|  |
| --- |
| **Sử dụng git bash cơ bản** |
| {30/11/2016} |
|  |



|  |  |
| --- | --- |
| Người lập văn bản : | Phê duyệt bởi: |
| *Đỗ Quang Hiệp* | *{Reviewer-FullName}* |

**Contents:**

[Bài 1: Bắt đầu – Cấu hình git lần đầu 4](#_Toc468658551)

[I. Cài đặt Git 4](#_Toc468658552)

[II. Danh Tính Của Bạn 4](#_Toc468658553)

[III. Kiểm Tra Cấu Hình 4](#_Toc468658554)

[Bài 2: Cách tạo một repository 5](#_Toc468658555)

[I. Khái niệm 5](#_Toc468658556)

[II. Tạo local repository 5](#_Toc468658557)

[III. Sao chép một kho chứa đã tồn tại 6](#_Toc468658558)

[Bài 3: Hiểu thêm về Commit và Staging Area 8](#_Toc468658559)

[I. Staging Area là gì? 8](#_Toc468658560)

[II. Commit là gì và nó hoạt động ra sao? 8](#_Toc468658561)

[III. Điều kiện gì để commit một tập tin? 9](#_Toc468658562)

[IV. Bỏ qua Staging Are để commit 9](#_Toc468658563)

[V. Untracked 9](#_Toc468658564)

[VI. Tracked 10](#_Toc468658565)

[VII. Chuyển tập tin từ Tracked về Untracked 11](#_Toc468658566)

[Bài 4: Git Log và Undo Commit 12](#_Toc468658567)

[I. Xem git log 12](#_Toc468658568)

[II. Lọc log với pretty 12](#_Toc468658569)

[III. Undo Commit 13](#_Toc468658570)

[IV. Bỏ tập tin ra khỏi Staging Area 14](#_Toc468658571)

[Bài 5: Đánh dấu commit với Tag 15](#_Toc468658572)

[I. Lightweight Tag và Annotated Tag 15](#_Toc468658573)

[II. Cách tạo Lightweight Tag 15](#_Toc468658574)

[III. Cách tạo Annotated Tag 16](#_Toc468658575)

[IV. Thêm Tag cho các commit cũ 16](#_Toc468658576)

[V. Nhập tag vào branch 17](#_Toc468658577)

[Bài 6 : Sơ lược Remote Respository và Origin 18](#_Toc468658578)

[I. Đổi tên remote 18](#_Toc468658579)

[II.Thêm một remote 18](#_Toc468658580)

[III. Sự khác nhau giữa clone, fetch và pull 19](#_Toc468658581)

[IV. Các loại giao thức của Remote Repository 19](#_Toc468658582)

[Bài 7: Branch – Kỹ thuật phân nhánh 21](#_Toc468658583)

[I. Branch trong git là gì? 21](#_Toc468658584)

[II. Cách tạo một branch 22](#_Toc468658585)

[III. Check out một branch 22](#_Toc468658586)

[IV.Gập dữ liệu từ một branch 23](#_Toc468658587)

[V. Xóa branch 23](#_Toc468658588)

[VI. Làm việc với Remote Branch 24](#_Toc468658589)

[Bài 8: Chia sẻ mẹo nhỏ dùng git bash 25](#_Toc468658590)

# **Bài 1: Bắt đầu – Cấu hình git lần đầu**

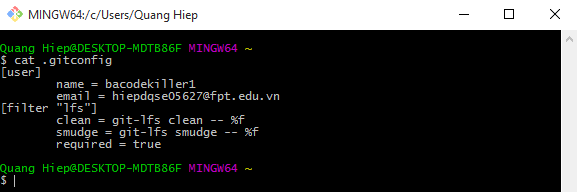
## **I. Cài đặt Git**

- Download và cài đặt git tại địa chỉ : [http://git-scm.com/download/win](https://git-scm.com/download/win)

## **II.** [**Danh Tính Của Bạn**](https://git-scm.com/book/vi/v1/B%E1%BA%AFt-%C4%90%E1%BA%A7u-C%E1%BA%A5u-H%C3%ACnh-Git-L%E1%BA%A7n-%C4%90%E1%BA%A7u#Danh-Tính-Của-Bạn)

- Việc đầu tiên bạn phải làm khi cấu hình Git là chỉ định tên tài khoản và địa chỉ e-mail. Điều này rất quan trọng vì mỗi Git sẽ sử dụng chúng cho mỗi lần commit, những thông tin này được gắn bất di bất dịch vào các commit. Mở git bash lên và gõ lần lượt các lệnh:  
**$ git config --global user.name "bacodekiller1"  
$ git config --global user.email "hiepdqse05627@fpt.edu.vn"**

## **III.** [**Kiểm Tra Cấu Hình**](https://git-scm.com/book/vi/v1/B%E1%BA%AFt-%C4%90%E1%BA%A7u-C%E1%BA%A5u-H%C3%ACnh-Git-L%E1%BA%A7n-%C4%90%E1%BA%A7u#Kiểm-Tra-Cấu-Hình)

- Sau khi thiết lập cấu hình xong bạn có thể kiểm tra thông tin chứng thực trên user của bạn bằng lệnh:  
 + **ls -a** :là lệnh liệt kê tất cả các thư mục hiện hành.   
 + **cat .gitconfig** : “cat” là lệnh để xem các nội dung chứa trong thư mục.  
 - Bạn cũng có thể kiểm tra bằng lệnh: **git config –list**  
 - Bạn cũng có thể kiểm tra giá trị của một từ khoá riêng biệt nào đó bằng cách sử dụng git **config {key}**ví dụ: **git config user.name**   
 \* **Lưu ý** : Sử dụng lệnh **cd ~/** để trỏ đúng vào thư mục gốc của Users.

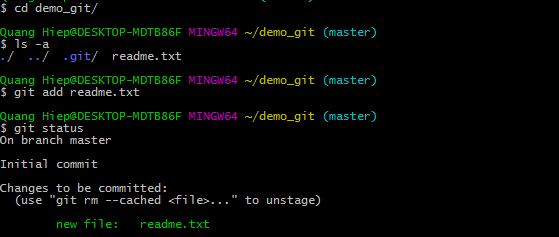
# **Bài 2: Cách tạo một repository**

## **I. Khái niệm**

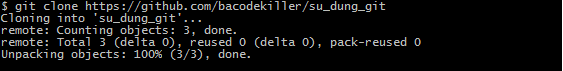
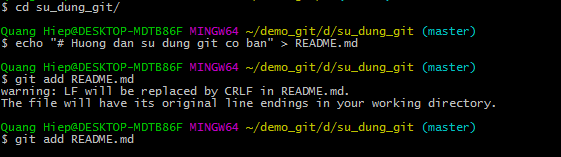
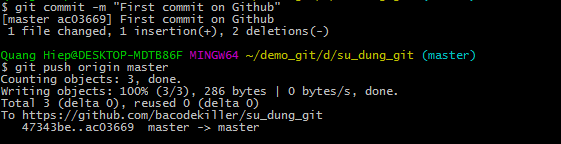
**- Repository**  (kho chứa) nghĩa là nơi mà bạn sẽ lưu trữ mã nguồn và một người khác có thể sao chép (clone) lại mã nguồn đó nhằm làm việc. **Repository** có hai loại là **Local** **Repository** (Kho chứa trên máy cá nhân) và**Remote Repository** (Kho chứa trên một máy chủ từ xa).

## **II. 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 tên\_folder** để nó tự khởi tạo thư mục:

  
 - Lệnh này sẽ tạo một thư mục mới có tên **.git**, thư mục này chứa tất cả các tập tin cần thiết cho kho chứa, đó chính là bộ khung/xương của kho chứa Git. Cho tới thời điểm hiện tại, vẫn chưa có gì trong dự án của bạn được theo dõi (track) hết.  
 - Nếu kho chứa của bạn đã có sẵn mã nguồn thì bạn cần **phải đưa các tập tin về trạng thái Tracked** nhằm có thể làm việc được với Git. Để làm việc này, trước hết bạn cần:  
 + Vào thư mực vừa tạo bằng lệnh: **cd tên\_thư\_mục**   
 + Sau đó dùng lệnh **ls -a** để liệt kê các file   
 + Sử dụng lệnh **git add tên\_file.**   
 + Sau đó bạn có thể sử dụng lệnh **git status** để xem danh sách các tập tin đã được tracked.  
 - Và sau khi tập tin đã được đưa vào trạng thái tracked và nếu một tập tin đã tracked thì nó phải được đưa vào lại Staging Area cũng bằng lệnh **git add** thì bạn mới có thể tiến hành ủy thác (**commit**) nhằm lưu lại bản chụp các thay đổi. Lệnh commit sẽ có cấu trúc **git commit -m** "**Lời nhắn**", lúc này tất cả các tập đang trong trạng thái tracked (file mới) hoặc một tập tin đã được tracked nhưng có một sự thay đổi mới thì sẽ được commit.  
****

## **III. Sao chép một kho chứa đã tồn tại**

- Nếu như bạn muốn có một bản sao của một kho chứa Git có sẵn - ví dụ như, một dự án mà bạn muốn đóng góp vào - câu lệnh bạn cần là **git clone [url].**   
 - Bây giờ hãy truy cập vào thư mục vừa clone repository về và thử tạo ra một file tên là **README.md**, sau đó dùng lệnh **git add** để đưa file này vào Staging Area.  
 **\* Lưu ý** : Muốn thêm các nội dung vào thì bạn phải được người tạo cấp quyền.  
  
****  
 - **echo** là một lệnh điển hình của kịch bản shell và tập tin bó dung để in một đoạn văn bản ra màn hình hoặc file.  
  
 ****   
 - Ở trên, mình sử dụng lệnh **git push origin master** để đẩy các tập tin đã được commit lên Github. Lưu ý rằng **bạn sẽ cần nhập tài khoản và mật khẩu Github**.  
 - **origin**nghĩa là tên remote và **master** là tên branch. Bây giờ bạn có thể kiểm tra kho chứa trên Github rồi đó.

# **Bài 3: Hiểu thêm về Commit và Staging Area**

## **I. 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**   
  


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 bài trước.

## **II. 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.  
 - 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.

## **III. 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:

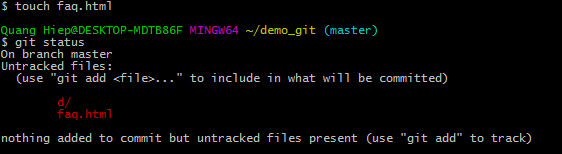
* **Tracked** – Là tập tin đã được đánh dấu theo dõi trong Git để bạn làm việc với nó. Và trạng thái Tracked nó sẽ có thêm các trạng thái phụ khác là Unmodified (chưa chỉnh sửa gì), Modified (đã chỉnh sửa) và Staged (đã sẵn sàng để commit).
* **Untracked** – Là tập tin còn lại mà bạn sẽ không muốn làm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

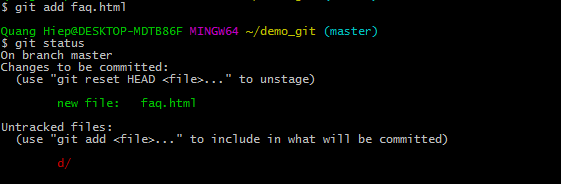
**-** Nhưng bạn phải nên biết rằng nếu tập tin đó đã được Tracked nhưng đang rơi vào trạng thái (Modified) thì nó vẫn sẽ không thể commit được mà bạn phải đưa nó về Staged cũng bằng lệnh git add.

## **IV. Bỏ qua Staging Are 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"**

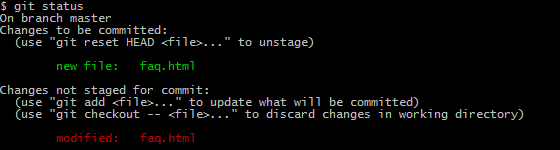
## **V. Untracked**

- Nếu bạn tạo ra hoặc thêm vào một tập tin mới vào trong thư mục làm việc của bạn thì nó sẽ ở trạng thái Untracked. Bây giờ mình thử tạo ra một tập tin mới tên là **faq.html**, sau đó dùng lệnh **git status** để xem trạng thái của Git trong thư mục làm việc. **

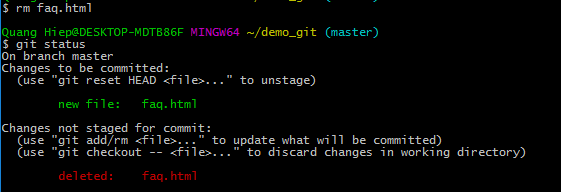
**-Note**: Lệnh **touch** là tạo ra một tập tin rỗng.  
 - Bây giờ bạn sẽ thấy nó đã liệt kê ra tên tập tin đang ở trạng thái Untracked. Để đưa nó về Tracked bạn sẽ sử dụng lệnh **git add** và xem lại trạng thái của nó.  
  
 - Vậy là tập tin của bạn đã được đưa vào trạng thái Staged và nó có thể được commit Tại sao? Vì bạn phải biết rằng nếu một tập tin ở trạng thái Untracked mà được đưa về Tracked thì nó sẽ nằm ở trạng thái Staged luôn, trừ khi bạn thay đổi nội dung tập tin này thì nó sẽ đưa về trạng thái Modified và nó không thể commit trừ khi bạn gõ lệnh **git add** cho nó. Bạn có thể thấy những tập tin Untracked còn lại.

**VI. Tracked**

- Một khi một tập tin đã được đưa về Tracked thì nó sẽ có thể thay đổi giữa 3 trạng thái khác nhau là **Modified**,**Unmodified** và **Staged**.  
 -Trước hết bây giờ mình đã có một tập tin mới đã được đưa về Staged với lệnh **git add** như ví dụ trên. Bây giờ mình tiến hành thay đổi nội dung của tập tin **faq.html** này và xem kết quả của lệnh **git status**.

 - Bạn có thấy sự kỳ lạ ở tập tin **faq.html** không? Đó là nó được hiển thị ở hai trạng thái Staged (có thể commit) và Modified (không thể commit) hay còn gọi là Unstaged. Sở dĩ có sự kỳ lạ đó ở đây là bởi vì trước đó bạn đã tạo ra tập tin **faq.html** và đưa về Tracked thì nó cũng đã được đưa về Staged để có thể commit. Tuy nhiên sau đó bạn lại chỉnh sửa nội dung của nó nên nó đã có một phiên bản khác nằm ở trạng thái Modified (không thể commit). Nếu bây giờ bạn gõ lệnh git commit để ủy thác nó thì bản chụp của tập tin faq.html ở lần cuối cùng bạn gõ lệnh git add sẽ được commit lên chứ nó không chứa các nội dung mà bạn vừa thêm vào. Và để nó có thể commit tập tin faq.html đã được chỉnh sửa thì bạn phải gõ lại lệnh **git add** **faq.html** lần nữa.

## **VII. Chuyển tập tin từ Tracked về Untracked**

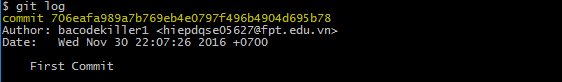
- Trong Git, bạn có thể đưa một tập tin từ Tracked về Untracked với lệnh **rm tên\_file**. Lệnh **rm** sẽ giúp bạn đưa tập tin về trạng thái Untracked nhưng không xóa hẳn trong ổ cứng.  
******

Còn nếu bạn muốn xóa nó luôn thì dùng lệnh **git rm -f tên\_file** và**nhớ cẩn thận** khi dùng lệnh này.

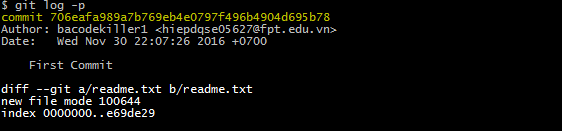
# Bài 4: Git Log và Undo Commit

Một việc bạn sẽ khá thường xuyên làm trong Git nếu làm việc theo nhóm đó là kiểm tra xem những ai đã commit vào dự án bạn đang làm việc, cũng như cách bạn undo lần commit trước đó nếu như bạn cảm thấy mình thiếu một tập tin nào đó trong lần commit trước để bổ sung vào.

## **I. Xem git log**

- Để xem lịch sử của các lần commit trước đó, bạn sử dụng lệnh **git log** là sẽ thấy.  


-Bạn thấy chứ? Mỗi lần commit sẽ có một checksum riêng, và nó cũng có ghi rõ ai là người commit vào và commit vào ngày bao nhiêu, lúc nào.  
 - Ngoài ra, bạn có thể chèn thêm tham số **-p**vào để hiển thị chi tiết của mỗi lần commit.

- ***-* Note**:   
+ Nếu log quá dài, dùng phím mũi tên lên xuống để đọc tiếp và ấn Ctrl + Z để thoát.  
+ Hoặc nếu bạn muốn chỉ muốn xem 1 lần commit gần nhất thì thêm tham số **-1** vào. + Bạn còn có thể sử dụng thêm một số tùy chọn xem log sau để tối ưu hơn quy trình đọc log.

* **--since, --after**: Xem các lần commit kể từ ngày nhất định.
* **--until**: Xem các lần commit trước từ ngày nhất định.
* **--author:** Xem các lần commit của một người nào đó.
* **--grep**: Lọc các chuỗi trong log và in ra.

**II. Lọc log với pretty**

- Tham số **--pretty** rất có ích nếu bạn muốn lọc xem một đối tượng nào đó trong lịch sử commit, ví dụ như chỉ xem lời nhắn commit hoặc chỉ xem email của người commit.  
 - Cách sử dụng tham số **--pretty**là bạn phải viết kèm các tag của nó như sau:  
 **git log –pretty=”%tag”  
 -** Các **%tag** phải dược đặt trong cặp dấu ngoặc kép và bạn có thể sử dụng nhiều **%tag** khác nhau.Danh sách các **%tag:**  
 % H - Cam kết băm

% H - viết tắt cam kết băm

% T - Cây băm

% T - viết tắt cây băm

% P - băm Chánh

% P - viết tắt băm mẹ

% an - Tên tác giả

% Ae - Tác giả e-mail

% ad - Tác giả ngày (định dạng tôn trọng -date = tùy chọn)

% Ar - Tác giả ngày, tương đối

% Cn - Tên có duyên

% Ce - email có duyên

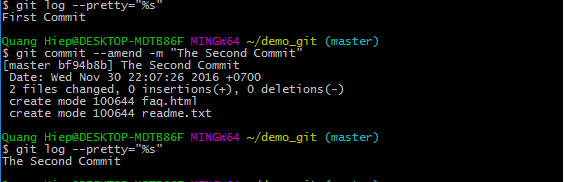
% Cd - ngày có duyên

% Cr - ngày có duyên, tương đối

% S - Chủ đề  
 -Ví dụ: 

## **III. Undo Commit**

-Nếu bạn cần xóa bỏ lần commit trước và cần undo để commit lại thì có thể sử dụng tham số  --**amend** trong lệnh **git commit.**

  
 - Lưu ý rằng undo nghĩa là bạn quay trở lại bước commit lần trước, do vậy nếu cần bổ sung tập tin nào vào để commit thì hãy đưa tập tin đó vào Staging Area trước.

## **IV. Bỏ tập tin ra khỏi Staging Area**

- Nếu bạn đã đưa một tập tin nào đó vào Staging Area nhưng bây giờ bạn muốn loại bỏ nó ra khỏi đây để không phải bị commit theo thì có thể sử dụng lệnh **git reset HEAD tên\_file.**

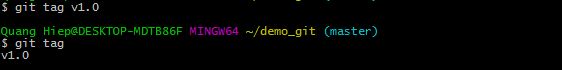
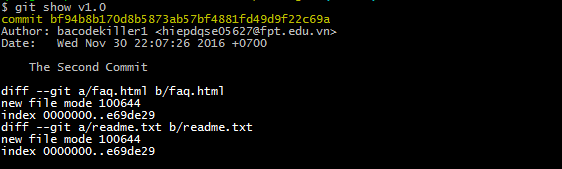
Bài 5: Đánh dấu commit với Tag

Trong khi bạn commit các lần chỉnh sửa của mình thì mọi thứ sẽ được đều lưu vào log mà mình đã hướng dẫn bạn cách xem **commit log** ở **bài trước**. Tuy nhiên nếu bạn commit quá nhiều thì sẽ gây khó khăn cho bạn về sau nếu cần xem lại thông tin của lần commit trước mà bạn có thể gắn thẻ đánh dấu (tag) cho mỗi commit và khi cần xem bạn chỉ cần sử dụng lệnh git **show tên\_tag** là đã có thông tin rất rõ ràng, ngoài ra nó còn giúp bạn dễ dàng diff (đối chiếu) sau này khi không cần nhớ checksum (dù chỉ cần nhớ vài ký tự đầu tiên) của mỗi commit mà chỉ cần nhớ tag, cũng như có thể tạo thêm branch từ tag để bạn thuận tiện hơn trong việc phân nhánh**.**

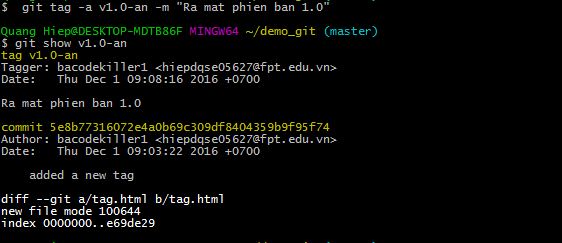
**I. Lightweight Tag và Annotated Tag**

\*Trong Git có hai kiểu tag chính đó là:  
 + **Lightweight Tag**: Các tag này chỉ đơn thuần là đánh dấu snapshot của commit.  
 + **Annotated Tag**: Với tag này, bạn có thể đặt tiêu đề cho tag, và khi xem nó sẽ có thông tin về người tag, ngày tag,…

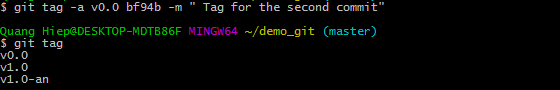
**II. Cách tạo Lightweight Tag**

- Trước tiên, bạn có thể gõ **git tag** để xem danh sách các tag có trong dự án của bạn. Sau đó để tạo thêm một tag, bạn có thể gõ **git tag tên\_tag** để tạo. Ví dụ như **v1.0** chẳng hạn.  
  
 - Bây giờ bạn có thể xem thông tin của lần commit được gắn tag này bằng lệnh **git show** **tên\_tag**. **Lưu ý** rằng lệnh trên nó sẽ đánh dấu lần commit cuối cùng của bạn vào **tag v1.0**.  


**III. Cách tạo Annotated Tag**

- Để tạo Annotated Tag thì bạn cũng sử dụng lệnh git tag nhưng sẽ có thêm tham số -a và tham số **-m** để thiết lập lời nhắn cho tag này.  
**** - Bạn có thể thấy khi show ra, cái Annotated tag sẽ có nhiều thông tin hơn là so với cái tag thông thường, và đây cũng là kiểu tag bạn nên sử dụng để có nhiều thông tin hơn.

**IV. Thêm Tag cho các commit cũ**

**-** Ở lệnh trên thì bạn chỉ tạo ra tag cho commit cuối cùng của bạn. Còn nếu bạn có rất nhiều commit trước đó mà cần gắn tag thì chỉ cần thêm mã checksum (hoặc một đoạn mã checksum) của lần commit đó.  
 - Để xem mã checksum của các lần commit trước đó thì bạn có thể sử dụng **git log** với tham số **--pretty** với giá trị **oneline**để lọc log nhé.  
  
 - Và bây giờ mình có thể đặt tag cho commit **The Second Commit** thì sẽ khai báo một đoạn mã checksum của nó vào lệnh **git tag** như sau.

- Mặc định lệnh git push sẽ không push các tag đã tạo lên repository mà bạn có thể dùng lệnh**git push --tags**để đẩy toàn bộ tag lên repository.

**V. Nhập tag vào branch**

- Mặc dù ở bài sau mình sẽ giải thích qua branch nhưng tạm thời bây giờ bạn có thể hiểu branch là **một phân nhánh trong một cây dự án** để bạn sửa mã nguồn mà không ảnh hưởng đến phân nhánh gốc (master).  
 - Bây giờ bạn có thể truy cập vào dữ liệu mà bạn đã commit thông qua tag kèm theo việc tạo một branch mới với lệnh **git checkout -b tên\_branch tên\_tag.** Ví dụ mình muốn đưa cái **tag v1.0-an** vào một branch mới tên version1 thì sẽ viết như sau:  


- Lúc này bạn đã tự động chuyển qua branch **version1** thay vì **master** ban đầu, kèm theo đó là dữ liệu của lần commit được **gắn tag v1.0-an**.  
 - Bây giờ nếu bạn muốn làm việc thêm với branch này thì cứ làm việc như khi bạn ở **master**. Để chuyển về lại master, bạn gõ lệnh **git checkout master.**  
 - Để push cái branch này lên bạn có thể sử dụng lệnh **git push origin version1**. Nếu bạn quên thì mình nhắc lại nhé, **origin** chính là tên địa chỉ remote của repository mặc định mà Git tự đặt khi bạn clone.

# Bài 6 : Sơ lược Remote Respository và Origin

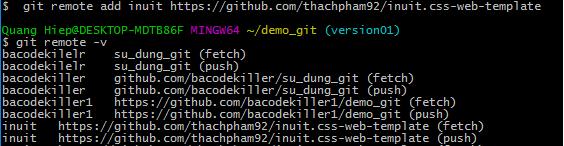
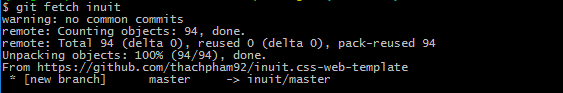
Quay trở lại bài [**tạo repository**](https://thachpham.com/tools/cach-tao-repository-cho-git.html)**,** mình có hướng dẫn bạn cách tạo một repository trên dịch vụ Github và Clone nó về máy. Như vậy, cái Github là một máy chủ repository từ xa nên mình sẽ gọi nó là Remote Repository, nghĩa là repository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


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.**

**I. 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 **bacodekiller1** thì sẽ đổi như sau: Bây giờ khi commit hay push bạn có thể gõ **git push bacodekiller1 master** để gửi mã nguồn lên remote repository này.

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

**III. Sự khác nhau giữa clone, fetch và pull**

1. 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.  
 2. 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.  
 3. 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.

**IV. 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.

1. Local RepositoryGiao 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/.**

2. 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 địnhdạng **http://domain.com/repository.git** hoặc **https://domain.com/repository.git.**

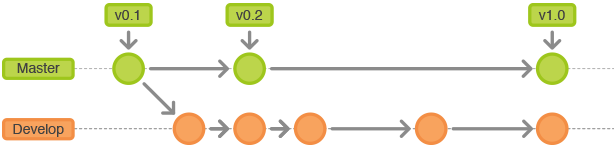
3. SSH Repositor

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.

Bài 7: Branch – Kỹ thuật phân nhánh

Trong khi làm việc với Git, bạn đã quá quen thuộc với việc chỉnh sửa mã nguồn, sau đó là commit mỗi khi chỉnh sửa xong và push lên remote repository nếu cần thiết. Nhưng bây giờ mình có một ví dụ đặt ra là mình muốn **tạo một phiên bản thử nghiệm với mã nguồn đang làm việc trong working tree hiện tại mà không gây ảnh hưởng đến các code hiện tại**. Vậy thì làm cách nào? Không lẽ clone một repository từ chính cái working tree hiện tại rồi sửa đổi hay sao? Như thế rất là mất công, mà lại không tối ưu và không thể đồng bộ hóa hoặc rất khó khăn để đồng bộ hóa.  
Mà trong Git, chúng ta sẽ sử dụng một giải pháp khác tuyệt vời hơn, dễ dàng hơn gọi là **phân nhánh** (branching), mà cụ thể là phân nhánh cái gì? Đó là phân nhánh trong working tree hiện tại bạn đang làm việc đấy, và mỗi nhánh chúng ta sẽ gọi nó là một **branch**.

**I. Branch trong git là gì?**

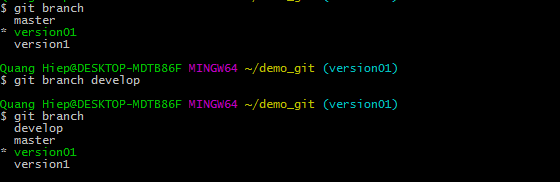
- Khi bắt đầu khởi tạo một repository hoặc clone một repository, bạn sẽ có một nhánh (branch) chính tên là **master** (bạn có thể hiểu master là một cái thân cây). Đây là branch mà sẽ chứa toàn bộ các mã nguồn chính trong repository.  
 - Đó là lý do tại sao, ở các phần trước khi chúng ta push hoặc pull hay làm một số việc khác thì lại có tham số **master**trong câu lệnh, đó nghĩa là chúng ta đang thực hiện thao tác trên branch **master.**

Nhánh Master là nhánh chính mặc định trong working tree. Bạn có thể tạo ra một (hoặc nhiều nhánh mới) với tên là Develop chẳng hạn.

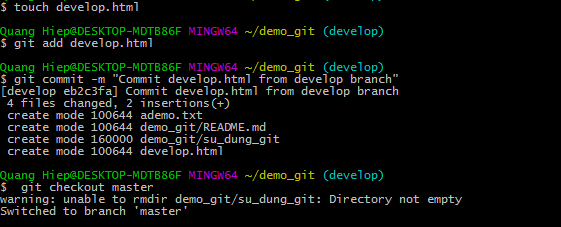
-Bây giờ nếu bạn muốn tạo một sự thay đổi nào đó mà có thể trực tiếp sử dụng trên working tree hoặc commit, push lên repository mà không ảnh hưởng đến branch master thì sẽ cần tạo ra một branch mới với tên là **develop**chẳng hạn. Và từ đó mỗi khi bạn thực hiện lệnh checkout vào branch nào đó thì toàn bộ mã nguồn trên working tree của bạn sẽ được đổi sang môi trường dành cho branch đang checkout.

- Giờ mình lấy một ví dụ đơn giản, bên branch**master** bạn tạo một tập tin **master.html** rồi commit lên, lúc này khi bạn qua cái branch **develop** (đã được tạo trước đó) thì cái tập tin **master.html** kia sẽ không có vì nó đã được commit bởi branch **master**, không liên quan gì tới **develop**. Tương tự, các thay đổi của bạn bên **develop** cũng sẽ không ảnh hưởng gì tới bên **master**cả. Điều này có một cái thú vị là nó tương tác trực tiếp trên máy tính của bạn, ví dụ tập tin index.html của **master** có nội dung khác và index.html bên **develop** thì khi bạn dùng lệnh checkout là nội dung trên máy tính nó tự đổi tương ứng, đó là lý do bạn có thể test nhiều phiên bản trên máy tính mà không cần đổi thư mục, chỉ cần checkout cái branch.

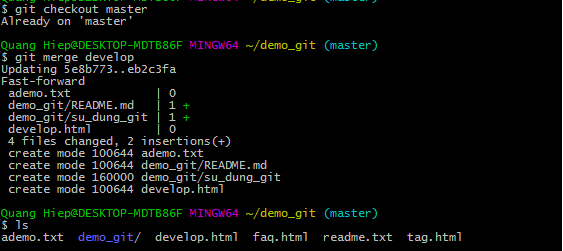
**II. Cách tạo một branch**

- Trước tiên bạn có thể xem toàn bộ các branch mà bạn đang có trong working tree bằng lệnh**git** **branch**. Sau đó nếu muốn tạo thêm branch, chỉ cần gõ lệnh **git branch tên\_brand**. Ví dụ mình cần tạo branch **develop.**

**III. Check out một branch**

- Checkout ở đây nghĩa là bạn truy cập kiểm tra mã nguồn trong branch đó để làm việc đấy. Để làm việc này, bạn sử dụng lệnh **git checkout tên\_branch.** **-** Lúc này bạn đã đổi sang branch **develop** rồi, bây giờ bạn sẽ làm việc ở branch mới chuyển. Để chuyển về branch chính thì bạn dung lệnh: **git checkout master**  
 - Bây giờ bạn thử tạo một tập tin nào đó, sau đó commit ở branch **develop** rồi chuyển về branch master sẽ thấy những gì bạn đã làm ở branch **develop**hoàn toàn vô nghĩa ở **master.** Dưới đây là ví dụ về việc ở branch **master**không có tập tin develop.html được tạo ra từ branch **develop.  
**

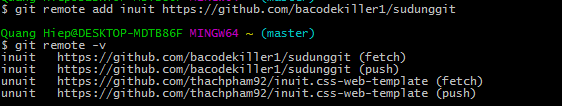
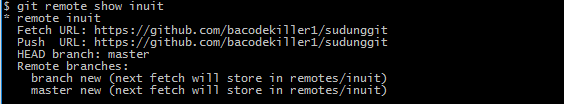
**IV.Gập dữ liệu từ một branch**

- Nếu mỗi branch nó nằm riêng như vậy thì bạn muốn sử dụng các thay đổi ở một branch nào đó cho master thì sao? À, chúng ta có thể sử dụng lệnh **git merge** để chuyển dữ liệu từ một branch nào đó về branch mà bạn đang trỏ đến. Lưu ý là ở branch cần chuyển về đã phải được commit. Ví dụ mình cần chuyển dữ liệu từ **branch develop** về **master** thì sẽ làm lần lượt các lệnh sau:****

**V. Xóa branch**

- Nếu bạn không cần dùng tới branch nào nữa thì có thể xóa với lệnh **git branch -d** **tên\_branch**. **Lưu ý** là cái branch muốn nóphải được gộp dữ liệu (merge) về master.   
 - Sau khi xóa xong, nó sẽ báo branch đó đã được móc vào commit với mã checksum nào (eb2c3fa).

**VI. Làm việc với Remote Branch**

- Quay lại một xíu với bài remote repository, bây giờ bạn hãy tạo thêm một remote mới từ địa chỉ **https://github.com/bacodekiller1/sudunggit**  và đặt tên cho remote này là **inuit** vào working tree của bạn.  
  
 - Bây giờ bạn có thể xem toàn bộ branch của cái remote **inuit** mới thêm vào bằng lệnh **git remote show inuit.  
** - Bây giờ bạn có thể chọn một cái remote branche cần fetch dữ liệu về. Và bạn có thể commit nó và thử push nó lên repository của bạn.

# Bài 8: Chia sẻ mẹo nhỏ dùng git bash

**1. Lệnh history để xem lại lịch sử các lệnh bạn đã gõ trên git bash.  
2. Lệnh clear để xóa sạch màn hình trở lại trạng thái ban đầu.  
3. Khi bạn gõ sai 1 lệnh hay 1 chữ cái trong lệnh mà muốn sửa lại lệnh mà không phải gõ hết toàn bộ lại lệnh thì bạn ấn dấu trên bàn phím sau đó sửa lệnh.  
4. Khi bạn muốn vào hoặc chọn một tập tin nào đó mà dài hoặc là bạn chỉ nhớ vài kí tự đầu bạn có thể gõ 1 vài kí tự đầu rồi ấn nút Tab nó sẽ hiện ra tập tin đó cho bạn.  
5. Khi bạn vào đường dẫn thư mục mà muốn trở lại thư mục trước bạn hãy dùng lệnh cd..  
6. Dùng lệnh mkdir tên \_thư\_mục để tạo 1 folder.  
7. Nếu muốn tạo thư mục có khoảng trống ở giữa bạn dùng lệnh: mkdir tên\ thư \mục  
8. Đổi tên thư mục bạn dùng lệnh: mv tên\_cũ tên\_mới.**