# GIT – CÀI ĐẶT GIT

Trong phần này mình sẽ hướng dẫn các bạn cách cài đặt Git trên hệ điều hành Windows. Vì Git được phát triển dựa trên môi trường Linux nên với Windows bạn phải cài đặt phần mềm Git để thực hiện các dòng lệnh của Git.

## 1. Cài đặt Git

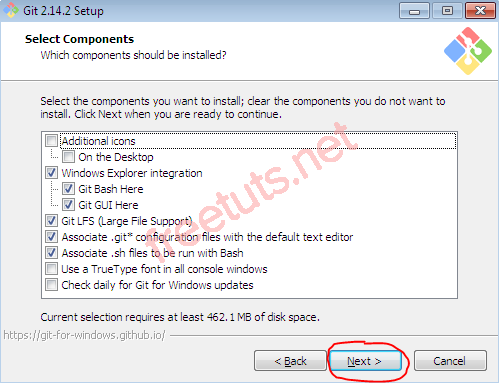
Bước đầu tiên bạn cần download Git về, bạn lên trang <https://git-scm.com/download/win> để download nhé, hệ thống sẽ tự nhận biết bạn đang sử dụng hệ điều hành 32bit hay 64bit để chọn đúng version.

Sau khi download xong bạn hãy chạy file đó và thực hiện các bước cài đặt như giao diện. Thực chất Git đã tự chọn các thông số tốt nhất khi cài đặt rồi nên bạn chỉ việc click **Next**là được.

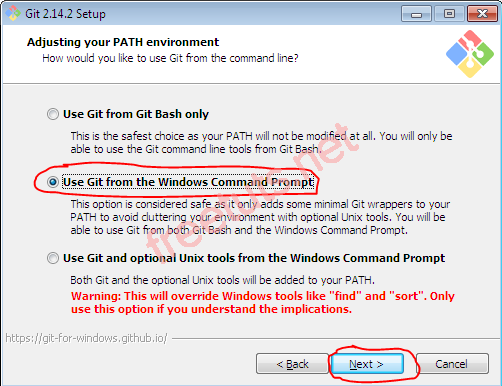
**Bước 1**: Click Next



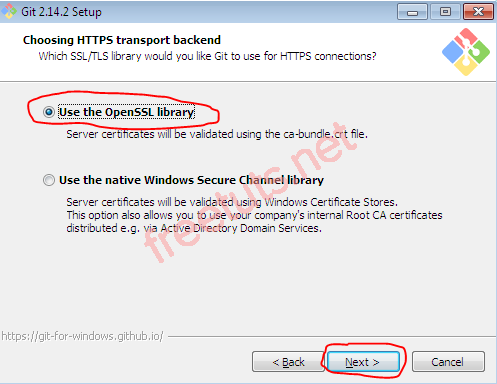
**Bước 2**: Click Next



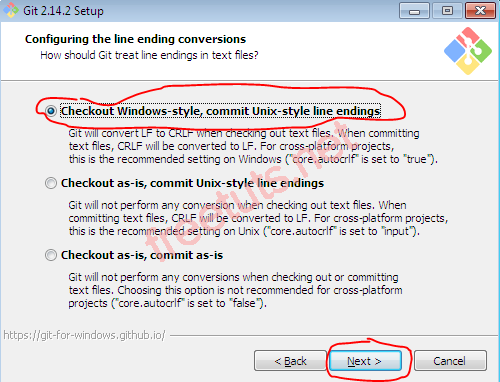
**Bước 3**: Click Next



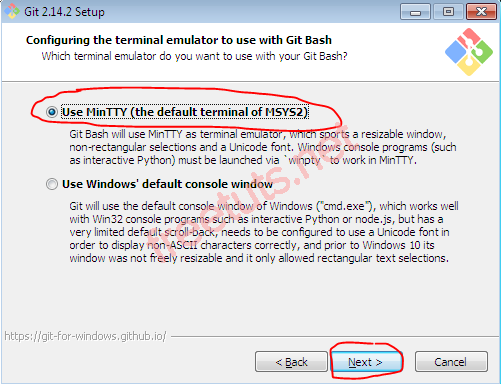
**Bước 4**: Click Next



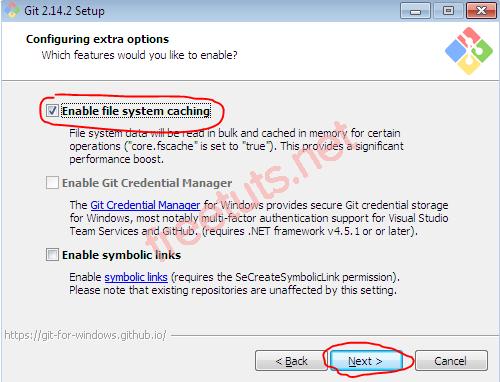
**Bước 5**: Click Next



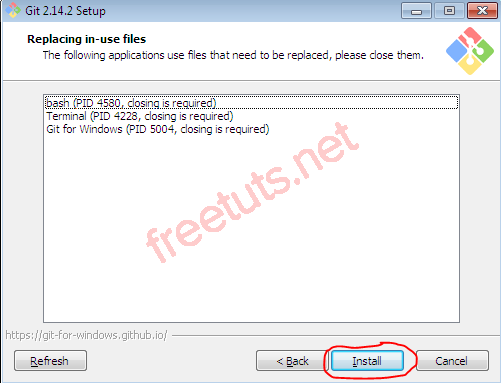
**Bước 6**: Click Next



**Bước 7**: Click Next



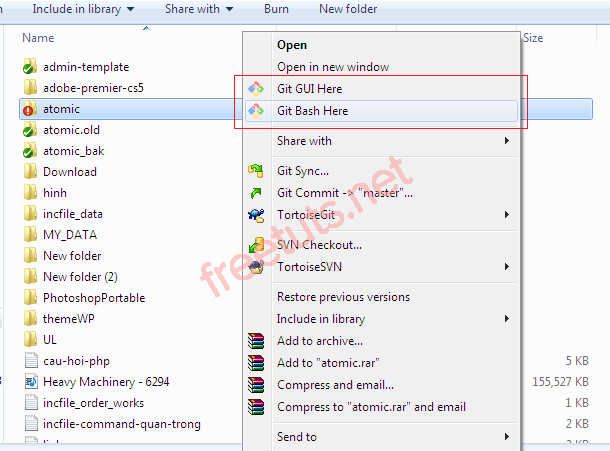
**Bước 8**: Click Install



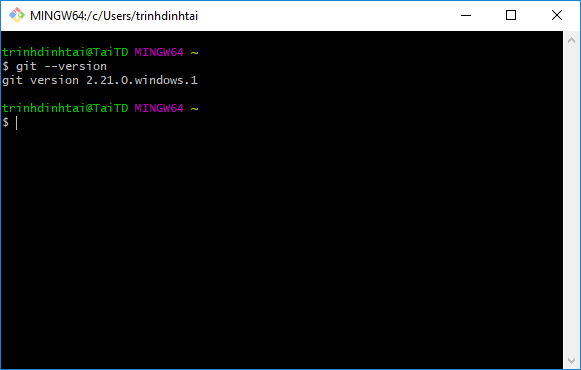
Bạn hay chờ cho việc cài đặt hoàn tất.

## 2. Kiểm tra Version

Sau khi cài đặt xong ta cần kiểm tra version để xem chắc chắn là đã cài đặt thành công. Bạn hãy vào một folder bất kì và click chuột trái, sau đó chọn **Git Bash Here** (nếu không có tức là cài đặt không thành công).



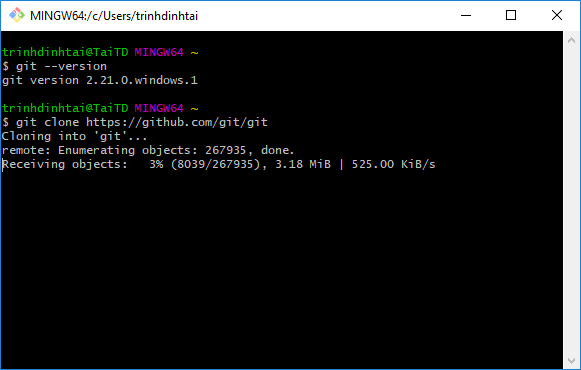
Một cửa sổ hiện ra, bạn hãy nhập vào lệnh git --version và nếu thấy kết quả như hình dưới đây là bạn đã cài đặt GIt thành công:



## 3. Download bằng lệnh

Trường hợp bạn đã cài đặt Git rồi và muốn download bản mới nhất thì có thể sử dụng lệnh clone để tải về.

|  |  |
| --- | --- |
| 1 | git clone <https://github.com/git/git> |



Sau khi download xong thì bạn tiến hành cài đặt bình thường.

**GIT – CẤU HÌNH USERNAME VÀ EMAIL**

Trước khi sử dụng Git thì bạn cần phải cấu hình hai thông số đó là username và email, nếu không bạn sẽ không thể thực hiện các thao tác như commit, clone, stash ...

Để cấu hình username thì bạn sử dụng lệnh sau:

git config --global user.name "Tên của bạn"

Để cấu hình email thì bạn sử dụng lệnh sau:

git config --global user.email "Email của bạn"

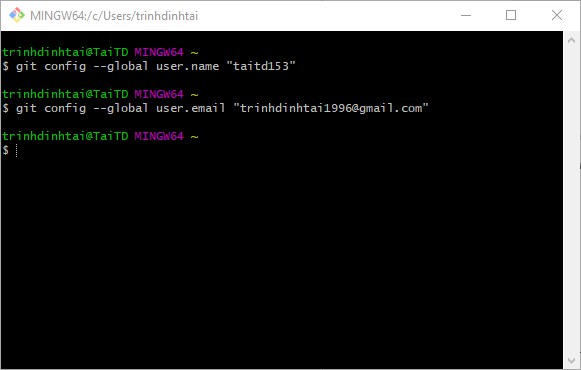
**Ví dụ**: Cấu hình hai thông tin cho Git  đó là "*taitd*" và "*trinhdinhtai1996@gmail.com*".

Bạn lần lượt nhập hai lệnh sau:

git config –global user.name "taitd"

git config --global user.email "trinhdinhtai1996@gmail.com"

Xem hình để hiểu rõ hơn cách làm.



# GIT – THAO TÁC VỚI THƯ MỤC

Để sử dụng Git nhuần nhuyễn thì đòi hỏi bạn phải có kỹ năng thao tác với các thư mục như: di chuyển giữa các thư mục, tạo một thư mục, xem danh sách thư mục ... Bản chất của Git là thực hiện lệnh thông qua màn hình console nên bạn hoàn toàn có thể sử dụng các lệnh console của Windows trong Git.

## 1. Các lệnh hay dùng

Sau đây là các lệnh thường dùng khi sử dụng Git để thao tác với thư mục.

### # Di chuyển tới một thư mục

Di chuyển tới một thư mục bất kì.

cd duongdandenthumuc

Chuyển sang ổ đĩa khác.

cd /TenODia

Lùi một cấp so với folder hiện tại.

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.

ls duongdandenthumuc

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.

rmdir DuongDanToiFolder

Nếu bạn chỉ nhập tên folder thì hệ thống sẽ xóa folder ngay tại 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.

pwd

# GIT - REPOSITORY

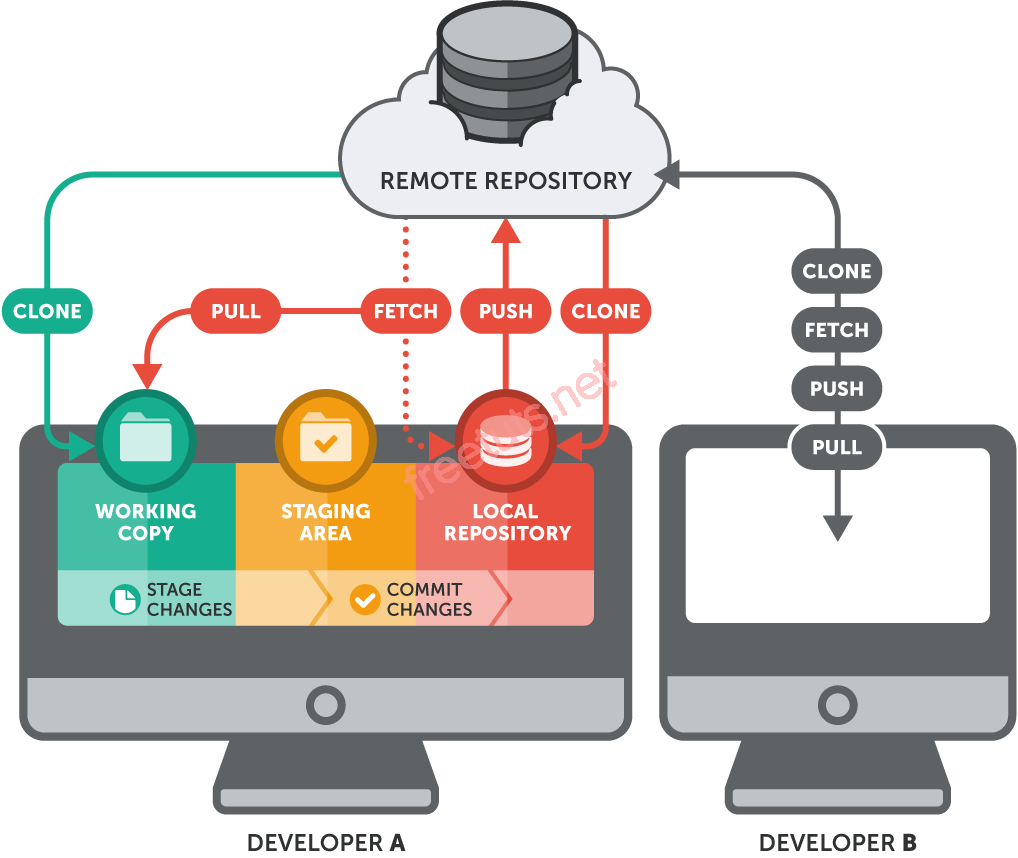
## 1. Repository là gì?

Repository hay còn gọi là Repo, dịch ra tiếng Việt có nghĩa là kho, đây chính là nơi chứa tất cả mã nguồn cho một dự án được quản lý bởi Git. Bạn cũng có thể hiểu một cách khác là Repository chính là thư mục chứa dự án của bạn trên local hoặc remote. Môi repo sẽ có hai cấu trúc dữ liệu chính đó là Object store và Index được lưu trữ ẩn trong thư mục .git

Có lại loại repository đó là **local repository** và **remote repository**.

* **Local repository**: Là repo được cài đặt trên máy tính của lập trình viên, repo này sẽ đồng bộ hóa với remote repo bằng các lệnh của git.
* **Remote repository**: Là repo được cài đặt trên server chuyên dụng, điển hình hiện nay là Github.

Bạn có thể tham khảo hình của git-tower.com dưới đây để hiểu rõ hơn.



## 2. Lệnh tạo Local Repository

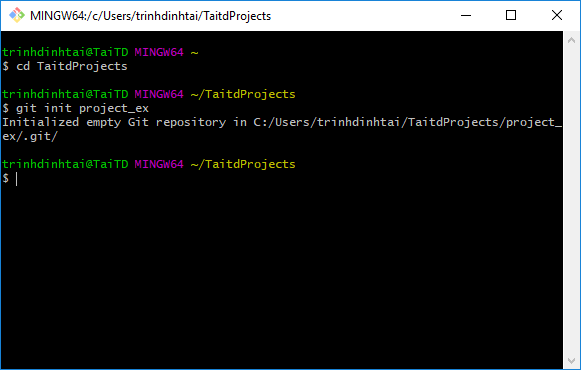
Trước hết, để tạo một repository thì bạn cần truy cập vào thư mục của mã nguồn với lệnh cd, sau đó sử dụng lệnh git init để khởi tạo repository trong thư mục đó. Ở ví dụ này, mình sẽ tạo tạo ra một thư mục mới để chứa code sau này và khởi tạo repository cho nó, mình sẽ dùng lệnh git init ten\_folder để nó tự khởi tạo thư mục.

**Bước 1**: Truy cập đến folder của dự án

cd TaitdProjects

**Bước 2**: Sử dụng lệnh git init để git tự động thực hiện thiết lập repo.

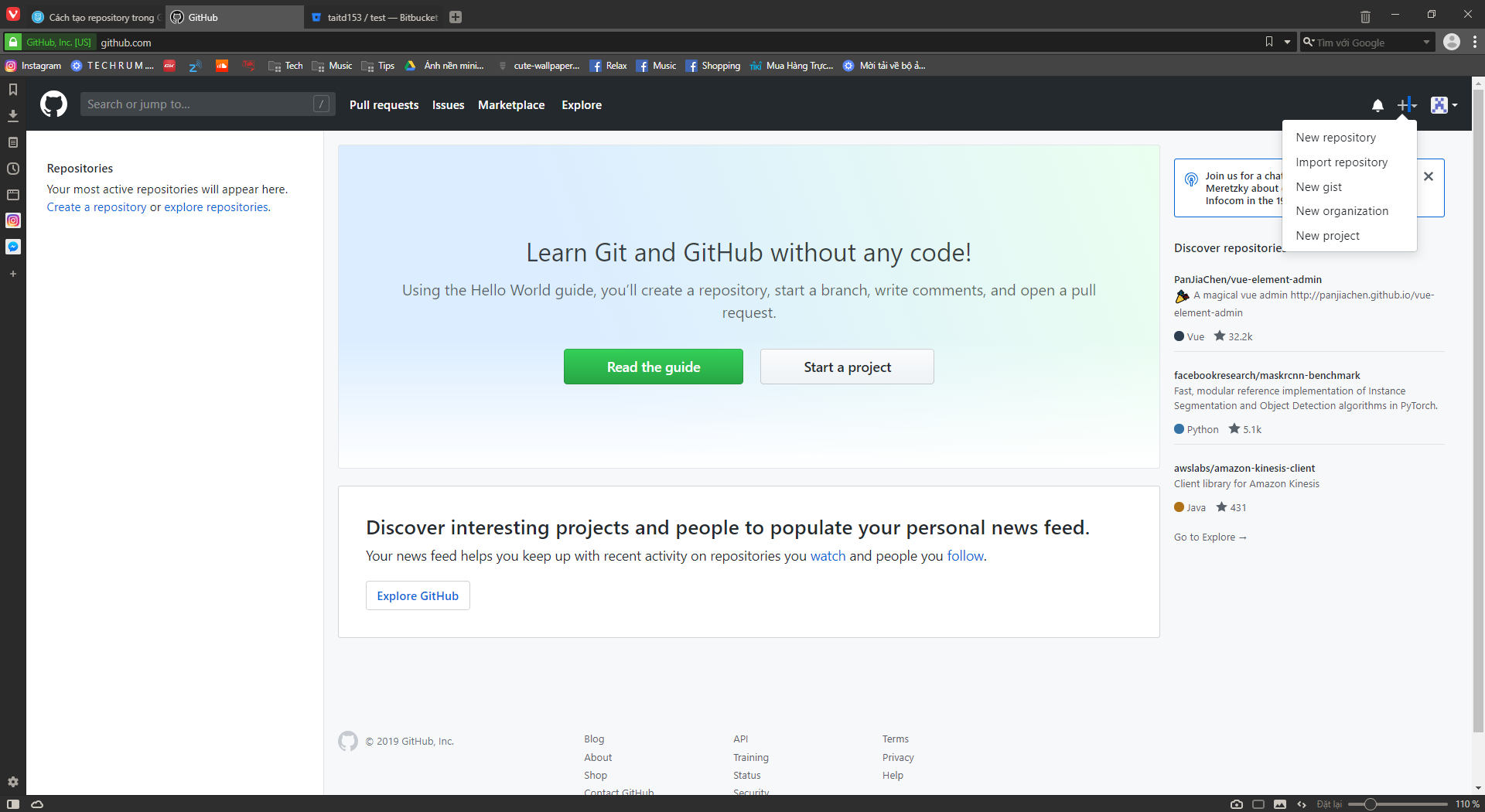
git init



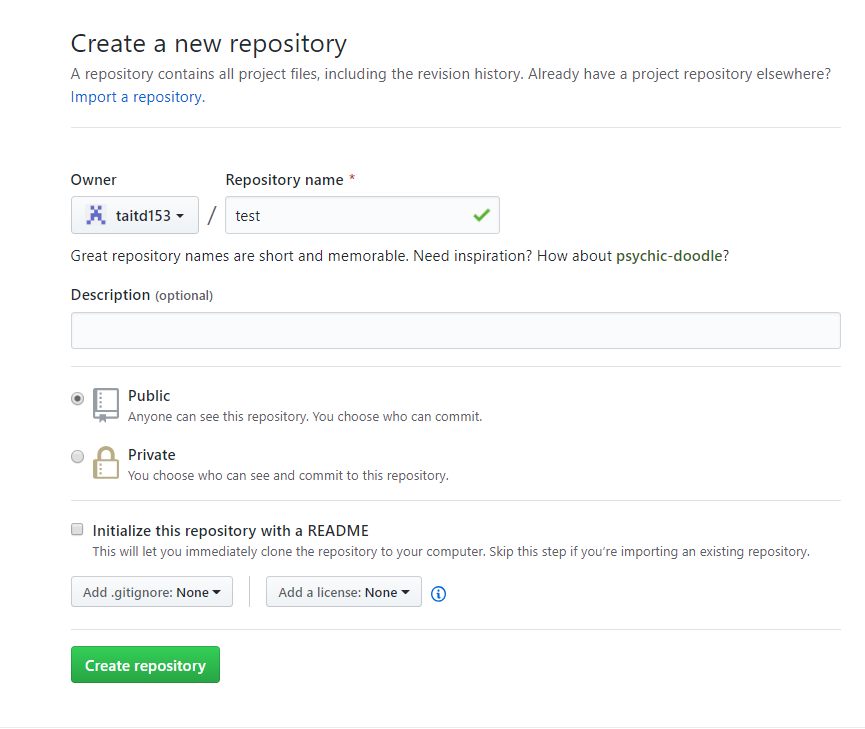
Ở đoạn trên, nó hiển thị dòng thông báo mình đã khởi tạo một kho Git trống tại đường dẫn như trên. Lưu ý rằng thư mục ẩn .git/ là nơi nó sẽ chứa các thiết lập về Git cũng như lưu lại toàn bộ thông tin về kho chứa, bạn **không cần đụng chạm gì vào thư mục**.git/ này.

## 3. Tạo Respository trên Github

Trước tiên bạn cần đăng nhập vào [Github](https://github.com/), sau đó ấn vào dấu + trên menu và chọn New repository.



Bạn sẽ cần đặt tên cho kho chứa của bạn. Bạn có thể chọn loại kho chứa là **Public** (ai cũng có thể clone) và **Private** (chỉ có những người được cấp quyền mới có thể clone).



*Điền thông tin khi tạo Repository trên Github*

Khi tạo xong nó sẽ dẫn bạn tới trang hướng dẫn làm việc với kho chứa vừa tạo. Và kho chứa của bạn bây giờ sẽ có địa chỉ là https://github.com/$user-name/$repository, ví dụ <https://github.com/taitd153/test>.

Việc của bạn bây giờ là hãy clone cái kho chứa này về máy của mình bằng lệnh git clone địa\_chỉ.

**$** git clone https://github.com/taitd153/test

Bây giờ hãy truy cập vào thư mục working tree (thư mục vừa clone repository về) và thử tạo ra một file tên là demo.txt, sau đó dùng lệnh git add để đưa file này vào Staging Area.

**$** cd test

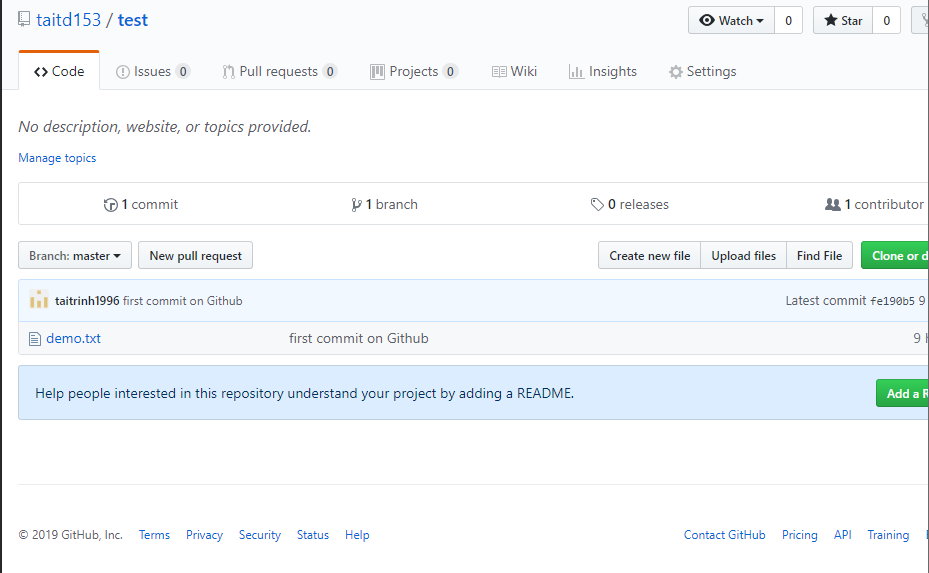
**$** git add demo.txt

**$** git commit -m "First commit on Github"

Tuy nhiên sau khi commit xong, tập tin đã được commit sẽ vẫn không thể xuất hiện trong kho chứa trên Github mà bạn phải làm thêm một việc nữa đó là dùng lệnh git push để đẩy các tập tin đã được commit lên Github. Lưu ý rằng **bạn sẽ cần nhập tài khoản và mật khẩu Github**.

**$** git push origin master

**origin** nghĩa là tên **remote** (xem ở bài sau) và **master** là tên **branch**, hai cái này mình sẽ giải thích kỹ hơn ở bài riêng của nó. Bây giờ bạn có thể kiểm tra kho chứa của bạn trên Github rồi đó.



*Kết quả sau khi push mã nguồn đã được commit lên Github*

Có thể bạn sẽ thấy mỗi khi push lên Github thì nó hiển thị ra thông báo quá dài dòng, bạn có thể cho ẩn các thông báo đi mà chỉ hiển thị dòng khai báo username và password trên Github thì có thể thiết lập với lệnh này.

**$** git config --global push.default simple

Tuy nhiên nếu bạn không thích bị hỏi mật khẩu nữa thì có thể sử dụng SSH với Github, mình sẽ nói cái này sau.

# GIT – COMMIT VÀ STAGING AREA

Trong bài [tạo repository cho Git](https://thachpham.com/tools/cach-tao-repository-cho-git.html) mình có nhắc qua về cụm từ **Staging Area** và một tính năng là **commit**(ủy thác), vậy hai cái này là gì thì mình sẽ giải thích kỹ hơn trong bài này để bạn biết cách sử dụng cho đúng.

## 1. Staging Area là gì ?

Staging Area nghĩa là một khu vực mà nó sẽ được chuẩn bị cho quá trình commit. Trước hết, bạn cần phải hiểu rằng trong các hệ thống quản lý phiên bản (Version Control System) thì các dữ liệu sẽ được lưu trữ ở hai nơi, một là thư mục bạn đang làm việc trên máy tính (working tree, mình không nhắc lại nữa đâu) và một là kho chứa mã nguồn (repository) sau khi bạn đã thực hiện thay đổi (ví dụ như kho chứa trên [Github](https://github.com/)).

Nhưng với Git thì nó có thêm một lựa chọn nữa đó là có thêm một khu vực trung gian gọi là **Staging Area** và đây chính là một lợi thế lớn của Git. Staging Area nghĩa là khu vực sẽ lưu trữ những thay đổi của bạn trên tập tin để nó có thể được commit, vì muốn commit tập tin nào thì tập tin đó phải nằm trong Staging Area. Một tập tin khi nằm trong Staging Area sẽ có trạng thái là **Stagged** (xem thêm ở dưới).

[](https://thachpham.com/wp-content/uploads/2015/04/git-staging-area.png)

*Mô hình giải thích cách hoạt động của Staging Area.*

Và để đưa một tập tin vào Staging Area thì bạn sẽ cần phải sử dụng lệnh git add tên\_file mà mình đã có ví dụ ở phần trước.

## 2. Commit là gì

### Commit là gì và nó hoạt động ra sao?

Hiểu đơn giản hơn, commit nghĩa là một hành động để Git lưu lại một bản chụp (snapshot) của các sự thay đổi trong thư mục làm việc, và các tập tin và thư mục được thay đổi đã phải nằm trong Staging Area. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của mã nguồn kèm theo tên và địa chỉ email của người commit. Ngoài ra trong Git bạn cũng có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một phân nhánh (branch) khác, đây là mấu chốt của việc bạn sẽ dễ dàng khôi phục lại các thay đổi trước đó mà mình có giới thiệu qua ở phần giới thiệu serie này.

Và tất nhiên, lệnh commit trong Git sẽ là git commit -m "Lời nhắn".

Và nếu bạn **muốn đưa tập tin lên repository thì bạn phải commit nó trước** rồi sau đó lệnh git push origin master sẽ có nhiệm vụ đưa toàn bộ các tập tin đã được commit lên repository.

#### Điều kiện gì để commit một tập tin?

Nếu bạn muốn commit một tập tin đó, bạn sẽ cần phải đưa tập tin đó vào trạng thái tracked bằng lệnh git add tên\_file. Trong git có hai loại trạng thái chính đó là Tracked và Untracked, cụ thể:

* **Tracked** – Là tập tin đã được đánh dấu theo dõi trong Git để bạn làm việc với nó. Và trạng thái Tracked nó sẽ có thêm các trạng thái phụ khác là Unmodified (chưa chỉnh sửa gì), Modified (đã chỉnh sửa) và Staged (đã sẵn sàng để commit).
* **Untracked** – Là tập tin còn lại mà bạn sẽ không muốn làm việc với nó trong Git.

Nhưng bạn phải nên biết rằng nếu tập tin đó đã được Tracked nhưng đang rơi vào trạng thái (Modified) thì nó vẫn sẽ không thể commit được mà bạn phải đưa nó về Staged cũng bằng lệnh git add.

#### Bỏ qua Staging Area để commit

Như mình có nói ở trên là một tập tin sau khi được thay đổi hay tạo mới thì nó phải được thêm vào Staging Area với lệnh git add. Tuy nhiên, bạn có thể đưa một tập tin đã được Tracked để commit mà không cần đưa nó vào Staging Area với tham số -a trong lệnh git commit. Ví dụ: git commit -a -m "Skipped Staging Are to commit".

# GIT - REMOTE RESPOSITORY VÀ ORIGIN

Quay trở lại bài [tạo repository](https://thachpham.com/tools/cach-tao-repository-cho-git.html), mình có hướng dẫn bạn cách tạo một repository trên dịch vụ Github và Clone nó về máy. Như vậy, cái Github là một máy chủ repository từ xa nên mình sẽ gọi nó là Remote Repository, nghĩa là repository này không nằm trên máy tính của bạn.

Và ở phần đó, bạn có thấy mình kêu các bạn gửi dữ liệu lên repository bằng cách dùng lệnh git push origin master sau khi commit không? Cái master là tên branch mà mình sẽ không nói ở bài này, nhưng cái origin trong đoạn đó chính là **tên remote repository**. Mặc định khi clone một repository thì nó tự đặt tên là origin.

Để kiểm tra tên remote, bạn có thể gõ lệnh git remote -v.

**$** git remote -v

origin https://github.com/taitd153/test (fetch)

origin https://github.com/taitd153/test (push)

Trong đó bạn có thể thấy cái repository mình đã clone đều được đặt tên là origin, và mỗi repository bạn có hai đều có hai hành động là fetch (lấy dữ liệu về từ server) và push (gửi dữ liệu lên server).

Nhìn lại đoạn lệnh git push origin master ở trên, điều đó có nghĩa là bạn gửi tất cả các thay đổi trên mã nguồn ở máy bạn lên remote tên là origin với branch master.

## 1. Đổi tên remote

Nếu bạn không thích tên origin thì có thể đổi tên nó lại nó bằng tên khác cho dễ quản lý nếu như bạn có nhiều remote trong một dự án với lệnh git remote rename tên\_cũ tên\_mới. Ví dụ mình cần đổi từ origin sang taitd thì sẽ đổi như sau:

**$** git remote rename origin taitd

**$** git remote -v

taitd https://github.com/thachphamblog/hoc-git.git (fetch)

taitd https://github.com/thachphamblog/hoc-git.git (push)

Bây giờ khi commit hay push bạn có thể gõ git push taitd master để gửi mã nguồn lên remote repository này.

## 2. Thêm một remote

Trường hợp bạn cần thêm một cái remote để lấy dữ liệu khi cần thì có thể sử dụng lệnh git remote add tên\_remote URL. Ví dụ mình cần remote một repository và đặt tên nó là inuit thì sẽ viết như sau:

**$** git remote add unuit https://github.com/thachpham92/inuit.css-web-template

**$** git remote -v

thach https://github.com/thachphamblog/hoc-git (fetch)

thach https://github.com/thachphamblog/hoc-git (push)

inuit https://github.com/thachpham92/inuit.css-web-template (fetch)

inuit https://github.com/thachpham92/inuit.css-web-template (push)

Sau đó nếu bạn muốn lấy dữ liệu từ cái inuit kia về thì chỉ cần sử dụng lệnh git fetch inuit.

**$** git fetch inuit

warning: no common commits

remote: Counting objects: 94, done.

remote: Total 94 (delta 0), reused 0 (delta 0), pack

Unpacking objects: 100% (94/94), done.

From https://github.com/thachpham92/inuit.css-web-te

\* [new branch] master -> inuit/master

Lưu ý là lệnh git fetch nó chỉ lấy về và lưu vào database của Git trên máy chứ không được gộp vào repository của bạn. Để gộp vào bạn có thể gõ thêm lệnh git merge inuit, trong đó inuit là tên remote.

Còn nếu bạn muốn nó lấy về trực tiếp mà không cần gộp thì sử dụng lệnh git pull tên\_remote, tuy nhiên mình khuyến khích bạn nên gộp vào branch khi cần và nhớ cẩn thận trong việc gộp, tốt nhất nên tạo thư mục mới trong thư mục làm việc của bạn rồi vào đó mà lấy về.

## 3. Sự khác nhau giữa clone, fetch và pull

Có thể bây giờ bạn đã biết được 3 lệnh để lấy dữ liệu về từ repository đó là git clone, git fetch và git pull. Nhưng cả ba loại đều là lấy dữ liệu, thế sự khác nhau của nó là gì?

#### **git clone**

Lệnh này sẽ sao chép toàn bộ dữ liệu trên repository và sao chép luôn các thiết lập về repository, tức là nó sẽ tự động tạo một master branch trên máy tính của bạn. Lệnh này chỉ nên sử dụng khi bạn cần tạo mới một Git mới trên máy tính với toàn bộ dữ liệu và thiết lập của một remote repository.

#### **git pull**

Lệnh này sẽ tự động lấy toàn bộ dữ liệu từ remote repository và gộp vào cái branch hiện tại bạn đang làm việc.

#### **git fetch**

Lệnh này sẽ lấy toàn bộ dữ liệu từ remote repository nhưng sẽ cho phép bạn gộp thủ công vào một branch nào đó trên thư mục Git ở máy tính.

Tạm thời bạn nên hiểu thế, ở bài branch bạn sẽ hiểu sâu hơn.

### Các loại giao thức của Remote Repository

Chúng ta không chỉ kết nối với một remote repository qua giao thức HTTP hay HTTPS mà còn có thể chọn nhiều giao thức khác, dưới đây là một vài giao thức remote repository.

#### **Local Repository**

Giao thức này nghĩa là bạn kết nối tới một repository nào đó trên chính máy tính của bạn và URL của giao thức sẽ có dạng /path/repository/.

#### **HTTP Repository**

Giao thức thông dụng nhất cũng như dễ hiểu nhất, thường được sử dụng nếu bạn dùng các dịch vụ remote repository như Github hay Assembla, nó sẽ bao gồm định dạng http://domain.com/repository.git hoặc https://domain.com/repository.git.

#### **SSH Repository**

Giao thức này thường được dùng trên các nhu cầu tạo một server repository riêng và kết nối thông qua giao thức SSH. Đường dẫn của giao thức này sẽ có dạng là user@server:/path/repository.git. Ở phần cuối serie mình sẽ hướng dẫn bạn cách tạo một repository server riêng và nó là lựa chọn rất tốt nếu bạn làm việc nhóm mà không cần phụ thuộc vào các dịch vụ như Github hay Assembla.

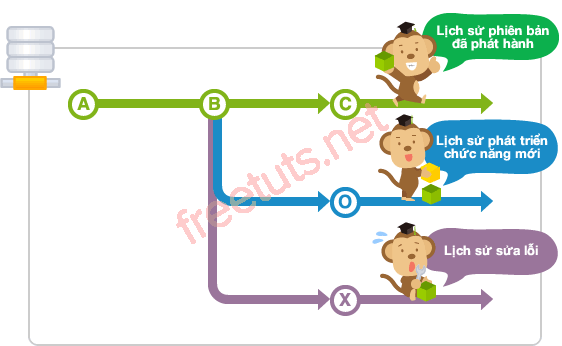
# GIT - BRANCH

Khi làm việc với Git bắt buộc chúng ta phải thông qua branch, sau đây chúng ta sẽ tìm hiểu khái niệm branch.

## 1. Branch là gì?

Đối với những dự án có nhiều thành viên tham gia thì mỗi thành viên sẽ nhận được rất nhiều task từ leader, vì vậy việc xử lý các task này trên cùng một thời gian là rất khó vì dễ bị đụng code. Để giải quyết vấn đề này thì chúng ta sẽ sử dụng branch của Git, tương ứng với mỗi nhiệm vụ chúng ta sẽ tạo một branch và làm việc trên đó, các branch này sẽ hoạt động riêng lẻ và không ảnh hưởng lẫn nhau.

Vậy branch là những phân nhánh ghi lại luồng thay đổi của lịch sử, các hoạt động trên mỗi branch sẽ không ảnh hưởng lên các branch khác nên có thể tiến hành nhiều thay đổi đồng thời trên một [repository](https://freetuts.net/git-tao-repository-1071.html), giúp giải quyết được nhiều nhiệm vụ cùng lúc.



## 2. Branch master

Khi bạn tạo một repository thì Git sẽ thiết lập branch mặc định là master, nghĩa là nó sẽ tự tạo một branch master và mọi hoạt động của ban lúc này đều nằm trên branch master. Chúng ta cũng có thể xem đây là branch mặc định đóng vai trò cập nhật dữ liệu và đồng bộ với remote repository.

Giả sử bạn đang có 10 tasks, lúc này bạn không nên làm việc trực tiếp trên branch master mà hãy tạo ra nhưng branch khác và branch master chỉ nên dùng để pull code từ **remote branch master** và merge với các branch còn lại. Mỗi branch làm việc sẽ được ghi lại lịch sử nên trong quá trình làm việc bạn hoàn toàn có thể rollback lại quá khứ dựa vào chỉ mục index mà Git đã lưu.

## 3. Tạo branch

Ở phần trước mình đã giới thiệu khái niệm [branch là gì](https://freetuts.net/git-branch-la-gi-branch-master-1072.html) rồi nhỉ, vậy thì trong phần này mình sẽ hướng dẫn bạn cách tạo mới một branch.

Như mình đã trình bày ở bài trước khi làm việc nhóm thì leader sẽ giao rất nhiều task tại nhiều vị trí khác nhau trong project, vì vậy nếu bạn làm việc trên 1 phân nhánh cho tất cả các task thì sẽ rất khó quản lý. Ví dụ bạn làm 2 task A và B trên cùng branch master, khi bạn làm task A xong thì leader bảo public task A thì lúc này bắt buộc bạn phải chờ cho task B xong, đây giống như là hàng đợi có thứ tự. Nhưng nếu bạn làm việc trên 2 phân nhánh tách biệt thì khi public phân nhánh A thì phân nhánh B sẽ không bị ảnh hưởng.

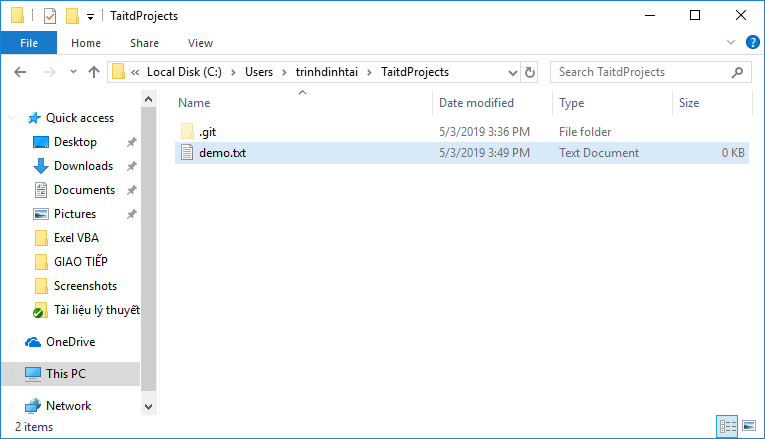
Giả sử dự án của mình nằm trong thư mục C:\Users\trinhdinhtai\TaitdProjects lúc này bạn cần phải di chuyển đến thư mục này như sau:

cd ~

cd TaitdProjects

**Leader giao dự án**: Hãy tạo hai file txt có nội dung tùy ý.

Nếu là lần đầu tiên sử dụng Git cho dự án thì sau khi tạo [repository](https://freetuts.net/git-tao-repository-1071.html) bạn phải thực hiện hai thao tác đó là **thêm mới một file** và **commit lên branch master** thì lúc này branch master mới được tạo. Minh chứng bằng cách sau khi thực hiện hai thao tác này bạn sẽ thấy xuất hiện một folder .git ẩn, đây là folder chứa history và mọi thứ của Git (ta không cần quan tâm đến các folder con của nó).



**Bước 1**: Tạo mới một file demo.txt nằm trong dự án

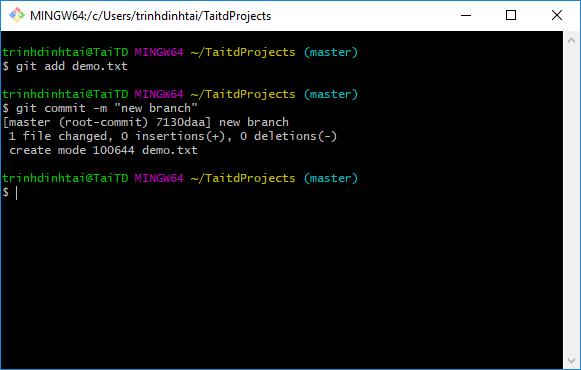
**Bước 2**: Chạy lệnh sau để add file vào Git history

git add demo.txt

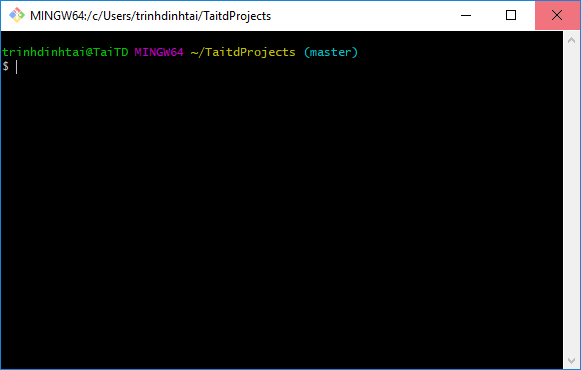
**Bước 3**: Chạy lệnh sau để commit lên branch master, lúc này branch master mới được tạo.

git commit -m "Tao file dau tien de tao branch master"

Xem hình sẽ dễ hiểu hơn.



Mặc định khi bạn mở Git bash lần đầu tiên thì branch master được chọn như hình sau.



Lúc này ta không nên làm việc trên branch master mà hãy tạo thêm 2 branch bằng cú pháp sau:

git branch <branchname>

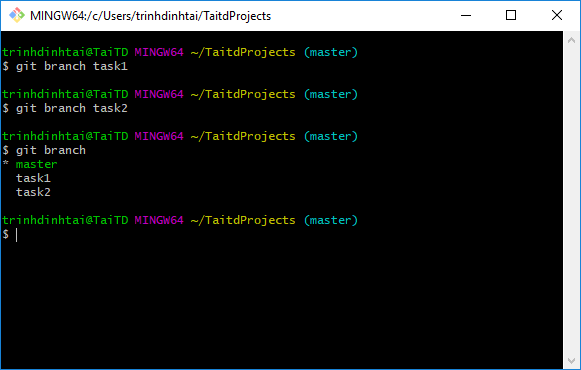
Dựa vào cú pháp này mình sẽ tạo 2 branch cho 2 task như sau:

git branch task1

git branch task2

Sau đó bạn nhập lệnh sau để xem danh sách branch, branch nào có dấu \* đằng trước là branch hiện tại.

Nếu như hình sau thì tức là bạn đã tạo thành công.



## 4. Checkout Branch

Nếu chúng ta làm việc trên nhiều branch thì phải thực hiện thao tác chuyển đổi branch.

Để chuyển đổi branch làm việc thì bạn sử dụng cú pháp sau.

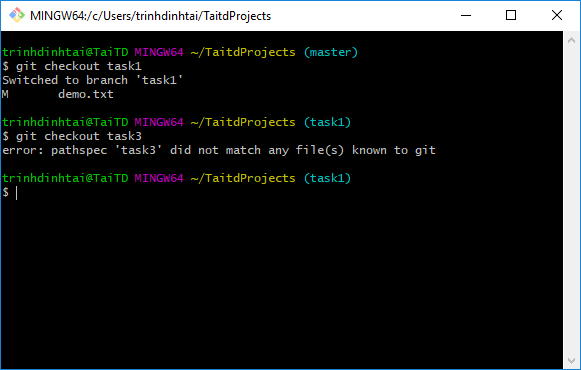
git checkout <branchname>

Như vậy muốn đổi branch làm việc từ master sang task1 thì ta làm như sau.

git checkout task1

Nếu kết quả trả về là "Switched to branch 'task1'" thì tức là bạn đã chuyển đổi thành công.

Nếu branch bạn nhập vào không tồn tại thì sẽ bị thông báo lỗi "error: pathspec 'branch\_name' did not match any file(s) known to git."



## 5. Git commit branch

Trước khi chuyển đổi sang branch khác thì bạn nên lưu lại tất cả những gì đã làm bằng lệnh **commit**. Nếu bạn không commit thì sau khi chuyển sang branch khác những thay đổi của bạn sẽ không được lưu lại trong history và sau này bạn không thể rollback lại được.

Để lưu lại những gì đã làm vào branch thì bạn sử dụng lệnh commit với cú pháp như sau:

git commit -m "message"

Trong đó message là tin nhắn bạn muốn lưu cho lần commit đó. Ví dụ sau khi sửa xong chức năng đăng nhập thì bạn sẽ commit với nội dung như sau:

git commit -m "sua phan dang nhap"

### **Trước khi commit**

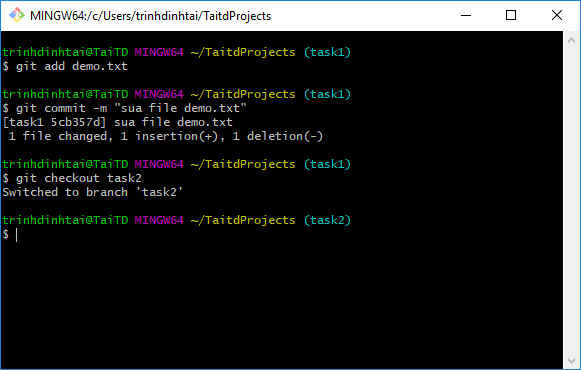
Trước khi commit thì bạn phải sử dụng lệnh git add <file\_name> để khai báo là sẽ commit file này, trường hợp bạn muốn commit tất cả các file có thay đổi thì sử dụng lệnh git commit ..

**Ví dụ**: Mình sửa file demo.txt và muốn commit nó lên thì thực hiện các bước như sau:

git add demo.txt

git commit –m "sua file demo.txt"

Xem hình để hiểu rõ hơn.



### **Minh chứng 2 branch tách biệt nhau**

Ở các bài trước mình có nói là mỗi branch sẽ tách biệt nhau, và để minh chứng điều đó thì mình sẽ đưa ra một ví dụ.

Tại branch task1 bạn hãy tạo một file file1.txt, sau đó thực hiện thao tác commit.  Tiếp theo bạn chuyển sang branch task2 thì lập tức file1.txt sẽ biến mất vì task2 không hề tồn tại file1.txt. Nhưng khi bạn chuyển sang lại branch task1 thì lập tức file đó lại xuất hiện.

## 6. Merge branch

Merge branch tức là bạn gộp hai branch lại với nhau, thao tác này thường dùng để merge branch khác vào branch master trước khi push lên [remote repository](https://freetuts.net/git-tao-repository-1071.html), hoặc merge hai branch thành một để giải quyết chung một task.

Để merge một branch bất kì vào branch hiện tại thì bạn sử dụng cú pháp sau:

git merge <branch\_name>

Giả sử bạn đang ở branch master, bây giờ bạn muốn merge branch task1 vào branch master thì làm như sau.

git merge task1

Giả sử bạn đang ở branch task1, bạn muốn merge branch task2 vào branch master thì làm như sau.

git checkout master

git merge task2

Bạn hãy tạo một file tên là task1.txt tại branch task1 với nội dung như sau:

“This is task1”

Tiêp theo bạn quay lại branch master bằng lệnh checkout.

git checkout master

Lúc này bạn thấy file task1.txt biên mất, bạn hãy chạy lệnh merge như sau.

git merge task1

Lập tức bạn sẽ thấy fie task1.txt đã xuất hiện, lý do là tất cả dữ liệu thay đổi ở branch task1 đã được trộn chung vào branch master.

## 7. Xóa local branch

Để xóa local branch thì bạn thực hiện lệnh sau:

$ git branch -d <branch\_name>

Trong đó branch\_name chính là tên của branch mà bạn muốn xóa. Lưu ý là bạn không thể xóa branch mà bạn đang làm việc, ví dụ bạn đang làm việc trên branch client thì bạn không thể xóa branch client, thay vào đó ban phải checkout sang branch khác để xóa branch client.

Khi chạy lệnh trên sẽ có hai trường hợp xảy ra.

**Trường hợp 1**: Nếu branch đang ở trạng thái fully merged thì bạn xóa bình thường.

**Trường hợp 2**: Nếu branch đang ở trạng thái not fully merged thì git sẽ cảnh báo, lúc này ban phải sử dụng lệnh sau để xóa.

$ git branch -D <branch\_name>

Trường hợp 2 này ban phải cân nhắc vì nếu xóa branch thì bạn sẽ mất đi các commit đã thực hiện trên đó.