**1. What is VCS?**

- Giữ được mọi thứ trong phiên bản.

- Revert lại file ở phiên bản trước.

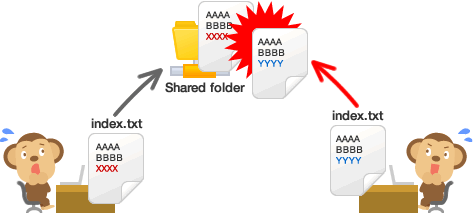
- Revert toàn bộ project ở phiên bản trước.

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.

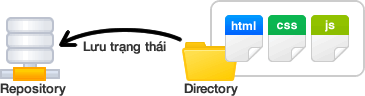
Thêm nữa, khi định ghi đè (overwrite) lên file mới nhất đã chỉnh sửa của người khác bằng file đã chỉnh sửa dựa trên file cũ, thì khi đăng (upload) lên server sẽ hiện ra cảnh cáo. Vì thế, sẽ không xảy ra thất bại về việc đã ghi đè lên nội dung chỉnh sửa của người khác mà không hề hay biết.

Khi quản lý file bằng Git, lịch sử cập nhật sẽ được lưu trong Git. Vì không cần copy trước file dùng để sao lưu (Backup) sẵn có nên rất thuận tiện.



**2. Repository quản lý lịch sử**

Repository là nơi sẽ ghi lại trạng thái của thư mục và file. Trạng thái được lưu lại đang được chứa như là lịch sử thay đổi của nội dung. Bằng việc đặt thư mục muốn quản lý lịch sử thay đổi dưới sự quản lý của repository, có thể ghi chép lại lịch sử thay đổi của thư mục và file trong thư mục đó.

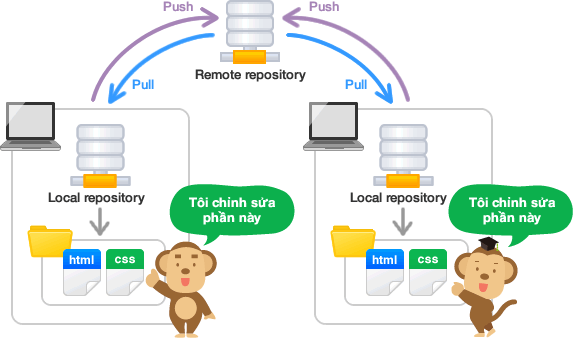


**3. Remote repository và local repository**

Đầu tiên, repository của Git được phân thành 2 loại là remote repository và local repository.

* Remote repository: Là repository để chia sẻ giữa nhiều người và bố trí trên server chuyên dụng.
* Local repository: Là repository bố trí trên máy của bản thân mình, dành cho một người dùng sử dụng.

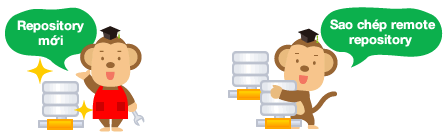
Do repository phân thành 2 loại là local và remote nên với những công việc bình thường thì có thể sử dụng local repository và thực hiện trên toàn bộ máy sẵn có. Khi muốn công khai nội dung công việc mà bản thân đã làm trên local repository, thì sẽ upload lên remote repository rồi công khai. Thêm nữa, thông qua remote repository cũng có thể lấy về nội dung công việc của người khác.



**4. Tạo repository**

Có 2 cách tạo local repository mà mình có sẵn.

Cách thứ nhất là tạo repository hoàn toàn mới, cách thứ hai là sao chép remote repository rồi tạo.

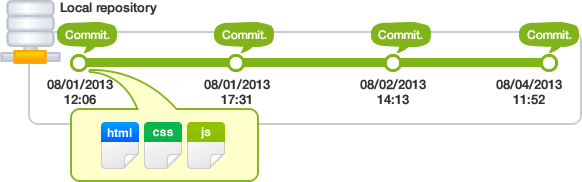


**5. Commit sẽ ghi lại thay đổi**

Để ghi lại việc thêm/ thay đổi file hay thư mục vào repository thì sẽ thực hiện thao tác gọi là Commit.

Khi thực hiện commit, trong repository sẽ tạo ra commit (hoặc revision) đã ghi lại sự khác biệt từ trạng thái đã commit lần trước đến trạng thái hiện tại.

Commit này đang được chứa tại repository trong trạng thái đã nối tiếp với nhau theo thứ tự thời gian như trong hình dưới đây. Bằng việc lần theo commit này từ trạng thái mới nhất thì có thể hiểu được lịch sử thay đổi trong quá khứ hoặc nội dung thay đổi đó.



Các commit này, được đặt tên bởi 40 ký tự alphabet không trùng nhau đã được tính toán từ thông tin commit. Bằng việc chỉ định tên này, có thể chỉ định commit từ trong repository.

Những thay đổi mang ý nghĩa khác nhau chẳng hạn như thêm chức năng hay sửa lỗi thì hãy cố gắng chia ra rồi commit. Để sau này khi xem lịch sử và tìm kiếm một nội dung thay đổi định sẵn sẽ dễ dàng hơn.

Khi thực hiện commit có yêu cầu nhập giải thích commit (commit message). Vì commit message là bắt buộc nên nếu để trống mà thực hiện thì commit sẽ thất bại.

Vì phần giải thích đó sẽ trở thành thông tin quan trọng khi bản thân xem lại lịch sử sau đó hay trường hợp tìm kiếm nội dung thay đổi mà người khác commit, nên hãy chú ý để viết giải thích sao cho dễ dàng hiểu được nội dung thay đổi. Viết giải thích theo hình thức tiêu chuẩn trên Git như sau:

Dòng thứ 1: Tóm tắt nội dung thay đổi trong commit

Dòng thứ 2: Dòng trống

Dòng thứ 3 trở đi: Lý do đã thay đổi

**II. Giới thiệu về Git và Github**

**1. Git là gì?**

Git là một hệ thống kiểm soát phiên bản phổ biến. Nó được tạo ra bởi Linus Torvalds vào năm 2005 và được duy trì bởi Junio ​​Hamano kể từ đó.

Nó dùng để:

* Theo dõi thay đổi mã
* Theo dõi ai đã thực hiện thay đổi
* cộng tác mã hóa

**2. Git làm gì?**

* Quản lý dự án với **Repositories (Kho lưu trữ)**
* **Clone (Sao chép)** một dự án để làm việc trên một bản sao cục bộ
* Kiểm soát và theo dõi các thay đổi với **Staging (Dàn dựng)** và **Committing (Cam kết)**
* **Branch (Chi nhánh)** và **Merge (Hợp nhất)** để cho phép làm việc trên các phần và phiên bản khác nhau của dự án
* **Pull (Kéo)** phiên bản mới nhất của dự án vào một bản sao cục bộ
* **Push (Đẩy)** các bản cập nhật cục bộ vào dự án chính

**3. Làm việc với Git**

* Khởi tạo Git trên một thư mục, biến nó thành **Repositories (Kho lưu trữ)**
* Git hiện tạo một thư mục ẩn để theo dõi các thay đổi trong thư mục đó
* Khi một tệp bị thay đổi, thêm hoặc xóa, nó được coi là **modified (đã sửa đổi)**
* Bạn chọn các file sửa đổi mà bạn muốn **Stage (Dàn dựng)**
* Các tệp **Staged (theo giai đoạn)** được **Committed (Cam kết)** , điều này sẽ nhắc Git lưu trữ ảnh chụp **permanent (nhanh vĩnh viễn)** của các tệp
* Git cho phép bạn xem toàn bộ lịch sử của mọi cam kết.
* Bạn có thể hoàn nguyên về bất kỳ cam kết nào trước đó.
* Git không lưu trữ một bản sao riêng của từng tệp trong mỗi lần xác nhận, nhưng sẽ theo dõi các thay đổi được thực hiện trong mỗi lần xác nhận!

**4. GitHub là gì?**

* Git không giống như GitHub.
* GitHub tạo ra các công cụ sử dụng Git.
* GitHub là kho lưu trữ mã nguồn lớn nhất thế giới và đã thuộc sở hữu của Microsoft từ năm 2018.
* Trong hướng dẫn này, chúng tôi sẽ tập trung vào việc sử dụng Git với GitHub.

**5. Tạo thư mục Git**

Mkdir myproject : mkdir làm cho 1 thư mục mới

Cd myproject : cd thay đổi thư mục làm việc hiện tại

**6. Khởi tạo Git**

Git init

Initialized empty Git repository in /Users/user/myproject/.git/

=> Tạo thành công kho lưu trữ đầu tiên.

Sau đây là các lệnh thường dùng khi sử dụng Git.

# Di chuyển tới một thư mục

Di chuyển tới một thư mục bất kì.

|  |  |
| --- | --- |
| 1 | cd duong\_dan\_toi\_thu\_muc |

Giả sử bạn đang ở thư mục D:/freetuts, lúc này bạn muốn di chuyển tới folder git trong thư mục freetuts thì làm như sau:

cd git

Code đổi ổ đĩa.

cd /TenODia

VD: cd /D

Lùi 1 cấp so với folder hiện tại.

|  |  |
| --- | --- |
| 1 | cd .. |

Để đến thư mục của User trên Windows thì ta dùng cú pháp sau, đây là thư mục nằm trong ổ C:\Users\TenUser.

cd ~

# Xem danh sách file và folder

Để xem danh sách các file và folder nằm trong thư mục thì ta dùng lệnh sau.

|  |  |
| --- | --- |
| 1 | ls DuongDanToiFolder |

Nếu bạn không truyền đường dẫn tới folder thì nó sẽ lấy folder hiện tại.

# Tạo mới folder

Để tạo mới folder thì bạn sử dụng cú pháp sau.

mkdir DuongDanToiFolder

Nếu bạn chỉ nhập tên folder thì hệ thống sẽ tạo ngay tại thư mục đang đứng.

# Xóa folder

Để xóa folder thì bạn sử dụng cú pháp sau.

|  |  |
| --- | --- |
| 1 | rmdir DuongDanToiFolder |

Nếu bạn chỉ nhập tên folder thì hệ thống sẽ xóa folder nằm trong thư mục đang đứng.

# Xem đường dẫn thư mục hiện tại

Để xem thông tin đường dẫn tuyệt đối của thư mục hiện tại thì bạn dùng lệnh sau.

|  |  |
| --- | --- |
| 1 | pwd |

**III. Tệp Git mới**

**1. Git thêm tệp mới**

Bạn vừa tạo repo Git cục bộ đầu tiên của mình. Nhưng nó trống rỗng.

Vì vậy, hãy thêm một số tệp hoặc tạo một tệp mới bằng trình soạn thảo văn bản yêu thích của bạn. Sau đó lưu hoặc di chuyển nó vào thư mục bạn vừa tạo.

Nếu bạn muốn tìm hiểu cách tạo tệp mới bằng trình soạn thảo văn bản, bạn có thể truy cập hướng dẫn HTML của chúng tôi: Trình chỉnh  
[sửa HTML](https://www.w3schools.com/html/html_editors.asp)

Trong ví dụ này, tôi sẽ sử dụng một tệp HTML đơn giản như sau:

Ví dụ

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<p>This is the first file in my new Git Repo.</p>  
  
</body>  
</html>

- Đưa file index.html vừa tạo vào myproject /user/myproject/.git/

- Gõ lệnh ls trong MINGW64 của git

ls

index.html

ls sẽ **liệt kê** các tập tin trong thư mục. Chúng ta có thể thấy điều đó index.html ở đó.

- Sau đó, kiểm tra Git status và xem liệu nó có phải là một phần của kho lưu trữ không:

git status

On branch master

No commits yet

Untracked files:

(use "git add ..." to include in what will be committed)     index.html nothing added to commit but untracked files present (use "git add" to track)

=> Bây giờ Git đã **biết** về tệp, nhưng chưa **thêm** nó vào kho lưu trữ.

Các tệp trong thư mục kho lưu trữ Git của bạn có thể ở một trong 2 trạng thái:

* Đã theo dõi - các tệp mà Git biết và được thêm vào kho lưu trữ
* Không bị theo dõi - các tệp nằm trong thư mục làm việc của bạn nhưng chưa được thêm vào kho lưu trữ

 Khi bạn thêm tệp lần đầu tiên vào kho lưu trữ trống, tất cả chúng đều không bị theo dõi. Để Git theo dõi chúng, bạn cần sắp xếp chúng hoặc thêm chúng vào môi trường dàn dựng.

Chúng tôi sẽ đề cập đến môi trường dàn dựng trong chương tiếp theo.

**IV. Môi trường dàn dựng Git**

**1. Môi trường dàn dựng Git**

Một trong những chức năng cốt lõi của Git là các khái niệm về Môi trường dàn dựng và Cam kết.

Khi bạn đang làm việc, bạn có thể thêm, chỉnh sửa và xóa tệp. Nhưng bất cứ khi nào bạn đạt được một mốc quan trọng hoặc hoàn thành một phần công việc, bạn nên thêm các tệp vào Môi trường chạy thử.

**Các tệp theo giai đoạn** là các tệp sẵn sàng được **cam kết** với kho lưu trữ mà bạn đang làm việc. Bạn sẽ tìm hiểu thêm về commit thời gian ngắn.

Hiện tại, chúng tôi đã hoàn thành công việc với index.html. Vì vậy, chúng tôi có thể thêm nó vào Môi trường dàn dựng:

git add index.html

Hãy kiểm tra trạng thái :

git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached ..." to unstage)     new file: index.html

=> Bây giờ tệp đã được thêm và Môi trường tổ chức

**2. Git thêm nhiều hơn 1 tệp**

Bạn cũng có thể tạo nhiều tệp cùng một lúc. Hãy thêm 2 tệp nữa vào thư mục làm việc. Sử dụng lại trình soạn thảo văn bản.

Tệp README.mdmô tả kho lưu trữ (được khuyến nghị cho tất cả các kho lưu trữ):

# hello-world  
Hello World repository for Git tutorial  
This is an example repository for the Git tutoial on https://www.w3schools.com  
  
This repository is built step by step in the tutorial.

bluestyle.css

body {  
background-color: lightblue;  
}  
  
h1 {  
color: navy;  
margin-left: 20px;  
}

Bây giờ hãy thêm tất cả các tệp trong thư mục hiện tại vào Môi trường tổ chức:

git add --all

Sử dụng –all thay vì tên tệp riêng lẻ sẽ stage thay đổi tất cả các tệp (mới, đã sửa đổi và đã xóa).

git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached ..." to unstage) new file: README.md new file: bluestyle.css new file: index.html

**Note:** The shorthand command for git add --all is git add -A

**V. Cam kết Git (Commit)**

**1. Cam kết Git**

Vì chúng tôi đã hoàn thành công việc của mình, chúng tôi đã sẵn sàng chuyển từ stage sang commit repo của mình.

Thêm các cam kết theo dõi tiến độ và thay đổi của chúng tôi khi chúng tôi làm việc. Git xem xét từng commit điểm thay đổi hoặc "điểm lưu". Đó là một điểm trong dự án mà bạn có thể quay lại nếu tìm thấy lỗi hoặc muốn thực hiện thay đổi.

Khi chúng tôi commit, chúng tôi nên **luôn luôn** bao gồm một **tin nhắn** .

Bằng cách thêm các thông báo rõ ràng vào mỗi mục commit, bạn (và những người khác) sẽ dễ dàng biết được điều gì đã thay đổi và khi nào.

git commit -m "First release of Hello World!"

[master (root-commit) 221ec6e] First release of Hello World!

3 files changed, 26 insertions(+)

create mode 100644 README.md

create mode 100644 bluestyle.css

create mode 100644 index.html

Lệnh commit thực hiện một cam kết và thêm một thông báo.-m "message"

Môi trường dàn dựng đã được cam kết với repo của chúng tôi, với thông báo:  
"Bản phát hành đầu tiên của Hello World!"

**2. Git Commit without Stage**

Đôi khi, khi bạn thực hiện những thay đổi nhỏ, việc sử dụng môi trường dàn dựng có vẻ như lãng phí thời gian. Có thể thực hiện các thay đổi trực tiếp, bỏ qua môi trường dàn dựng. Tùy chọn này -asẽ tự động tạo giai đoạn cho mọi tệp đã thay đổi, đã được theo dõi.

Hãy thêm một bản cập nhật nhỏ vào index.html:

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<p>This is the first file in my new Git Repo.</p>  
<p>A new line in our file!</p>  
  
</body>  
</html>

Và kiểm tra trạng thái của kho lưu trữ của chúng tôi. Nhưng lần này, chúng ta sẽ sử dụng tùy chọn --short để xem các thay đổi một cách cô đọng hơn:

git status --short

M index.html

**Lưu ý:** Cờ trạng thái ngắn là:

* ?? - Tập tin không bị theo dõi
* A - Tệp được thêm vào giai đoạn
* M - Tệp đã sửa đổi
* D - Các tệp đã xóa

Chúng tôi thấy tệp mà chúng tôi mong đợi đã được sửa đổi. Vì vậy, hãy cam kết trực tiếp:

git commit -a -m "Updated index.html with a new line"

[master 09f4acd] Updated index.html with a new line

1 file changed, 1 insertion(+)

**Cảnh báo:** Việc bỏ qua Môi trường dàn dựng thường không được khuyến nghị.

Bỏ qua bước giai đoạn đôi khi có thể khiến bạn đưa vào những thay đổi không mong muốn.

**3. Nhật ký cam kết Git**

Để xem lịch sử của các lần xác nhận cho một kho lưu trữ, bạn có thể sử dụng lệnh log:

git log

commit 09f4acd3f8836b7f6fc44ad9e012f82faf861803 (HEAD -> master)

Author: w3schools-test Date: Fri Mar 26 09:35:54 2021 +0100 Updated index.html with a new line commit 221ec6e10aeedbfd02b85264087cd9adc18e4b26 Author: w3schools-test Date: Fri Mar 26 09:13:07 2021 +0100 First release of Hello World!

**VI. Nhánh Git**

**1. Làm việc với Git Branch**

Trong Git, a branchlà phiên bản mới/riêng biệt của kho lưu trữ chính.

Giả sử bạn có một dự án lớn và bạn cần cập nhật thiết kế trên đó.

Nó sẽ hoạt động như thế nào nếu không có và có Git:

Không có Git:

* Tạo bản sao của tất cả các tệp có liên quan để tránh ảnh hưởng đến phiên bản trực tiếp
* Bắt đầu làm việc với thiết kế và thấy mã đó phụ thuộc vào mã trong các tệp khác, điều đó cũng cần phải thay đổi!
* Tạo bản sao của các tệp phụ thuộc. Đảm bảo rằng mọi tệp phụ thuộc đều tham chiếu đúng tên tệp
* KHẨN CẤP! Có một lỗi không liên quan ở đâu đó khác trong dự án cần được khắc phục càng sớm càng tốt!
* Lưu tất cả các tệp của bạn, ghi chú tên của các bản sao bạn đang làm việc
* Xử lý lỗi không liên quan và cập nhật mã để sửa lỗi
* Quay lại thiết kế và hoàn thành công việc ở đó
* Sao chép mã hoặc đổi tên tệp để thiết kế cập nhật có trên phiên bản trực tiếp
* (2 tuần sau, bạn nhận ra lỗi không liên quan không được sửa trong bản thiết kế mới do bạn đã sao chép các tệp trước khi sửa)

Với Git:

* Với một nhánh mới có tên là new-design, hãy chỉnh sửa mã trực tiếp mà không ảnh hưởng đến nhánh chính
* KHẨN CẤP! Có một lỗi không liên quan ở đâu đó khác trong dự án cần được khắc phục càng sớm càng tốt!
* Tạo một nhánh mới từ dự án chính có tên là sửa lỗi nhỏ
* Sửa lỗi không liên quan và hợp nhất nhánh sửa lỗi nhỏ với nhánh chính
* Bạn quay trở lại nhánh thiết kế mới và hoàn thành công việc ở đó
* Hợp nhất nhánh thiết kế mới với nhánh chính (được cảnh báo về phần sửa lỗi nhỏ mà bạn đang thiếu)

Các nhánh cho phép bạn làm việc trên các phần khác nhau của dự án mà không ảnh hưởng đến nhánh chính.

Khi công việc hoàn thành, một nhánh có thể được hợp nhất với dự án chính.

Bạn thậm chí có thể chuyển đổi giữa các nhánh và làm việc trên các dự án khác nhau mà chúng không can thiệp lẫn nhau.

Phân nhánh trong Git rất nhẹ và nhanh!

**2. Chi nhánh Git mới**

Hãy thêm một số tính năng mới vào trang index.html của chúng tôi.

Chúng tôi đang làm việc trong kho lưu trữ cục bộ của mình và chúng tôi không muốn làm phiền hoặc có thể phá hỏng dự án chính.

Vì vậy, chúng tôi tạo một cái mới branch:

git branch hello-world-images

Bây giờ chúng tôi đã tạo một cái mới branchgọi là " hello-world-images"

Hãy xác nhận rằng chúng tôi đã tạo một cái mới branch:

git branch

hello-world-images

\* master

Chúng ta có thể thấy nhánh mới có tên "hello-world-images", nhưng nhánh \* bên cạnh master xác định rằng chúng ta hiện đang ở trên đó branch.

Checkout là lệnh được sử dụng để kiểm tra tệp branch. Di chuyển chúng tôi **từ** hiện tại branch, **đến** cái được chỉ định ở cuối lệnh:

git checkout hello-world-images

Switched to branch 'hello-world-images'

Bây giờ chúng tôi đã chuyển không gian làm việc hiện tại của mình từ nhánh chính sang nhánh mới branch

Mở trình chỉnh sửa yêu thích của bạn và thực hiện một số thay đổi.

Đối với ví dụ này, chúng tôi đã thêm một hình ảnh (img\_hello\_world.jpg) vào thư mục làm việc và một dòng mã trong tệp index.html:

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<div><img src="img\_hello\_world.jpg" alt="Hello World from Space"  
style="width:100%;max-width:960px"></div>  
<p>This is the first file in my new Git Repo.</p>  
<p>A new line in our file!</p>  
  
</body>  
</html>

Chúng tôi đã thực hiện các thay đổi đối với một tệp và thêm một tệp mới vào thư mục làm việc (cùng thư mục với tệp main branch).

Bây giờ hãy kiểm tra trạng thái của hiện tại branch:

git status

On branch hello-world-images

Changes not staged for commit:

(use "git add ..." to update what will be committed) (use "git restore ..." to discard changes in working directory) modified: index.html Untracked files: (use "git add ..." to include in what will be committed) img\_hello\_world.jpg no changes added to commit (use "git add" and/or "git commit -a")

Vì vậy, hãy đi qua những gì xảy ra ở đây:

* Có những thay đổi đối với index.html của chúng tôi, nhưng tệp không được sắp xếp cho commit
* img\_hello\_world.jpg không phải tracked

Vì vậy, chúng tôi cần thêm cả hai tệp vào Môi trường tổ chức cho việc này branch:

git add –all

Sử dụng –all thay vì tên tệp riêng lẻ sẽ **Giai đoạn** tất cả các tệp đã thay đổi (mới, đã sửa đổi và đã xóa).

Kiểm tra status của branch:

git status

On branch hello-world-images

Changes to be committed:

(use "git restore --staged ..." to unstage)     new file: img\_hello\_world.jpg     modified: index.html

Chúng tôi hài lòng với những thay đổi của mình. Vì vậy, chúng tôi sẽ cam kết chúng với branch:

git commit -m "Added image to Hello World"

[hello-world-images 0312c55] Added image to Hello World

2 files changed, 1 insertion(+)

create mode 100644 img\_hello\_world.jpg

Bây giờ chúng ta có một cái mới branch, khác với cái chính branch.

**Lưu ý:** Sử dụng -btùy chọn trên checkoutsẽ tạo một nhánh mới và di chuyển đến nhánh đó nếu nhánh đó chưa tồn tại

**3. Chuyển đổi giữa các nhánh**

Bây giờ hãy xem làm việc với các nhánh khác nhau nhanh chóng và dễ dàng như thế nào và nó hoạt động tốt như thế nào.

Chúng tôi hiện đang ở trên chi nhánh hello-world-images. Chúng tôi đã thêm một hình ảnh vào nhánh này, vì vậy hãy liệt kê các tệp trong thư mục hiện tại:

ls

README.md bluestyle.css img\_hello\_world.jpg index.html

Chúng tôi có thể thấy tệp mới img\_hello\_world.jpgvà nếu chúng tôi mở tệp html, chúng tôi có thể thấy mã đã bị thay đổi. Tất cả là như nó phải được.

Bây giờ, hãy xem điều gì xảy ra khi chúng ta thay đổi nhánh thành master

git checkout master

Switched to branch 'master'

Hình ảnh mới không phải là một phần của nhánh này. Liệt kê lại các tệp trong thư mục hiện tại:

ls

README.md bluestyle.css index.html

img\_hello\_world.jpg không còn ở đó nữa! Và nếu chúng tôi mở tệp html, chúng tôi có thể thấy mã được hoàn nguyên về mã trước khi thay đổi.

Xem làm việc với các chi nhánh dễ dàng như thế nào? Và làm thế nào điều này cho phép bạn làm việc trên những thứ khác nhau?

**4. Chi nhánh khẩn cấp**

Bây giờ hãy tưởng tượng rằng chúng ta chưa hoàn thành với hello-world-images, nhưng chúng ta cần sửa lỗi trên master.

Tôi không muốn gây rối trực tiếp với master và tôi không muốn gây rối với hình ảnh img\_hello\_world.jpg, vì nó vẫn chưa được thực hiện.

Vì vậy, chúng tôi tạo một chi nhánh mới để đối phó với trường hợp khẩn cấp:

git checkout -b emergency-fix

Switched to a new branch 'emergency-fix'

Bây giờ chúng tôi đã tạo một nhánh mới từ nhánh chính và thay đổi thành nhánh đó. Chúng tôi có thể sửa lỗi một cách an toàn mà không làm ảnh hưởng đến các nhánh khác.

Hãy sửa lỗi tưởng tượng của chúng tôi:

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<p>This is the first file in my new Git Repo.</p>  
<p>This line is here to show how merging works.</p>  
  
</body>  
</html>

Chúng tôi đã thực hiện các thay đổi trong tệp này và chúng tôi cần chuyển những thay đổi đó sang nhánh chính.

Kiểm tra trạng thái:

git status

On branch emergency-fix

Changes not staged for commit:

(use "git add ..." to update what will be committed) (use "git restore ..." to discard changes in working directory) modified: index.html no changes added to commit (use "git add" and/or "git commit -a")

giai đoạn tệp và cam kết:

git add index.html

git commit -m "updated index.html with emergency fix"

[emergency-fix dfa79db] updated index.html with emergency fix

1 file changed, 1 insertion(+), 1 deletion(-)

Bây giờ chúng tôi đã có bản sửa lỗi sẵn sàng cho bản chính và chúng tôi cần hợp nhất hai nhánh.

**VII. Hợp nhất chi nhánh Git**

**1. Hợp nhất các nhánh**

Chúng tôi đã sẵn sàng sửa lỗi khẩn cấp, vì vậy hãy hợp nhất nhánh chính và nhánh sửa lỗi khẩn cấp.

Đầu tiên, chúng ta cần đổi sang nhánh chính:

git checkout master

Switched to branch 'master'

Bây giờ chúng tôi hợp nhất nhánh hiện tại (chính) với bản sửa lỗi khẩn cấp:

git merge emergency-fix

Updating 09f4acd..dfa79db

Fast-forward

index.html | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

Vì nhánh sửa lỗi khẩn cấp đến trực tiếp từ master và không có thay đổi nào khác được thực hiện đối với master khi chúng tôi đang làm việc, nên Git coi đây là phần tiếp theo của master. Vì vậy, nó có thể "Tua đi nhanh" (Fast-forward), chỉ trỏ cả bản sửa lỗi chính và bản sửa lỗi khẩn cấp cho cùng một cam kết.

Vì bản sửa lỗi chính và bản sửa lỗi khẩn cấp về cơ bản là giống nhau, chúng ta có thể xóa bản sửa lỗi khẩn cấp vì nó không còn cần thiết nữa:

git branch -d emergency-fix

Deleted branch emergency-fix (was dfa79db).

**2. Hợp nhất xung đột**

Bây giờ chúng ta có thể chuyển sang hello-world-images và tiếp tục làm việc. Thêm một tệp hình ảnh khác (img\_hello\_git.jpg) và thay đổi index.html, để nó hiển thị:

git checkout hello-world-images

Switched to branch 'hello-world-images'

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<div><img src="img\_hello\_world.jpg" alt="Hello World from Space" style="width:100%;max-width:960px"></div>  
<p>This is the first file in my new Git Repo.</p>  
<p>A new line in our file!</p>  
<div><img src="img\_hello\_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>  
  
</body>  
</html>

Bây giờ, chúng tôi đã hoàn thành công việc của mình ở đây và có thể tạo giai đoạn và cam kết cho nhánh này:

git add --all

git commit -m "added new image"

[hello-world-images 1f1584e] added new image

2 files changed, 1 insertion(+)

create mode 100644 img\_hello\_git.jpg

Chúng tôi thấy rằng index.html đã được thay đổi trong cả hai nhánh. Bây giờ chúng tôi đã sẵn sàng để hợp nhất các hình ảnh hello-world thành chủ. Nhưng điều gì sẽ xảy ra với những thay đổi mà chúng tôi đã thực hiện gần đây trong bản gốc?

git checkout master

git merge hello-world-images

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

Việc hợp nhất không thành công do có xung đột giữa các phiên bản cho index.html. Hãy để chúng tôi kiểm tra trạng thái:

git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Changes to be committed:

new file: img\_hello\_git.jpg

new file: img\_hello\_world.jpg

Unmerged paths:

(use "git add ..." to mark resolution) both modified: index.html

Điều này xác nhận rằng có xung đột trong index.html, nhưng các tệp hình ảnh đã sẵn sàng và được sắp xếp để cam kết.

Vì vậy, chúng ta cần khắc phục xung đột đó. Mở tệp trong trình chỉnh sửa của chúng tôi:

<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

<link rel="stylesheet" href="bluestyle.css">

</head>

<body>

<h1>Hello world!</h1>

<div><img src="img\_hello\_world.jpg" alt="Hello World from Space" style="width:100%;max-width:960px"></div>

<p>This is the first file in my new Git Repo.</p>

<<<<<<< HEAD

<p>This line is here to show how merging works.</p>

=======

<p>A new line in our file!</p>

<div><img src="img\_hello\_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>

>>>>>>> hello-world-images

</body>

</html>

Chúng ta có thể thấy sự khác biệt giữa các phiên bản và chỉnh sửa nó theo ý muốn:

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<div><img src="img\_hello\_world.jpg" alt="Hello World from Space" style="width:100%;max-width:960px"></div>  
<p>This is the first file in my new Git Repo.</p>  
<p>This line is here to show how merging works.</p>  
<div><img src="img\_hello\_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>  
  
</body>  
</html>

Bây giờ chúng ta có thể tạo giai đoạn index.html và kiểm tra trạng thái:

git add index.html

git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

new file: img\_hello\_git.jpg

new file: img\_hello\_world.jpg

modified: index.html

Xung đột đã được khắc phục và chúng tôi có thể sử dụng cam kết để kết thúc quá trình hợp nhất:

git commit -m "merged with hello-world-images after fixing conflicts"

[master e0b6038] merged with hello-world-images after fixing conflicts

Và xóa nhánh hello-world-images:

git branch -d hello-world-images

Deleted branch hello-world-images (was 1f1584e).

Bây giờ bạn đã hiểu rõ hơn về cách thức hoạt động của các nhánh và sự hợp nhất. Đã đến lúc bắt đầu làm việc với một kho lưu trữ từ xa!

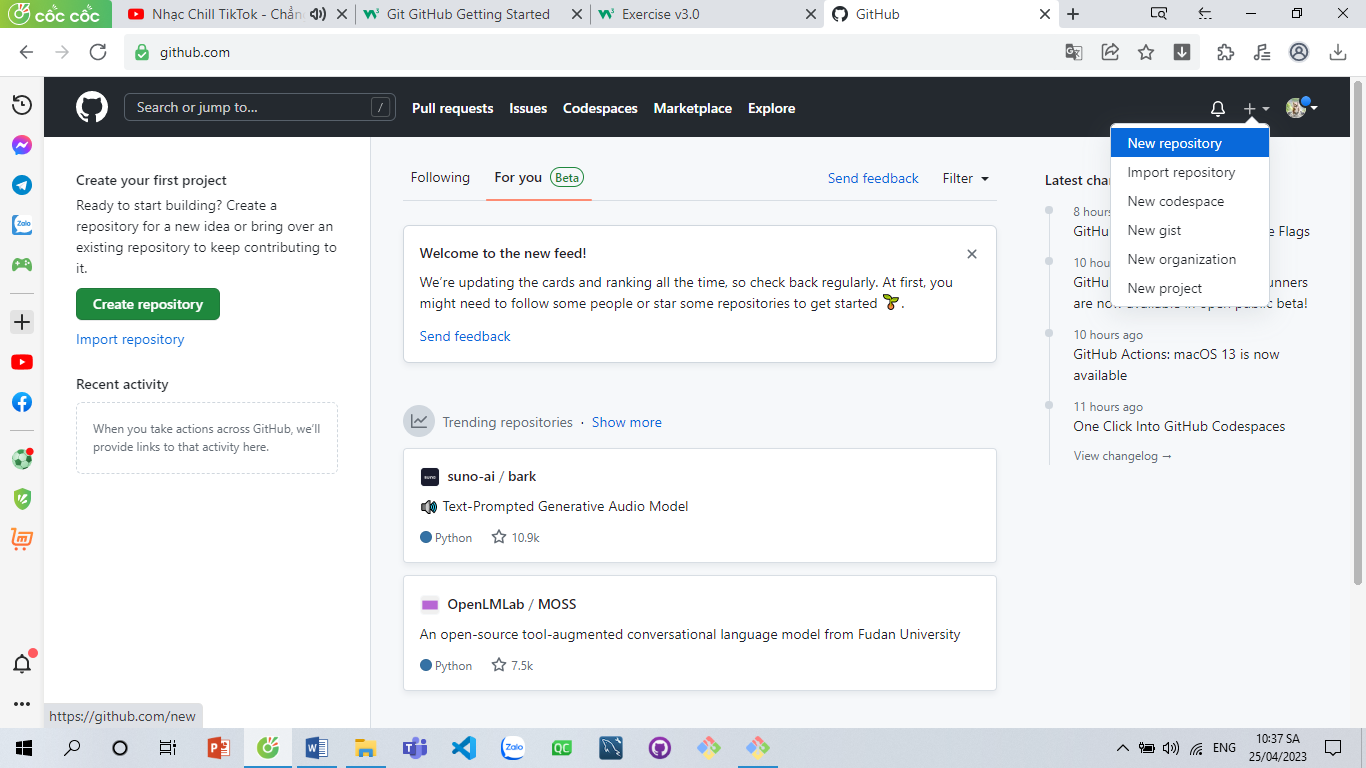
**VIII. Git GitHub**

**1. Tạo tài khoản GitHub**

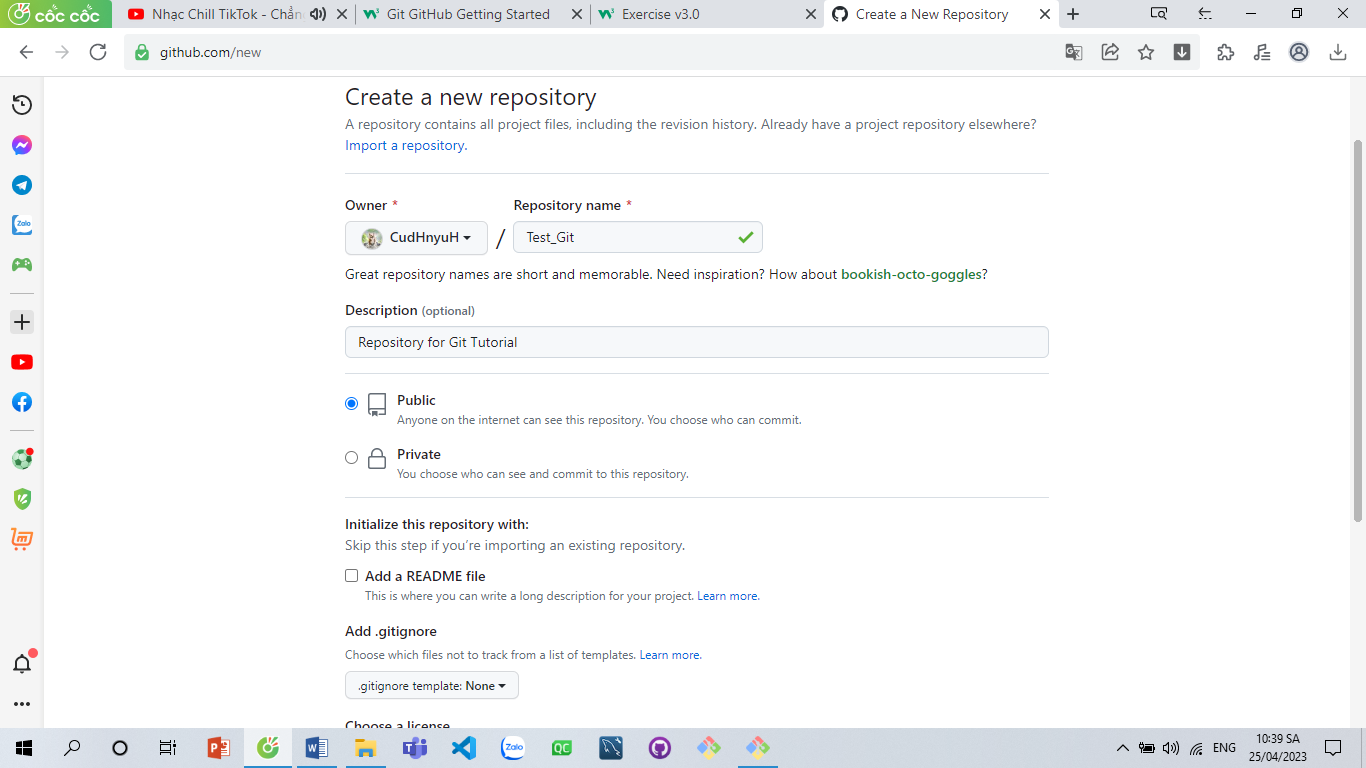
**Lưu ý:** Hãy nhớ sử dụng cùng một địa chỉ email mà bạn đã sử dụng trong cấu hình Git.

**2. Tạo Kho lưu trữ trên GitHub**

Bây giờ bạn đã tạo một tài khoản GitHub, hãy đăng nhập và tạo một Repo mới:



Và điền vào các chi tiết có liên quan:

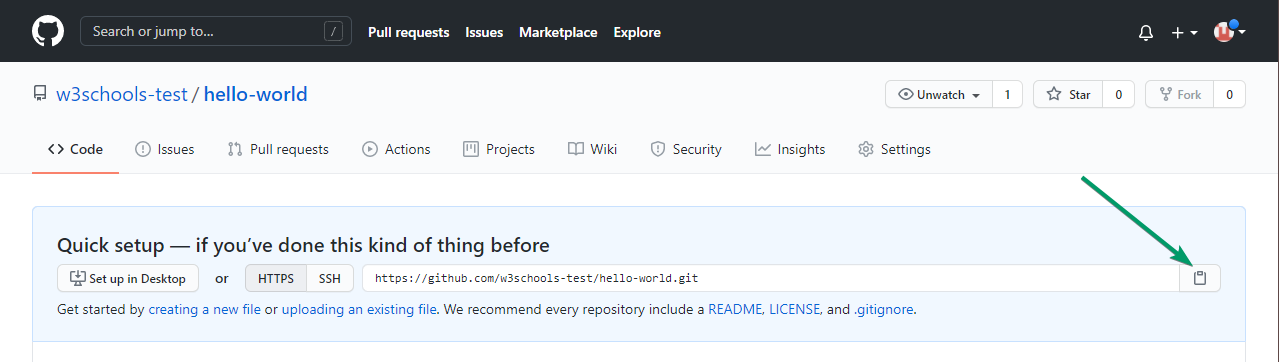


Chúng ta sẽ xem xét các tùy chọn khác nhau và ý nghĩa của chúng sau. Nhưng hiện tại, hãy chọn Công khai (nếu bạn muốn mọi người có thể xem repo) hoặc Riêng tư (nếu bạn muốn chọn ai sẽ có thể xem repo). Dù bằng cách nào, bạn sẽ có thể chọn người có thể **đóng góp** cho repo.

Sau đó nhấp vào "Tạo kho lưu trữ".

**3. Đẩy Repository lên GitHub**

Vì chúng tôi đã thiết lập một repo Git cục bộ, chúng tôi sẽ chuyển push sang GitHub:



Sao chép URL hoặc nhấp vào khay nhớ tạm được đánh dấu trong hình trên.

Bây giờ dán nó vào lệnh sau:

git remote add origin https://github.com/w3schools-test/hello-world.git

git remote add origin URL chỉ định rằng bạn đang thêm một kho lưu trữ từ xa, với , được chỉ định URL, như một origin kho lưu trữ Git cục bộ của bạn.

Bây giờ chúng ta sẽ đẩy nhánh chính của mình đến url gốc và đặt nó làm nhánh từ xa mặc định:

git push --set-upstream origin master

Enumerating objects: 22, done.

Counting objects: 100% (22/22), done.

Delta compression using up to 16 threads

Compressing objects: 100% (22/22), done.

Writing objects: 100% (22/22), 92.96 KiB | 23.24 MiB/s, done.

Total 22 (delta 11), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (11/11), done.

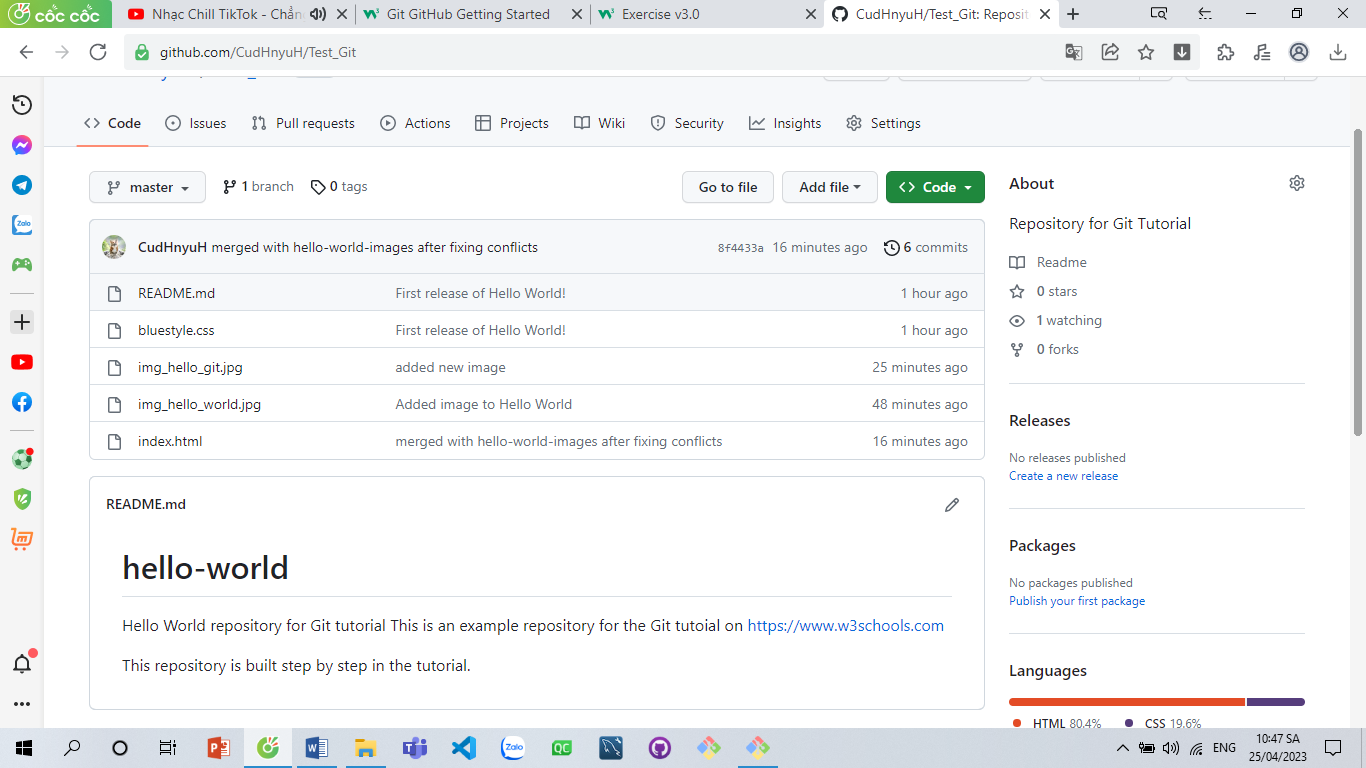
To https://github.com/w3schools-test/hello-world.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

**Lưu ý:** Vì đây là lần đầu tiên bạn kết nối với GitHub nên bạn sẽ nhận được một số loại thông báo xác thực kết nối này.

Bây giờ, hãy quay lại GitHub và xem kho lưu trữ đã được cập nhật:

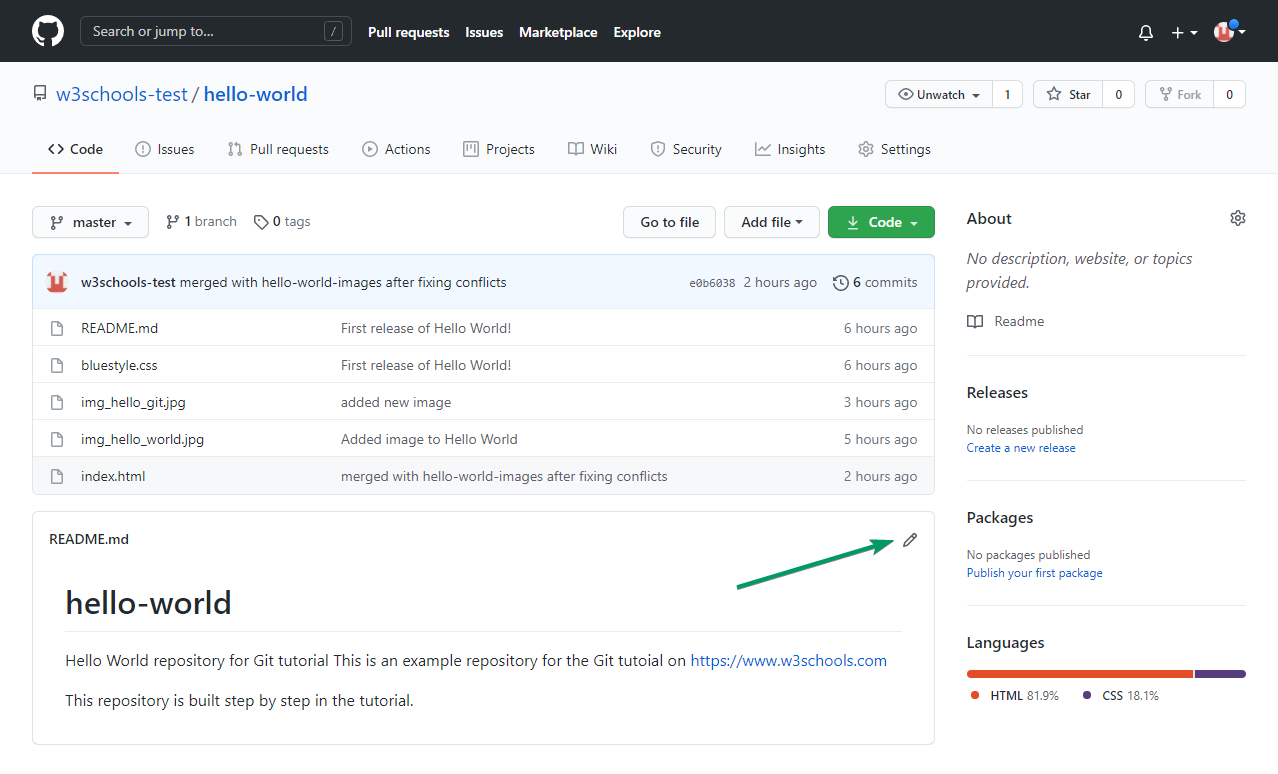


**IX. Chỉnh sửa mã Git GitHub**

**1. Chỉnh sửa mã trong GitHub**

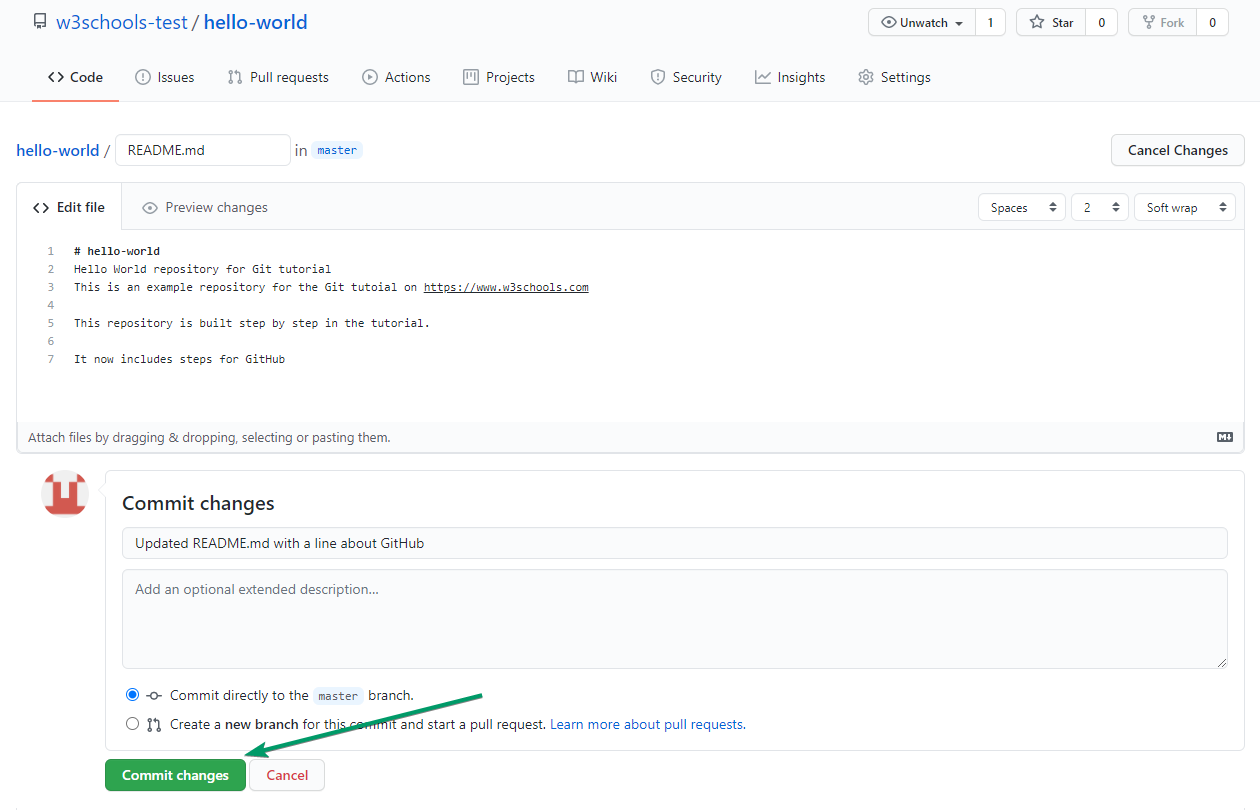
Ngoài việc là nơi lưu trữ nội dung Git, GitHub còn có một trình chỉnh sửa mã rất tốt.

Hãy thử chỉnh sửa tệp README.md trong GitHub. Chỉ cần nhấp vào nút chỉnh sửa:



Thêm một số thay đổi vào mã và sau đó là commit các thay đổi. Hiện tại, chúng tôi sẽ "Cam kết trực tiếp với nhánh chính".

Hãy nhớ thêm một mô tả cho commit:



Đó là cách bạn chỉnh sửa mã trực tiếp trong GitHub!

**2. Git Pull (Kéo) from GitHub**

*a. Pull để cập nhật các thay đổi*

Khi làm việc theo nhóm trong một dự án, điều quan trọng là mọi người luôn cập nhật.

Bất cứ khi nào bạn bắt đầu làm việc với một dự án, bạn sẽ nhận được những thay đổi gần đây nhất đối với bản sao cục bộ của mình.

Với Git, bạn có thể làm điều đó với pull.

Pull là sự kết hợp của 2 lệnh khác nhau:

* fetch
* merge

Chúng ta hãy xem xét kỹ hơn cách thức fetch, merge và pull hoạt động.

*b. Tìm nạp Git*

fetch nhận tất cả lịch sử thay đổi của một nhánh/repo được theo dõi.

Vì vậy, trên Git cục bộ của bạn, hãy cập nhật fetch để xem điều gì đã thay đổi trên GitHub:

git fetch origin

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 733 bytes | 3.00 KiB/s, done.

From https://github.com/w3schools-test/hello-world

e0b6038..d29d69f master -> origin/master

Bây giờ chúng tôi có changes gần đây, chúng tôi có thể kiểm tra status:

git status

On branch master

Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.

(use "git pull" to update your local branch)

nothing to commit, working tree clean

Chúng tôi đứng sau origin/master 1 commit. Đó phải là bản cập nhật README.md, nhưng hãy kiểm tra lại bằng cách xem log:

git log origin/master

commit d29d69ffe2ee9e6df6fa0d313bb0592b50f3b853 (origin/master)

Author: w3schools-test <77673807+w3schools-test@users.noreply.github.com>

Date: Fri Mar 26 14:59:14 2021 +0100

Updated README.md with a line about GitHub

commit e0b6038b1345e50aca8885d8fd322fc0e5765c3b (HEAD -> master)

Merge: dfa79db 1f1584e

Author: w3schools-test Date: Fri Mar 26 12:42:56 2021 +0100 merged with hello-world-images after fixing conflicts ... ...

Điều đó có vẻ như mong đợi, nhưng chúng tôi cũng có thể xác minh bằng cách hiển thị sự khác biệt giữa địa phương của chúng tôi master và origin/master:

git diff origin/master

diff --git a/README.md b/README.md

index 23a0122..a980c39 100644

--- a/README.md

+++ b/README.md

@@ -2,6 +2,4 @@

Hello World repository for Git tutorial

This is an example repository for the Git tutoial on https://www.w3schools.com

-This repository is built step by step in the tutorial.

-

-It now includes steps for GitHub

+This repository is built step by step in the tutorial.

\ No newline at end of file

*c. Hợp nhất Git*

merge kết hợp nhánh hiện tại, với một nhánh được chỉ định.

Chúng tôi đã xác nhận rằng các bản cập nhật như mong đợi và chúng tôi có thể hợp nhất nhánh hiện tại của mình ( master) với origin/master:

git merge origin/master

Updating e0b6038..d29d69f

Fast-forward

README.md | 4 +++-

1 file changed, 3 insertions(+), 1 deletion(-)

Kiểm tra lại status của chúng tôi để xác nhận chúng tôi được cập nhật:

git status

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

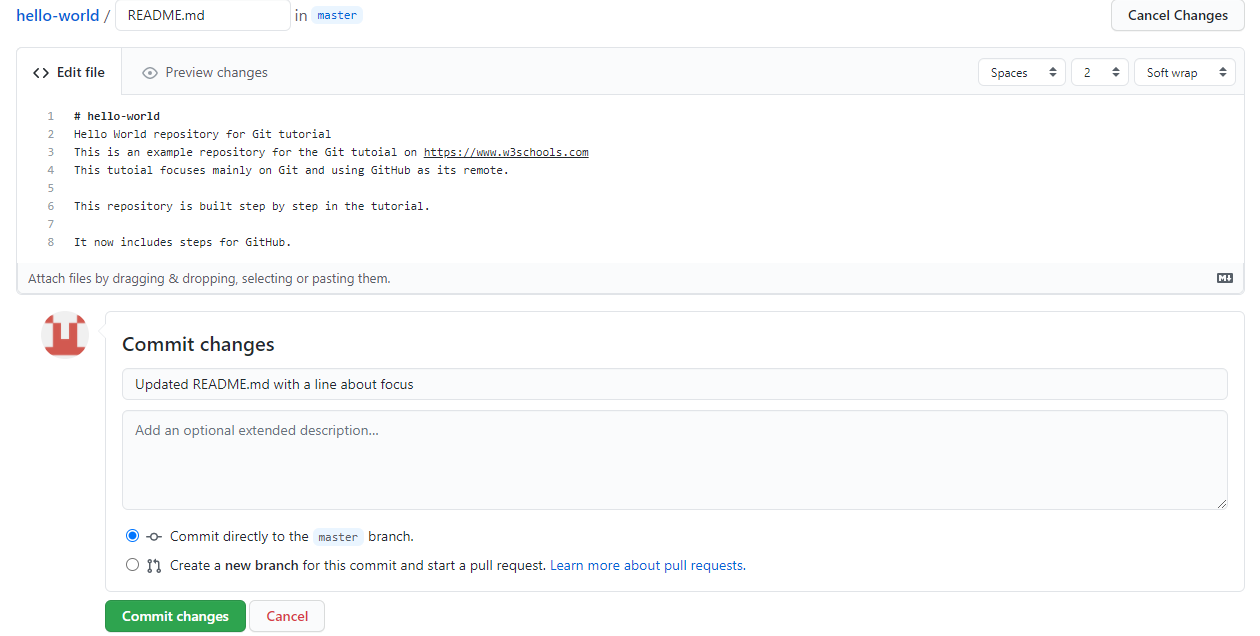
Ở đó! Git địa phương của bạn được cập nhật!

*d. Kéo Git*

Nhưng nếu bạn chỉ muốn cập nhật kho lưu trữ cục bộ của mình mà không cần thực hiện tất cả các bước đó thì sao?

Pull là sự kết hợp của fetch và merge. Nó được sử dụng để kéo tất cả các thay đổi từ kho lưu trữ từ xa vào nhánh mà bạn đang làm việc.

Thực hiện một thay đổi khác đối với tệp Readme.md trên GitHub.



Sử dụng pull để cập nhật Git cục bộ của chúng tôi:

git pull origin

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 794 bytes | 1024 bytes/s, done.

From https://github.com/w3schools-test/hello-world

a7cdd4b..ab6b4ed master -> origin/master

Updating a7cdd4b..ab6b4ed

Fast-forward

README.md | 2 ++

1 file changed, 2 insertions(+)

Đó là cách bạn luôn cập nhật Git cục bộ của mình từ một kho lưu trữ từ xa. Trong chương tiếp theo, chúng ta sẽ xem xét kỹ hơn cách push hoạt động trên GitHub.

**3. Đẩy Git lên GitHub**

*a. Đẩy các thay đổi lên GitHub*

Hãy thử thực hiện một số thay đổi đối với git cục bộ của chúng tôi và đẩy chúng lên GitHub.

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<div><img src="img\_hello\_world.jpg" alt="Hello World from Space" style="width:100%;max-width:640px"></div>  
<p>This is the first file in my new Git Repo.</p>  
<p>This line is here to show how merging works.</p>  
<div><img src="img\_hello\_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>  
  
</body>  
</html>

Cam kết các thay đổi:

git commit -a -m "Updated index.html. Resized image"

[master e7de78f] Updated index.html. Resized image

1 file changed, 1 insertion(+), 1 deletion(-)

Và kiểm tra trạng thái:

git status

On branch master

Your branch is ahead of 'origin/master' by 1 commit.

(use "git push" to publish your local commits)

nothing to commit, working tree clean

Bây giờ, đẩy các thay đổi của chúng tôi đến nguồn gốc từ xa của chúng tôi:

git push origin

Enumerating objects: 9, done.

Counting objects: 100% (8/8), done.

Delta compression using up to 16 threads

Compressing objects: 100% (5/5), done.

Writing objects: 100% (5/5), 578 bytes | 578.00 KiB/s, done.

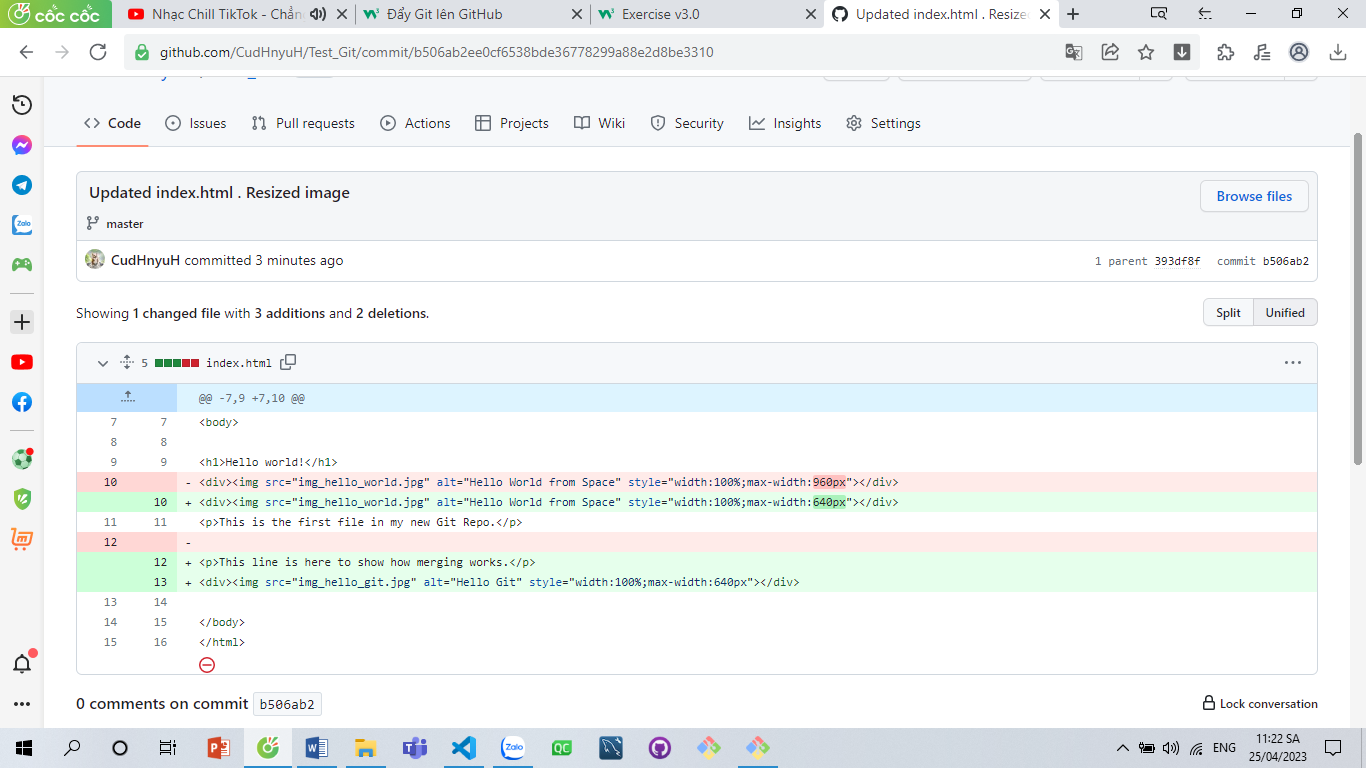
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (3/3), completed with 3 local objects.

To https://github.com/w3schools-test/hello-world.git

5a04b6f..facaeae master -> master

Truy cập GitHub và xác nhận rằng kho lưu trữ có một cam kết mới:

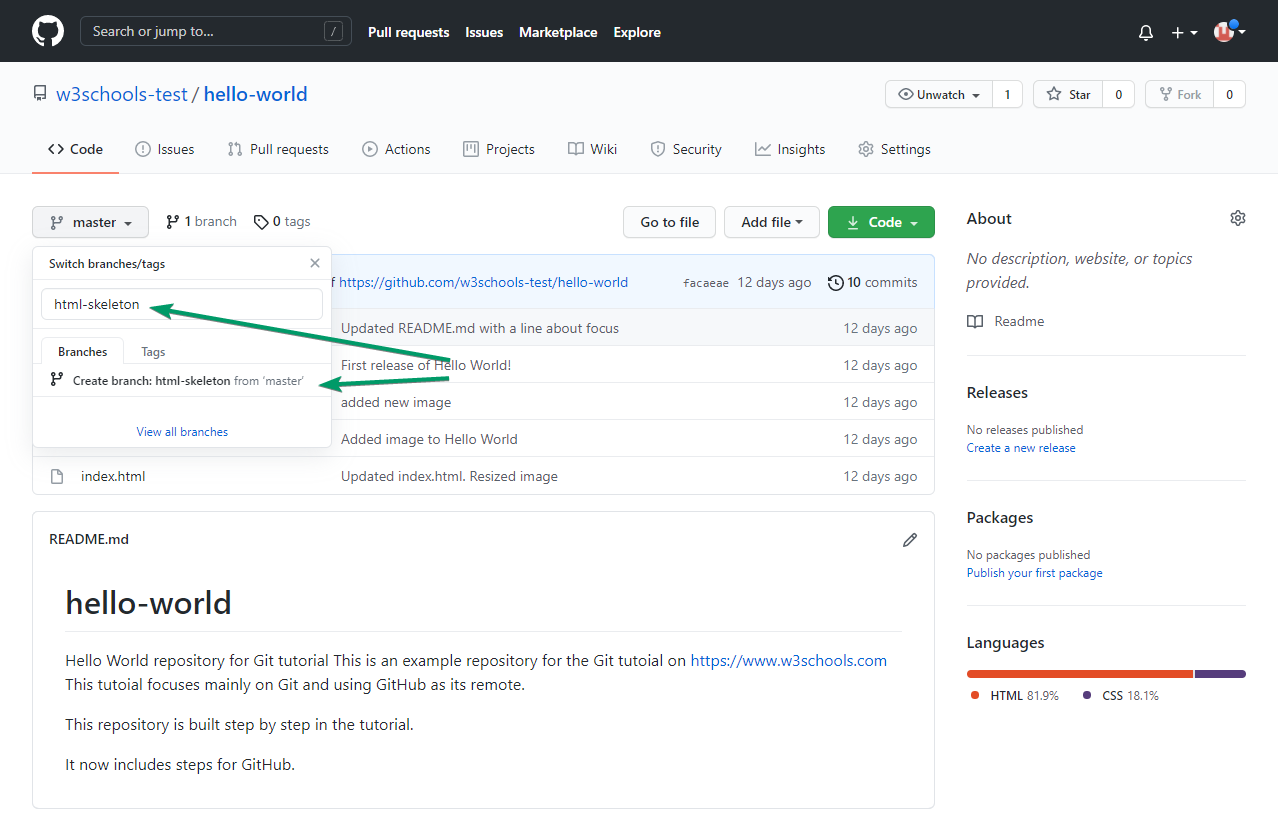


**4. Chi nhánh Git GitHub**

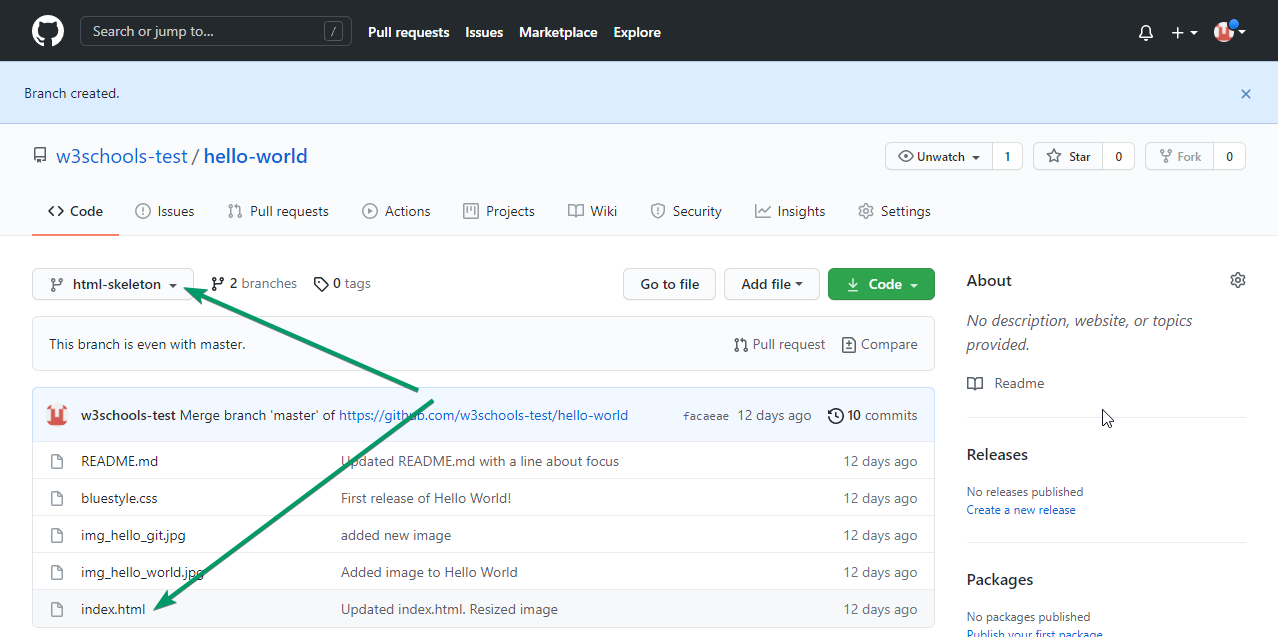
*a. Tạo một chi nhánh mới trên GitHub*

Trên GitHub, truy cập kho lưu trữ của bạn và nhấp vào nút nhánh "master".

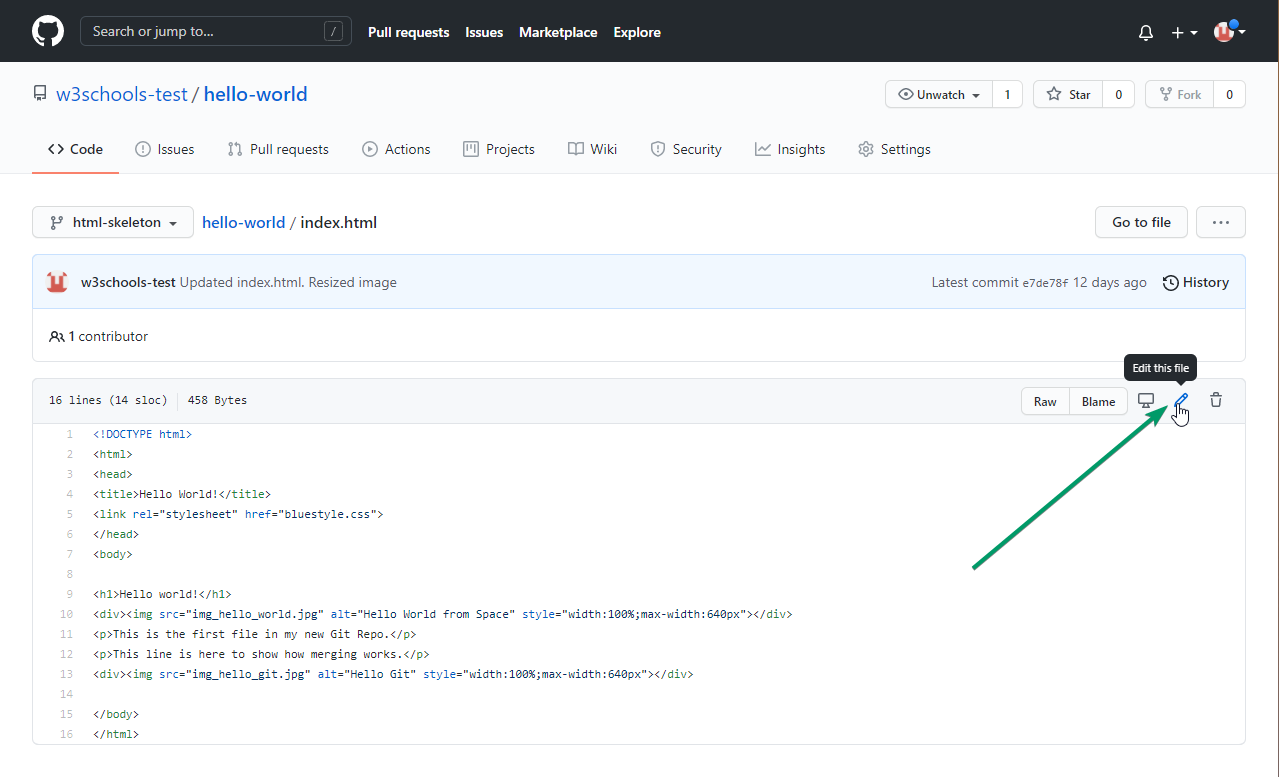
Ở đó bạn có thể tạo một Chi nhánh mới. Nhập tên mô tả và nhấp vào Tạo chi nhánh:



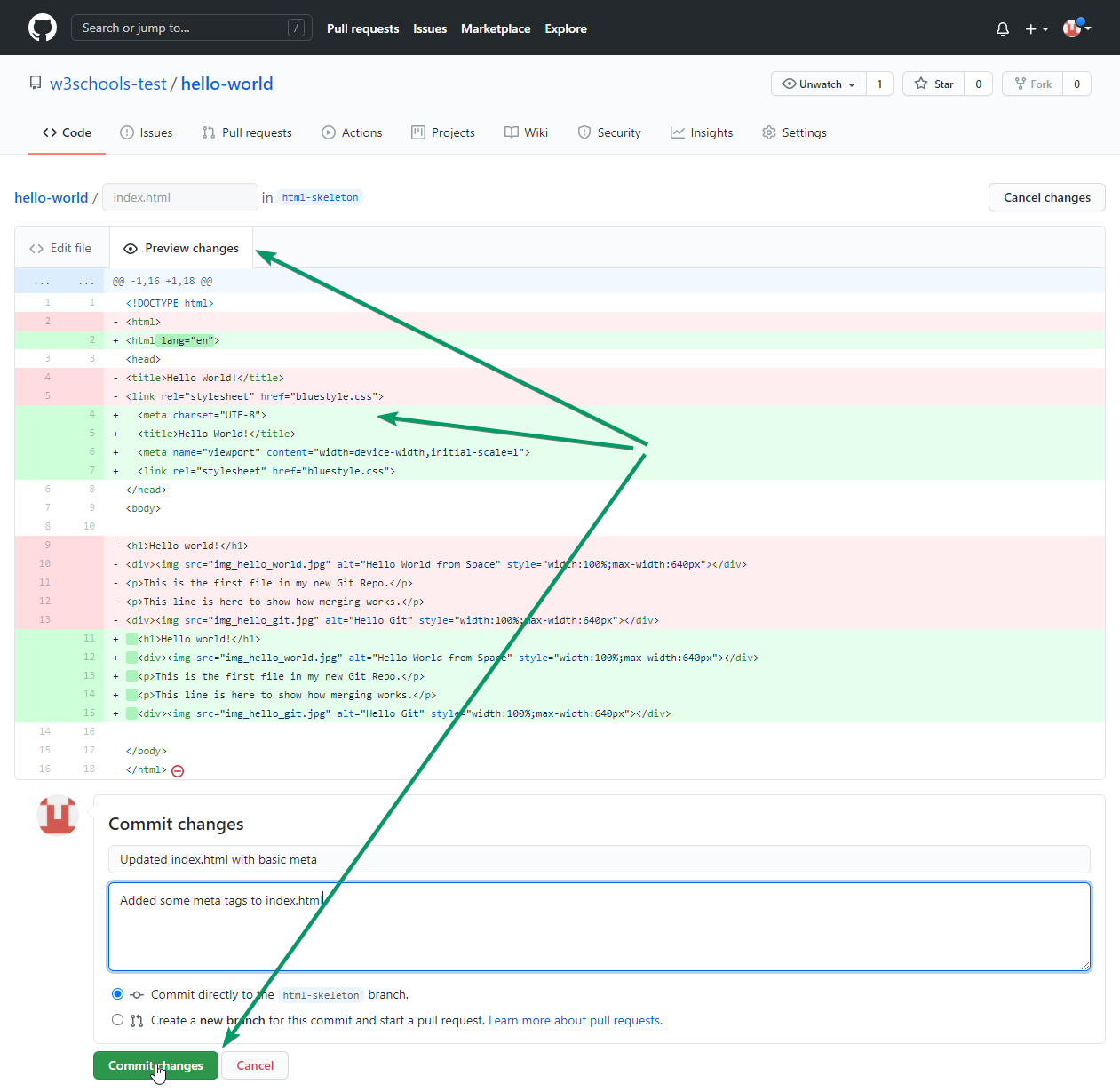
Bây giờ nên branch được tạo và hoạt động. Bạn có thể xác nhận bạn đang làm việc trên nhánh nào bằng cách nhìn vào nút nhánh. Thấy rằng bây giờ nó nói "html-skeleton" thay vì "main"?



Bắt đầu làm việc trên một tệp hiện có trong nhánh này. Nhấp vào tệp " index.html" và bắt đầu chỉnh sửa:



Sau khi chỉnh sửa xong tệp, bạn có thể nhấp vào tab "Xem trước các thay đổi" để xem những thay đổi bạn đã thực hiện được đánh dấu:



Nếu bạn hài lòng với thay đổi, hãy thêm nhận xét giải thích những gì bạn đã làm và nhấp vào Cam kết thay đổi.

Bây giờ bạn đã có một cái mới branch trên GitHub, được cập nhật với một số thay đổi!

**5. Git kéo nhánh từ GitHub**

*a. Kéo 1 chi nhánh từ GitHub*

branch Bây giờ hãy tiếp tục làm việc với Git mới của chúng tôi.

Cho phép pull từ kho lưu trữ GitHub của chúng tôi một lần nữa để mã của chúng tôi được cập nhật:

git pull

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 851 bytes | 9.00 KiB/s, done.

From https://github.com/w3schools-test/hello-world

\* [new branch] html-skeleton -> origin/html-skeleton

Already up to date.

Bây giờ chính của chúng tôi branchđược cập nhật. Và chúng ta có thể thấy rằng có một cái mới branch trên GitHub.

Kiểm tra nhanh status:

git status

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Và xác nhận những chi nhánh chúng tôi có và nơi chúng tôi đang làm việc vào lúc này:

git branch

\* master

Vì vậy, chúng tôi không có cái mới branch trên Git cục bộ của chúng tôi. Nhưng chúng tôi biết nó có sẵn trên GitHub. Vì vậy, chúng tôi có thể sử dụng -a tùy chọn để xem tất cả các nhánh cục bộ và từ xa:

git branch -a

\* master

remotes/origin/html-skeleton

remotes/origin/master

**Lưu ý:** branch -r chỉ dành cho các chi nhánh từ xa.

Chúng tôi thấy rằng nhánh html-skeleton có sẵn từ xa, nhưng không có trên git cục bộ của chúng tôi. Hãy cùng kiểm tra nào:

git checkout html-skeleton

Switched to a new branch 'html-skeleton'

Branch 'html-skeleton' set up to track remote branch 'html-skeleton' from 'origin'.

Và kiểm tra xem tất cả đã cập nhật chưa:

git pull

Already up to date.

Hiện tại chúng tôi có những chi nhánh nào và chúng tôi đang làm việc từ đâu?

git branch

\* html-skeleton

Master

Bây giờ, hãy mở trình chỉnh sửa yêu thích của bạn và xác nhận rằng các thay đổi từ nhánh GitHub đã được chuyển sang.

Đó là cách bạn kéo một nhánh GitHub đến Git cục bộ của mình.

**6. Git đẩy nhánh lên GitHub**

*a. Đẩy 1 chi nhánh lên GitHub*

Hãy thử tạo một nhánh cục bộ mới và đẩy nó lên GitHub.

Bắt đầu bằng cách tạo một nhánh, giống như chúng ta đã làm trước đó:

git checkout -b update-readme

Switched to a new branch 'update-readme'

Và chúng tôi thực hiện một số thay đổi đối với tệp README.md. Chỉ cần thêm một dòng mới.

Vì vậy, bây giờ chúng tôi kiểm tra chi nhánh status hiện tại.

git status

On branch update-readme

Changes not staged for commit:

(use "git add ..." to update what will be committed) (use "git restore ..." to discard changes in working directory) modified: README.md no changes added to commit (use "git add" and/or "git commit -a")

Chúng tôi thấy điều đó README.mdđã được sửa đổi nhưng không được thêm vào Môi trường tổ chức:

git add README.md

Kiểm tra chi nhánh status:

git status

On branch update-readme

Changes to be committed:

(use "git restore --staged ..." to unstage) modified: README.md

Chúng tôi hài lòng với những thay đổi của mình. Vì vậy, chúng tôi sẽ đưa commithọ đến branch:

git commit -m "Updated readme for GitHub Branches"

[update-readme 836e5bf] Updated readme for GitHub Branches

1 file changed, 1 insertion(+)

Giờ đây push, branch từ kho lưu trữ Git cục bộ của chúng tôi, sang GitHub, nơi mọi người có thể thấy các thay đổi:

git push origin update-readme

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 16 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 366 bytes | 366.00 KiB/s, done.

Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

remote:

remote: Create a pull request for 'update-readme' on GitHub by visiting:

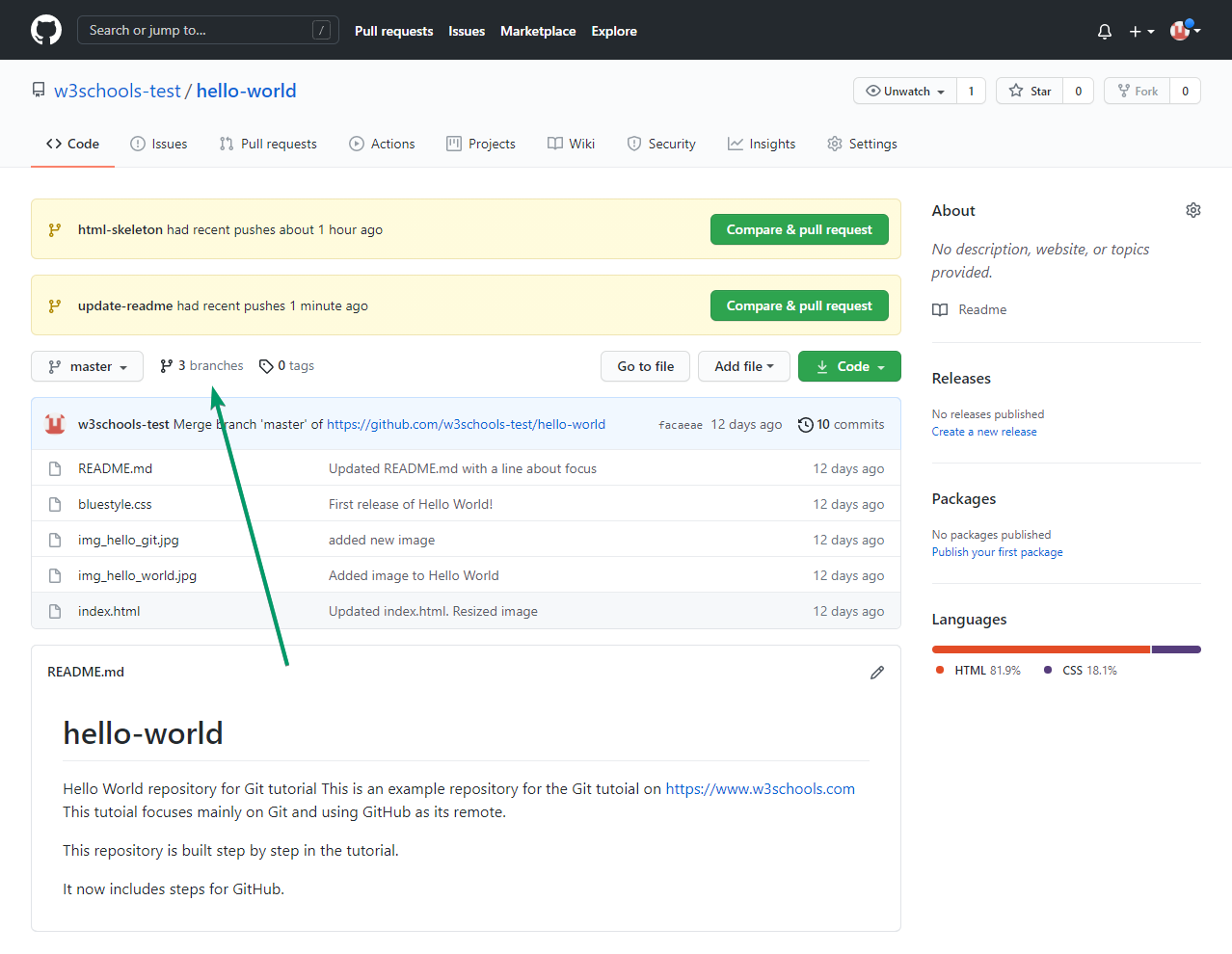
remote: https://github.com/w3schools-test/hello-world/pull/new/update-readme

remote:

To https://github.com/w3schools-test/hello-world.git

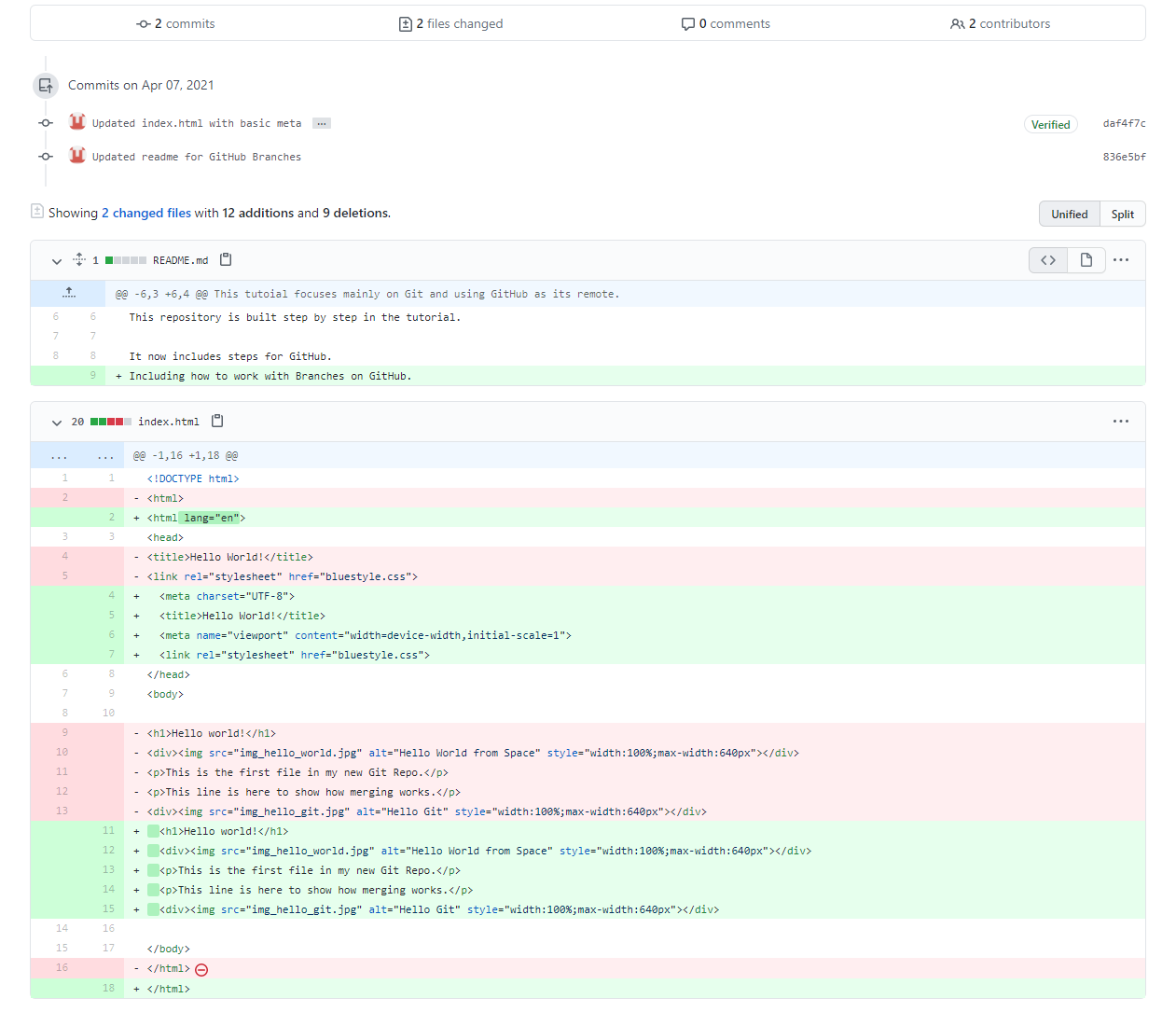
\* [new branch] update-readme -> update-readme

Truy cập GitHub và xác nhận rằng kho lưu trữ có một cái mới branch:



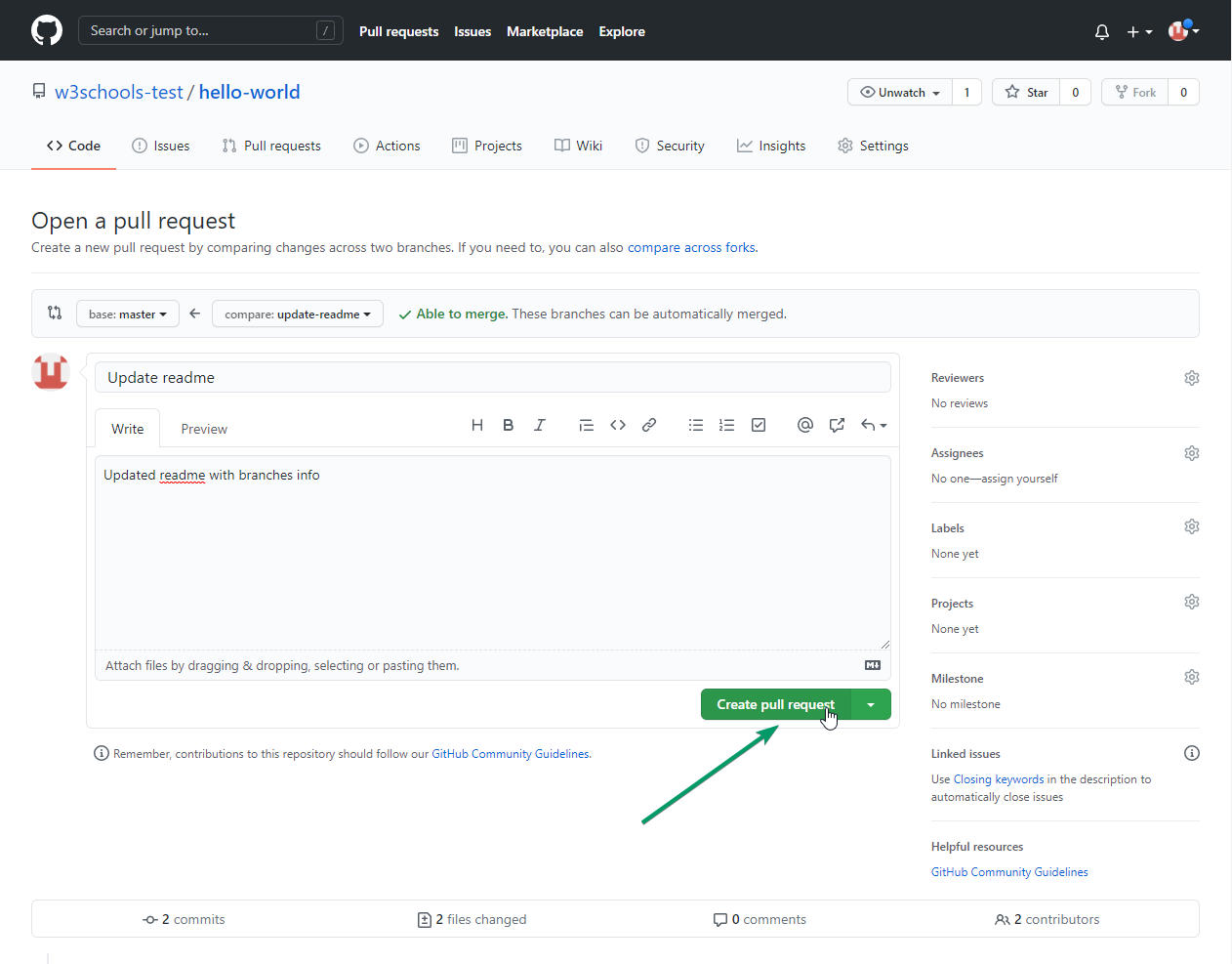
Trong GitHub, giờ đây chúng tôi có thể xem các thay đổi và merge chúng sẽ chuyển thành bản chính branch nếu chúng tôi chấp thuận.

Nếu bạn nhấp vào "So sánh & kéo yêu cầu", bạn có thể xem qua các thay đổi đã thực hiện và các tệp mới được thêm vào:



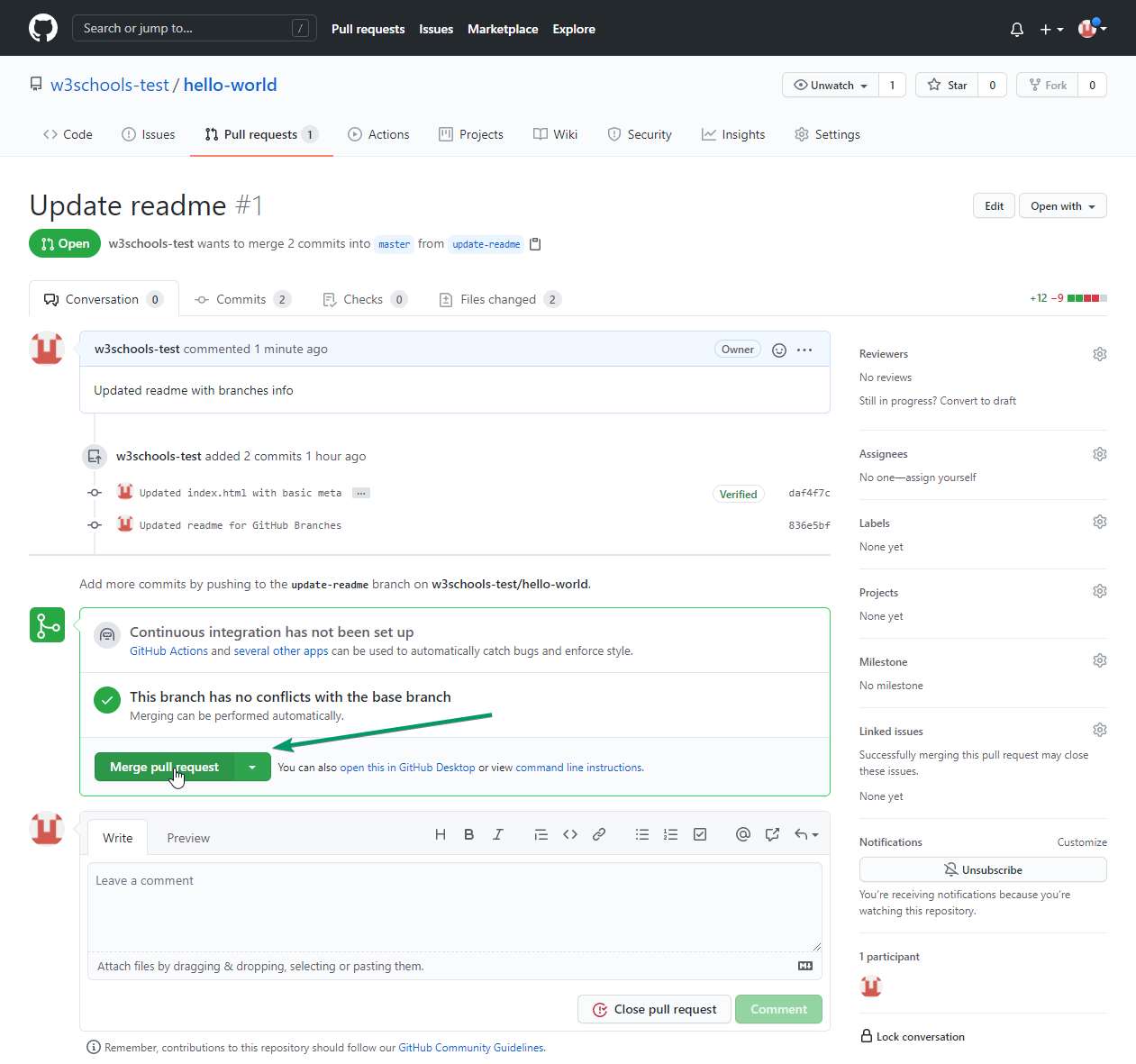
**Lưu ý:** So sánh này cho thấy cả những thay đổi từ update-readme và html-skeleton bởi vì chúng tôi đã tạo nhánh mới TỪ html-skeleton.

Nếu các thay đổi có vẻ ổn, bạn có thể tiếp tục, tạo một pull request:



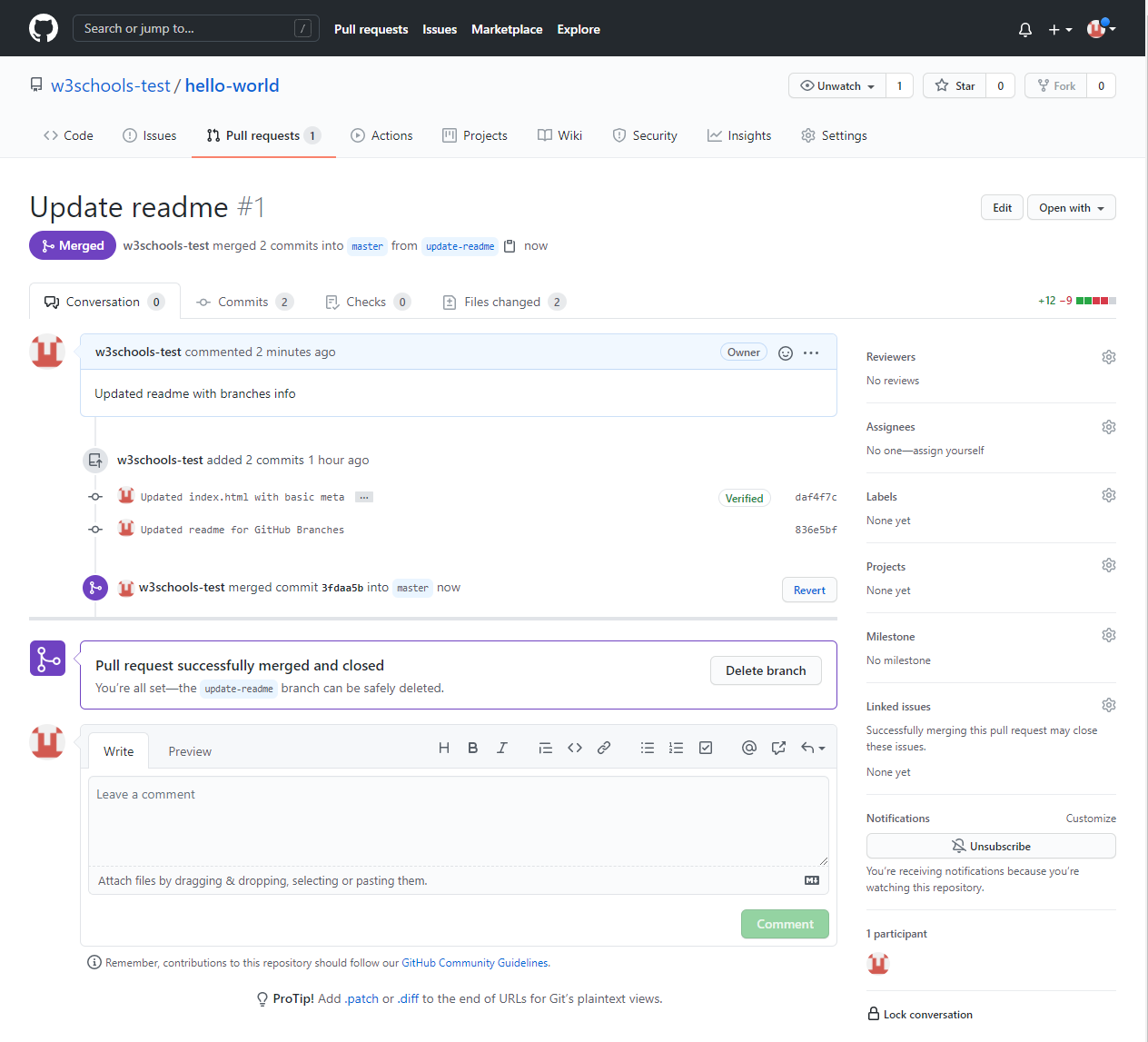
Yêu cầu kéo là cách bạn đề xuất các thay đổi. Bạn có thể yêu cầu một số người xem lại các thay đổi của bạn hoặc lấy phần đóng góp của bạn và hợp nhất nó vào chi nhánh của họ.

Vì đây là kho lưu trữ của riêng bạn, bạn có thể  merge tự lấy yêu cầu kéo của mình:

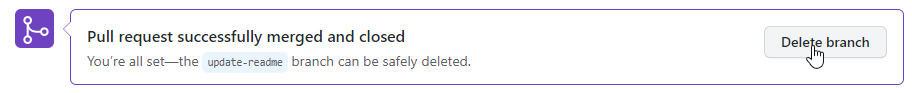


Yêu cầu kéo sẽ ghi lại các thay đổi, nghĩa là bạn có thể xem qua chúng sau để tìm ra những thay đổi đã thực hiện.

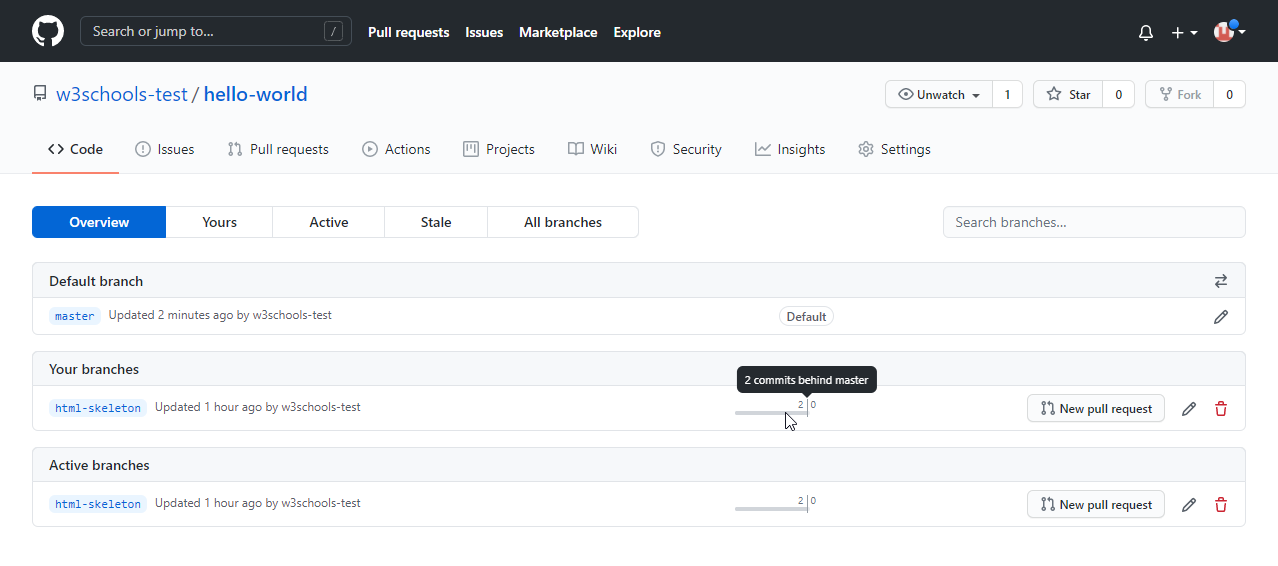
Kết quả sẽ là một cái gì đó như thế này:



Để repo không trở nên quá phức tạp, bạn có thể xóa nhánh hiện không sử dụng bằng cách nhấp vào "Xóa nhánh".



Sau khi bạn xác nhận rằng các thay đổi từ nhánh trước đó đã được đưa vào, hãy xóa luôn:



## **Lệnh GIT mức độ cơ bản**

Trong phần này chúng ta sẽ tìm hiểu những lệnh GIT cơ bản và được sử dụng thường xuyên nhất.

### git config

Git config là câu lệnh mà chúng ta phải thực thi đầu tiên cài đặt git lên máy. Câu lệnh này sẽ giúp các bạn thiết lập tên và email cá nhân của bạn, những thông tin này sẽ đính kèm trong mọi commit của bạn, đều này sẽ rất hữu ích khi chúng ta muốn biết đoạn code nào đó đã được ai triển khai để có thể thảo luận trong trường hợp chúng ta không hiểu rõ đoạn code đấy sử dụng cho mục đích gì.

$ git config --global user.name "Your name"

**$** git config --global user.email "Your email

### git version

Nhìn vào câu lệnh chắc hẵn chúng ta cũng đoán ra được câu lệnh này dùng để kiểm tra phiên bản git đang sử dụng trên máy.

**$** git version

### git init

Đây là câu lệnh đầu tiên khi chúng ta bắt đầu một dự án mới, câu lệnh này sẽ giúp chúng ta tạo một repository mới, sau đó nó sẽ được sử dụng để lưu trữ và quản lý mã nguồn trong repository này.

$ git init

// Hoặc bạn có thể đặt tên cho repo với lệnh

**$** git init **<**your repository name**>**

### git clone

Câu lệnh này sẽ giúp chúng ta download một repository đã tồn tại sẵn trên khô lưu trữu (github, gitlab v.v) về máy.

git clone **<**your project URL**>**

### git add

Git add là câu lệnh giúp chúng ta thêm tất cả các file code mới mới hoặc các file code được chỉnh sửa vào repository.

$ git add your\_file\_name - Thêm một file**(** thêm mới hoặc chỉnh sửa**)** vào staging area

**$** git add **\*** - Thêm tất cả các file **(**thêm mới hoặc chỉnh sửa**)** vào staging area

### git commit

Đây là câu lệnh được sử dụng phổ biến nhất, câu lệnh này sẽ giúp chúng ta lưu các thay đổi ở các file trong vùng staging area xuống repository.

Có thể hiểu git add dùng để thêm thêm các file được thay đổi hoặc thêm mới vào vùng staging area, và chúng sẽ sẵn sàng để commit và sau đó những thay đổi này sẽ được lưu xuống repository.

$ git commit -m “your useful commit message”

### git status

Câu lệnh này cho phép bạn xem tình trạng hiện tại của mã nguồn như có bao nhiêu file được thêm mới hoặc chỉnh sửa.  Những file nào đang nằm trong vùng staging area hoặc đang nằm ngoài staging area.

### git branch

Trong một Git repository luôn luôn tồn tại nhiều nhánh riêng biệt dùng để triển khai một tính năng nào đó độc lập với các nhánh khác.

Các lệnh branch các bạn có thể sử dụng:

$ git branch

Dùng để hiển thị tất cả các branch đang có.

$ git branch

Dùng để tạo một branch mới.

$ git branch -d **<**branch\_name**>**

Xoá branch.

### git checkout

Để di chuyển qua lại giữa các branch, chúng ta có thể sử dụng git checkout để đạt được điều này.

git checkout **<**branch\_name**>**

Ngoài ra các bạn có thể vừa chuyển qua một branch mới và tiện thể khởi tạo nếu chưa tồn tại với câu lệnh.

**$** git checkout -b **<**your\_new\_branch\_name**>**

## Lệnh GIT mức độ trung bình

Sau các lệnh GIT cơ bản thường xuyên được sử dụng, chúng ta sẽ tìm hiểu các lệnh ở mức độ trung bình, cường độ sử dụng ích hơn.

### git remote

Repository được các bạn khởi tạo với câu lệnh git init chỉ đang tồn tại trên máy local của các bạn. Nếu muốn lưu trữ repository này lên một dich vụ lưu trữu git từ xa nào đó chẳng hạn như gitlab, github thì các bạn cần phải sử dụng git remote để kết nối giữa chúng.

$ git remote add **<**shortname**>** **<**url**>**

Ví dụ

$ git remote add origin https://dev.azure.com/aCompiler/\_git/DemoProject

### git push

Khi đã kết nối giữa local và dịch vụ lưu trữ git, chúng ta cần sử dụng lệnh git push để đồng bộ những thay đổi được commit trên local lên dich vụ lưu trữ.

$ git push -u **<**short\_name**>** **<**your\_branch\_name**>**

Ví dụ

**$** git push -u origin feature\_branch

Ngoài ra trước khi sử dụng git push các bạn nên cấu hình origin và upstream.

$ git push --set-upstream **<**short\_name**>** **<**branch\_name**>**

Ví dụ

**$** git push --set-upstream origin feature\_branch

### git fetch

Git được sử dụng để làm việc nhóm, quản lý mã nguồn. Ngoài những commit của bạn thì còn vô số commit khác của các thành viên khác trong team. Sử dụng git fetch sẽ giúp chúng ta cập nhật tất cả những thông tin mới như commit, branch, v.v.

$ git fetch

### git pull

Câu lệnh này sẽ download tất cả những nội dung (không chỉ là metadata như git fetch) từ dịch vụ lưu trữ xuống local repository.

$ git pull **<**remote\_url**>**

### git stash

Git stash cho phép chúng ta lưu trữ các file được chỉnh sửa trong vùng nhớ tạm.

$ git stash

Nếu muốn xem tất cả các stash các bạn có thể sử dụng lệnh

$ git stash list

Nếu bạn muốn áp dụng các chỉnh sửa trong một stash nào đó lên branch hiện tại đang sử dụng.

$ git stash apply

or

**$** git stash pop

### git log

Với câu lệnh git log các bạn có thể xem tất cả những commit trước đó được sắp xếp theo thứ tự commit gần nhất cho đến những commit cũ hơn.

$ git log

### git shortlog

Nếu chỉ muốn xem git log với nội dung được tóm tắt ngắn gọn thì các bạn có thể sử dụng git shortlog.

$ git shortlog

### git show

Lệnh này dùng để xem thông tin chi tiết của một commit bất kỳ.

$ git show **<**your\_commit\_hash**>**

### git rm

Đôi lúc các bạn muốn xoá một file từ code base, trong trường hợp này các bạn có thể sử dụng git rm.

$ git rm **<**your\_file\_name**>**

### git merge

Git merge cho phép các bạn kết mã nguồn và những thay đổi trên một branch khác vào branch hiện tại.

$ git merge **<**branch\_name**>**

Câu lệnh này sẽ kết hợp những thay đổi trên branch có tên là <branch\_name> vào branch hiện tại.

## Lệnh GIT mức độ nâng cao

Những câu lệnh ở mức độ nâng cao thường ít được sử dụng, và yêu cầu các bạn phải có kiến thức đủ tốt về git trước khi sử dụng. Và hãy sử dụng chúng thật cẩn thận nhé.

### git rebase

Git rebase tương tự như git merge, nó sẽ kết hợp 1 branch vào branch hiện tại với một ngoại lệ, git rebase sẽ ghi lại tất cả các lịch sử commit.

Bạn nên sử dụng lệnh Git rebase khi bạn có nhiều branch riêng dùng để hợp nhất thành một branch duy nhất. Và nó sẽ làm cho lịch sử commit trở nên tuyến tính và dễ truy vết hơn.

$ git rebase **<**base**>**

### git bisect

Git bitsect giúp bạn tìm ra những bad commit.

Để bắt đầu sử dụng

**$** git bisect start

Cho git bisect biết về một commit tốt

**$** git bisect good a123

Cho git bisect biết về một commit xấu

**$** git bisect bad z123

### git cherry-pick

Git cherry-pick là một lệnh hữu ích. Đó là một lệnhcho phép bạn chọn bất kỳ commit nào từ một branch bất kỳ và áp dụng nó vào một branch hiện tại.

$ git cherry-pick **<**commit-hash**>**

### git archive

Lệnh Git archive sẽ kết hợp nhiều tệp thành một tệp duy nhất. Nó giống như một tiện ích zip, vì vậy nó có nghĩa là bạn có thể giải nén các tệp lưu trữ để lấy các tệp riêng lẻ.

$ git archive --format zip HEAD **>** archive-HEAD.zip

### git pull –rebase

Nếu bạn muốn download content từ dịch vụ lưu trữ và dùng rebase thay vì merge thì có thể sử dụng

$ git pull --rebase

### git blame

Nếu bạn cần kiểm tra nội dung của bất kỳ tệp nào, bạn cần sử dụng git blame. Nó giúp bạn xác định ai đã thực hiện các thay đổi đối với tệp.

$ git blame **<**your\_file\_name

### git tag

Trong Git, các thẻ tag rất hữu ích và bạn có thể sử dụng chúng để quản lý bản phát hành. Bạn có thể coi thẻ Git giống như một nhánh sẽ không thay đổi. Nó quan trọng hơn đáng kể nếu bạn đang phát hành công khai.

$ git tag -a v1.0.0

### git verify-commit

Lệnh git verify-commit sẽ kiểm tra chữ ký gpg. GPG hoặc “GNU Privacy Guard” là công cụ được sử dụng trong các tệp ký tên và chứa các chữ ký của chúng.

$ git verify-commit **<**commit**>**

### git verify-tag

Tương tự git verify commit, các bạn có thể kiểm tra trên tag với lệnh

$ git verify-tag **<**tag**>**

### git diff

Nếu các bạn muốn so sánh một file code nào thay đổi những gì trước khi commit thì các bạn có thể sử dụng

$ git diff HEAD **<**filename**>**

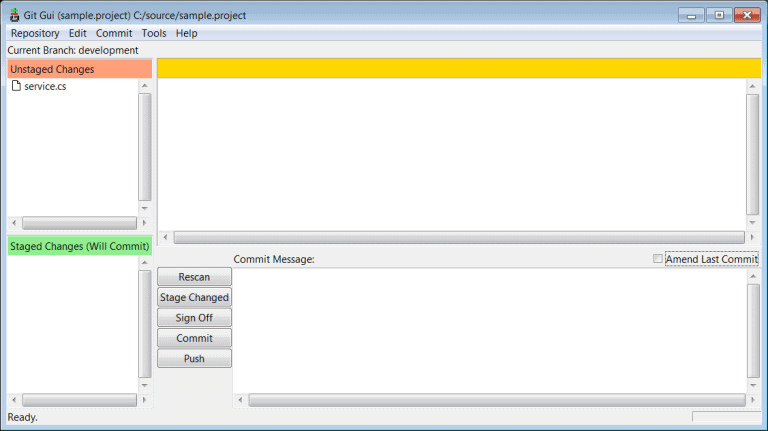
Để kiểm tra sự khác nhau giữa mã nguồn hiện tại đã được thay đổi so với local repo

**$** git diff HEAD **<**filename**>**

So sánh 2 branch

### git citool

Git citool là một giải pháp thay thế đồ họa của Git commit.



$ git citool

### git mv

Đổi tên git file từ tên cũ sang tên mới.s

$ git mv **<**old-file-name**>** **<new**-file-name**>**

### git clean

Bạn có thể xoá sạch các nội dung được thay đổi với các untracked files (chưa được theo dõi) với lệnh git clean.

$ git clean

### git help

Giúp bạn xem tất cả các thông tin cần thiết để sử dụng git.

$ git help **<**git\_command**>**

### git whatchanged

ệnh này thực hiện tương tự như git log nhưng ở dạng thô. Và đó là do nguyên nhân lịch sử.

$ git whatchanged