# Cơ bản về Git?

# Đề mục



# Chi tiết

## 1. Thế nào là `repository`, `branch`.

### `Repository` là gì:

Git là một công cụ quản lý mã nguồn, toàn bộ mã nguồn sẽ được lưu trữ trên một kho lưu trữ là Repository.

- Repository là một kho chứa tất cả những thông tin cần thiết để duy trì và quản lý các sửa đổi và lịch sử của toàn bộ project.
- Repository có hai loại là Local Repository (Kho chứa trên máy cá nhân) và Remote Repository (Kho chứa trên một máy chủ từ xa).

### `Branch` là gì:

- Branch là cái dùng để phân nhánh và ghi lại luồng của lịch sử. Branch đã phân nhánh sẽ

không ảnh hưởng đến branch khác nên có thể tiến hành nhiều thay đổi đồng thời trong cùng 1 repository
- Khi tạo mới một repository thì nhánh mặc định sẽ "master". Nhánh master thông thường là nhánh chính của ứng dụng.
- Branch đã phân nhánh có thể chỉnh sửa tổng hợp lại thành 1 branch bằng việc hợp lại (merge) với branch khác. Ví dụ thử nghiệm một tính năng mới và muốn không ảnh hưởng đến code chính có thể tạo một nhánh mới và sau khi xong sẽ hợp nhất lại với nhánh master.

# 2. Làm thế nào để xoá một `branch` ở phía local, một `branch` remote

## Để xóa `branch` ở phía local:

```
$ git branch -d <ten_branch>
```

- Đây là cách xóa an toàn, nếu branch cần xóa đã được merge vào một branch nào đó, hoặc khi tạo pull request vào một Remote Repository khác và đã được merge, thì có thể xóa branch bằng cách này. Nếu chưa được merge thì sẽ hiện thông báo lỗi là chưa được merge vào branch nào.

```
$ git branch -D <ten_branch>
```

Đây là hard delete, sẽ xóa branch bất kể branch đã được merge hay chưa.

> Cả 2 cách trên có thể xóa một lúc nhiều brach bằng cách thêm cách thêm nhiều tên branch ở cuối lệnh và cách nhau bằng một khoảng trắng.

## Để xóa một `branch` remote:

```
$ git push --delete <ten_remote> <ten_branch>
```

hoặc

```
```

```
$ git push <ten_remote> --delete <ten_branch>
```

> Tương tự như ở local, ta cũng có thể xóa một lúc nhiều branch.

# 3. Push branch với tên khác

Bình thường, chúng ta thường push lên branch giống với tên branch ở local. Tuy nhiên, vì lý do gì đó mà muốn push branch với một tên khác, ta có thể làm như sau:

```
$ git push origin <local_branch>:<remote_branch>
```

# 4. Thế nào là git `rebase`. Phân biệt `rebase` với `merge`

## Git `rebase` là gì?

Git rebase là tích hợp các thay đổi từ nhánh này vào nhánh khác. Trong git có 2 cách để thực hiện công việc này, đó là `git rebase` và `git merge`. Chúng ta cùng tìm hiểu rõ hơn về sự khác nhau giữa 2 cách này.

## Phân biệt `rebase` với `merge`

1. `git rebase` Trước tiên bạn chuyển sang branch đang làm việc

```
$ git checkout <working_branch>
```

Thực hiện tích hợp branch `<rebase_branch>` vào branch trên.

```
$ git rebase <rebase_branch>
```

Nó thực hiện bằng cách đi tới commit cha chung của hai nhánh (nhánh bạn đang làm việc `<working_branch>` và nhánh đang muốn rebase `<rebase_branch>`), tìm sự khác biệt trong mỗi

commit của nhánh mà bạn đang làm việc, lưu lại các thay đổi đó vào một tập tin tạm thời, khôi phục lại nhánh hiện tại về cùng một commit với nhánh bạn đang rebase, và cuối cùng áp dụng lần lượt các thay đổi. 2. `git merge`

```
$ git checkout <working_branch>
$ git merge <merge_branch>
```

- git dùng 2 bản commit cuối cùng của từng nhánh rồi tích hợp lại với nhau tạo thành 1 commit mới theo kiểu hình thoi. Thực hiện `merge` thì các commit đã tồn tại không bị thay đổi, chỉ tạo ra 1 commit mới tích hợp của 2 commit mới nhất.
- Đặc điểm: các commit của 2 nhánh được sắp xếp theo thời gian tạo commit.

> - Bạn sử dụng `git rebase` nếu như bạn muốn các sự thay đổi thuộc về branch của bạn luôn luôn là mới nhất. Và bạn có thể log một cách có hệ thống, dễ nhìn dễ theo dõi sau này.
> - Bạn sử dụng `git merge` nếu bạn muốn sắp xếp các commit theo mặc định. Bạn không biết về những gì mình làm gì trên branch đó thì dùng `merge` cho đảm bảo, việc theo dõi sau này có thể tốn nhiều thời gian.

# 5. Thế nào là `git stash`

Hãy tưởng tượng rằng bạn đang làm việc trên một tính năng ở trong một dự án phần mềm được quản lý bởi Git. Bạn đang ở ngay giữa chừng trong việc tạo ra một số thay đổi thì bạn nhận được một yêu cầu khắc phục một lỗi nghiêm trọng. Để bắt đầu giải quyết vấn đề, bạn cần một branch mới và một thư mục rỗng. Tuy nhiên, để chuyển sang nhánh mới bạn cần commit những phần dang dở ở trên, điều này có thể tạo ra một commit thừa. Trong trường hợp này, bạn nên nghĩ tới `git stash`, nó sẽ lưu lại trạng thái chưa được commit, và bạn có thể thoải mái chuyển sang branch khác để làm việc

## Lưu lại thay đổi

```
$ git stash save # hoặc "git stash"
```

Toàn bộ nội dung công việc đang làm dở đã được lưu lại, bạn có thể sử dụng `git stash` bao nhiêu lần tùy thích.

# Lấy lại thay đổi

- Bạn đã lưu lại các thay đổi vào `stash`, bây giờ bạn có thể xem lại danh sách các lần lưu thay đổi bằng lệnh sau

```
$ git stash list
stash@{0}: WIP on <branch_name>: <lastest commit>
stash@{1}: WIP on <branch_name>: <lastest commit>
```

- Nếu muốn xem cả nội dung của từng thay đổi thì thêm option `-p`

```
$ git stash list -p
```

-

- hoặc xem nội dung cụ thể hơn nữa của lần thay đổi thứ 1:

```
$ git stash show stash@{1}
```

-

- Để tiếp tục làm việc tại `stash` số 1, bạn có thể dùng lệnh sau

```
$ git stash apply stash@{1}
```

-

# Xoá các thay đổi không cần thiết

Đôi khi bạn muốn lấy lại thay đổi và xoá nội dung thay đổi lưu trong stack đi, khi đó bạn có thể

```
$ git stash apply stash@{1}
$ git stash drop stash@{1}
```

-

hoặc đơn giản hơn là

```
$ git stash pop stash@{1}
```

-

Thậm chí nếu muốn xoá toàn bộ stack thì có thể dùng clear

```
```

```
$ git stash clear
```

-

# 6. Làm thế nào xoá bỏ trạng thái của một vài commit gần đây

Có 2 cách để thực hiện công việc này:

Cách 1: Sử dụng git revert

```
$ git revert <commit_id>
```

Lệnh này tạo commit đảo ngược commit có commit_id đã chọn, commit chỉ định bị xoá bỏ, các commit mới hơn vẫn được giữ nguyên. Cách 2: Sử dụng git reset --hard

```
$ git reset --hard <commit_id>
```

Lệnh này sẽ xoá toàn bộ các commit trước đó và đưa branch về trạng thái của commit có commit_id đã chọn.

# 7. Làm thế nào để gộp một vài commit thành 1 commit duy nhất?

Để gộp nhiều commit thành 1 commit duy nhất, ta có thể sử dụng câu lệnh:

```
$ git rebase -i <id_commit_end>
```

hoặc

```
$ git rebase -i HEAD~<index>
```

`<id_commit_end>` là id của commit cuối trong nhóm cần gộp. `<index>`: số commit cần gộp. Ta có các lựa chọn pick|squash|fixup các commit trước khi save.

# 8. Phân biệt `git reset`, `git reset --hard`, `git reset --soft`

## `git reset`

```
$ git reset <commit_id>
```

Di chuyển HEAD về vị trí commmit reset và vẫn giữ nguyên tất cả các thay đổi của file, nhưng loại bỏ các thay đổi khỏi stage.

## `git reset --hard`

```
$ git reset --hard <commit_id>
```

Di chuyển con trỏ HEAD về vị trí commmit reset và loại bỏ tất cả sự thay đổi của file sau thời điểm commit reset.

## `git reset --soft`

```
$ git reset --soft <commit_id>
```

Lệnh này chỉ di chuyển HEAD về vị trí commit. Trạng thái của stage và tất cả sự thay đổi của file sẽ được giữ nguyên.

---

Revision #1
Created Thu, Aug 18, 2022 10:21 AM by Tình Leo
Updated Sun, Oct 5, 2025 2:12 PM by Tình Leo