh**

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git branch
* master
```

Chỉ có một nhánh Master (nhánh gốc)

- **Bước 2: Phân nhánh**

Dùng lệnh git branch +<Tên nhánh>

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git branch b1
```

- **Bước 3: Kiểm tra nhánh đã tạo**

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git branch
  b1
* master
```

Như vậy nhánh b1 đã tạo thành công.

### 6.1.2. Phân nhánh

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

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git checkout b1
Switched to branch 'b1'
```

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

Dùng lệnh **git branch**

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (b1)
$ git branch
* b1
  master
```

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

### 6.1.3. Gộp nhánh

- **Bước 1: Chuyển nhánh**

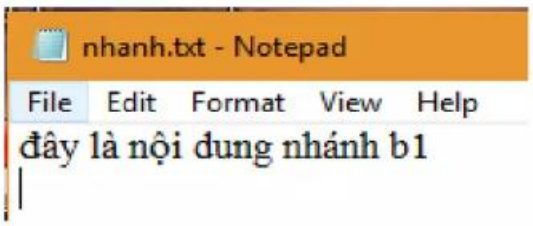
Chuyển sang bước b1

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git checkout b1
Switched to branch 'b1'
```

- **Bước 2: Tạo file text có tên "nhanh.txt"**

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (b1)
$ cat > nhanh.txt
nhanh.txt
```

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:




- **Bước 3: Thêm file "nhanh.txt" vào nhánh b1**

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (b1)
$ git add nhanh.txt
```

- **Bước 4: Ghi chú**

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (b1)
$ git commit -m "file b1"
[b1 59590bc] file b1
 1 file changed, 2 insertions(+)
 create mode 100644 nhanh.txt
```

- **Bước 5: Kiểm tra**

Lệnh ls dùng kiểm tra file hiện có nhánh b1

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (b1)
$ ls
download  new.txt  nhanh.txt  screenshot1.py  Svn/  test  upload.txt
```

Kiểm tra nhánh Master

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ ls
download  new.txt  screenshot1.py  Svn/  test  upload.txt
```

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

Đầu tiên chuyển sang nhánh Master

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (b1)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

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

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git merge b1
Updating 005098c..59590bc
Fast-forward
 nhanh.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 nhanh.txt
```

Tiếp theo kiểm tra các file nhánh Master

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ ls
download  new.txt  nhanh.txt  screenshot1.py  Svn/  test  upload.txt
```

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

### 6.1.4. Xóa nhánh

- **Bước 1: Chuyển sang nhánh Master**

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (b1)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

- **Bước 2: Xóa nhánh b1**

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

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git branch -d b1
Deleted branch b1 (was 59590bc).
```

- **Bước 3: Kiểm tra**

Dùng lệnh git branch để xem nhánh hiện có

```
HP APPLE@HPAPPLE MINGW64 ~/Documents/Github/test (master)
$ git branch
* master
```

Chỉ còn nhánh gốc là Master.

## 6.2. Quản lý các nhánh

- Để xem commit mới nhất trên từng nhánh, bạn có thể chạy lệnh git branch -v :

```
$ git branch -v
  iss53   93b412c fix javascript issue
* master  7a98805 Merge branch 'iss53'
  testing 782fd34 add scott to the author list in the readmes
```

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

```
$ git branch --merged
  iss53
* master
```

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

```
$ git branch --no-merged
  testing
```

- 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:

```
$ git branch -d testing
error: The branch 'testing' is not an ancestor of your current HEAD.
If you are sure you want to delete it, run 'git branch -D testing'.
```

## 7. GIT MERGE

### 7.1. 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. Ta 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

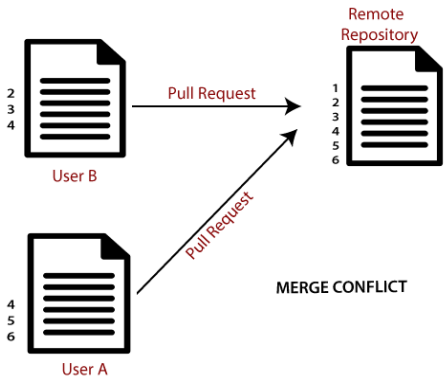
  **Git merge master**

- Git merge nhánh:

Git cho phép gộp toàn bộ nhánh này vào nhánh khác. Giả sử ta 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 ta làm như vậy.

### 7.2. Xung đột khi git merge

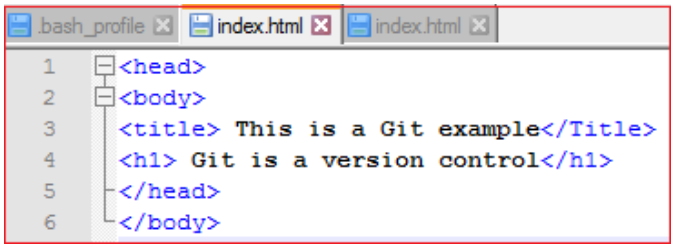
Khi hai nhánh đang cố gắng hợp nhất và cả hai đều được chỉnh sửa cùng một lúc và trong cùng một file, Git sẽ không thể xác định phiên bản nào sẽ thực hiện để thay đổi. Tình huống như vậy được gọi là xung đột hợp nhất. Nếu tình huống như vậy xảy ra, nó sẽ dừng ngay trước khi hợp nhất commit để ta có thể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




*Hình 41: Sơ đồ xung đột khi git merge.*

- **Ví dụ:**

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.



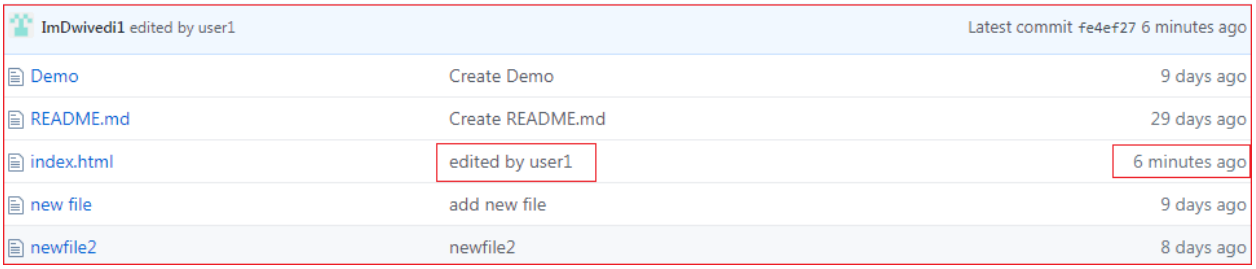

Cập nhật nó trong kho lưu trữ cục bộ với sự trợ giúp của lệnh git add.

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git add index.html
```

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

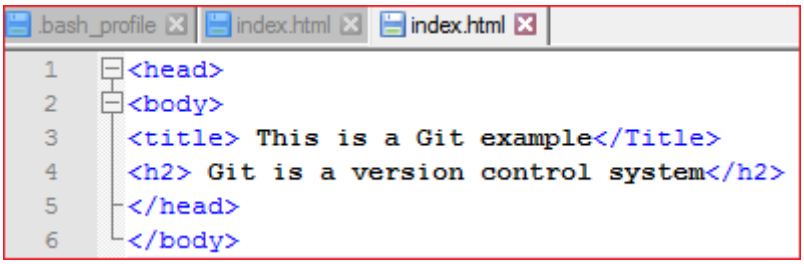
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git commit -m "edited by user1"
[master fe4ef27] edited by user1
 1 file changed, 1 insertion(+)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 345 bytes | 345.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ImDwivedi1/Git-Example
   039c01b..fe4ef27  master -> master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
```

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




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.




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

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git add index.html

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git commit -m " edited by user2"
[master 3ee71e0]  edited by user2
 1 file changed, 1 insertion(+)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git push origin master
To https://github.com/ImDwivedi1/Git-Example
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/ImDwivedi1/Git-Example'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$
```

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 chúng pull kho lưu trữ trước trước khi push. Xem lệnh dưới đây:

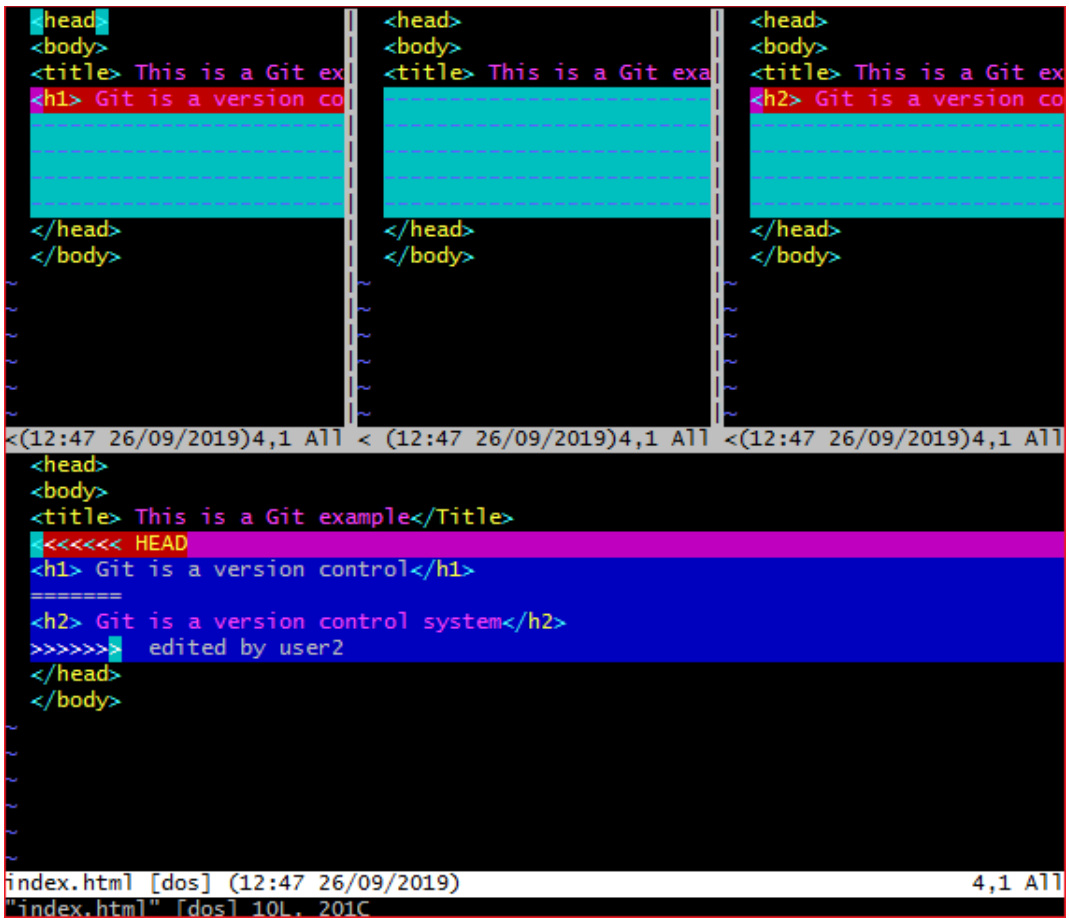
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git pull --rebase origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/Git-Example
 * branch            master     -> FETCH_HEAD
   039c01b..fe4ef27  master     -> origin/master
First, rewinding head to replay your work on top of it...
Applying:  edited by user2
Using index info to reconstruct a base tree...
M       index.html
Falling back to patching base and 3-way merge...
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0001  edited by user2
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort"
.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master|REBASE 1/1)
$
```

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>** .

### 7.3. 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, nó sẽ dẫn đến:




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:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master|REBASE 1/1)
$ git rebase --continue
Applying:  edited by user2

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 373 bytes | 124.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ImDwivedi1/Git-Example
   fe4ef27..b3db7dc  master -> master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$
```

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.

## 8. ĐÁNH DẤU – TAG

### 8.1. Liệt kê tag

Chỉ cần gõ **git tag:**

```
$ git tag
v0.1
v1.3
```

Lệnh này sẽ liệt kê các tag được sắp xếp theo thứ tự bảng chữ 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:

### 8.2. 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.

### 8.2.1. 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:

```
$ git tag -a v1.4 -m 'my version 1.4'
$ git tag
v0.1
v1.3
v1.4
```

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:

```
$ git show v1.4
tag v1.4
Tagger: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Feb 9 14:45:11 2009 -0800

my version 1.4
commit 15027957951b64cf874c3557a0f3547bd83b3ff6
Merge: 4a447f7... a6b4c97...
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sun Feb 8 19:02:46 2009 -0800

    Merge branch 'experiment'
```

### 8.2.2. 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:

```
$ git tag -s v1.5 -m 'my signed 1.5 tag'
You need a passphrase to unlock the secret key for
user: "Scott Chacon <schacon@gee-mail.com>"
1024-bit DSA key, ID F721C45A, created 2009-02-09
```

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

```
$ git show v1.5
tag v1.5
Tagger: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Feb 9 15:22:20 2009 -0800

my signed 1.5 tag
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.8 (Darwin)

iEYEABECAAYFAkmQurIACgkQON3DxfchxFr5cACeIMN+ZxLKggJQf0QYiQBwgySN
Ki0An2JeAVUCAiJ7Ox6ZEtK+NvZAj82/
=WryJ
-----END PGP SIGNATURE-----
commit 15027957951b64cf874c3557a0f3547bd83b3ff6
Merge: 4a447f7... a6b4c97...
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sun Feb 8 19:02:46 2009 -0800

    Merge branch 'experiment'
```

### 8.2.3. 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:

```
$ git tag v1.4-lw
$ git tag
v0.1
v1.3
v1.4
v1.4-lw
v1.5
```

Show commit:

```
$ git show v1.4-lw
commit 15027957951b64cf874c3557a0f3547bd83b3ff6
Merge: 4a447f7... a6b4c97...
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sun Feb 8 19:02:46 2009 -0800

    Merge branch 'experiment'
```

# KẾT LUẬN

Sau thời gian 2 tháng nghiên cứu với sự nỗ lực của nhóm và sự hướng dẫn tận tình của thầy Lương Công Duẩn, nhóm đã hoàn thành đề tài "Quản lý phiên bản trong thiết kế (Git)", nhóm xin tổng kết những gì đề tài đạt được như sau:

- Tìm hiểu khái niệm, đặc điểm và nguyên lý của Git.
- Nghiên cứu về khái niệm và nhiệm vụ, nguyên lý hoạt động.
- Xây dựng quy trình hướng dẫn sử dụng Git.
- Mô phỏng sử dụng Git và đánh giá kết quả.

Qua quá trình thực hiện đề tài, được sự hướng dẫn tận tình của thầy, nhóm đã được thử thách với một đề tài, kiến thức mới, học thêm được cách làm việc và tổ chức vấn đề khoa học hơn. Mặc dù đã có nhiều cố gắng nhưng nội dung khó tránh khỏi những sai sót, nhóm rất mong nhận được sự góp ý từ thầy giáo để có thể hoàn thiện hơn nữa.

Nhóm xin chân thành cảm ơn!

Nhóm sinh viên thực hiện

Nhóm 5