

Giới thiệu về Git và một số tính năng cơ bản


Rikkeisoft Blog
Where the dream begins

March 7, 2016 by [Nguyen Duc Ha \(http://blog.rikkeisoft.com/author/nguyen-duc-ha/\)](http://blog.rikkeisoft.com/author/nguyen-duc-ha/)

Chủ đề: Giới thiệu về Git và một số tính năng cơ bản

Người thực hiện: Nguyễn Đức Hà

Slide



Seminar team iOS
Giới thiệu Git
và một số tính năng cơ bản

Người thực hiện:
Nguyễn Đức Hà

[Giới thiệu Git và một số tính năng cơ bản \(http://www.slideshare.net/hp23192/gii-thiu-git-v-mt-s-tnh-nng-c-bn\)](http://www.slideshare.net/hp23192/gii-thiu-git-v-mt-s-tnh-nng-c-bn)

1. Sơ lược về quản lý phiên bản

Khái niệm: Quản lý phiên bản là một hệ thống lưu trữ các thay đổi của một tập tin (file) hoặc tập hợp các tập tin theo thời gian. Quản lý phiên bản không chỉ được áp dụng trong phát triển phần mềm mà còn có thể áp dụng trong nhiều lĩnh vực khác.

Quản lý phiên bản cho phép:

- Xem lại lịch sử thay đổi của dự án.
- Xem thông tin của các thay đổi của dự án: tác giả, nội dung.
- Khôi phục lại phiên bản cũ.

Có 3 phương pháp quản lý phiên bản:

- Quản lý phiên bản cục bộ: người dùng sao chép các file và lưu trữ cục bộ đồng thời sử dụng một cơ sở dữ liệu đơn giản để quản lý lịch sử thay đổi của các file này. Đại diện tiêu biểu của phương pháp này là hệ thống quản lý phiên bản rcs.
- Quản lý phiên bản tập trung: hệ thống này gồm một máy chủ lưu trữ tất cả các tập tin đã được phiên bản hóa và danh sách các máy client được quyền sửa đổi các tập tin trên máy chủ. Đại diện của phương pháp này là SVN, Perforce.
- Quản lý phiên bản phân tán: trong phương pháp này, các máy khách không chỉ lấy về phiên bản mới nhất của các file mà còn sao chép toàn bộ repository. Với phương pháp này, dữ liệu ở mỗi máy khách đều có thể được dùng để phục hồi dữ liệu ở máy chủ khi gặp sự cố. Đại diện của phương pháp này là Git và Mercurial.

2. Khái niệm cơ bản về Git

Khái niệm

Git là một “Hệ thống quản lý phiên bản phân tán” được viết bởi Linus Torvalds. Git là phần mềm mã nguồn mở, có thể làm việc được trên nhiều hệ điều hành khác nhau: Windows, Linux, Mac OS X,...

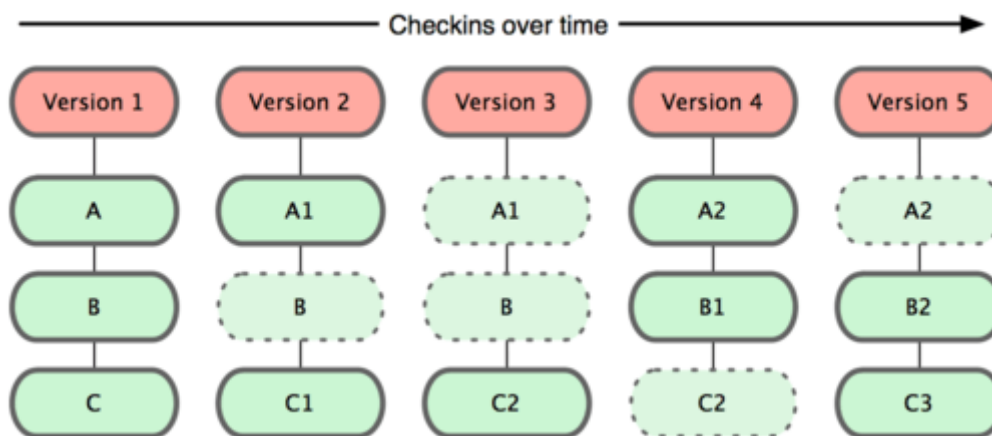
Đặc điểm của Git

- Cách xử lý dữ liệu:

Git coi dữ liệu của nó là một tập các ảnh (snapshot) của hệ thống tập tin. Điều này có nghĩa là mỗi phiên bản của dự án (có thể hiểu là một commit) sẽ là tập hợp của một số ảnh lưu lại nội dung của các tập tin của phiên bản đó.

Điều này mang đến nhiều tiện lợi cho việc theo dõi lịch sử, phục hồi dữ liệu và phân nhánh.

Cách Git xử lý dữ liệu có thể được mô tả như hình dưới:



(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_snapshot.png)

Cách tổ chức dữ liệu trong Git (Nguồn: <http://git-scm.com/book/v1/>)

- Thao tác với dữ liệu:

Hầu hết các thao tác với dữ liệu của Git có thể thực hiện cục bộ. Git thực hiện được việc này vì toàn bộ dữ liệu của dự án đều được lấy về và lưu trữ trên máy tính của người dùng.

Với tính năng này của Git, người dùng có thể làm việc trong nhiều trường hợp mà không nhất thiết phải có kết nối Internet. Điều này mang đến nhiều lợi thế cho Git so với các hệ thống quản lý dữ liệu khác.

- Tính toàn vẹn:

Các thay đổi trong Git được tham chiếu bằng một mã băm sử dụng cơ chế mã hóa SHA-1. Đồng thời, các thay đổi trong Git đều được thêm vào cơ sở dữ liệu do đó rất khó bị mất khi thay đổi và truyền tải dữ liệu. Với Git, người dùng có thể thoải mái thử nghiệm, lưu trữ mà không có ảnh hưởng đến dự án.

không sợ ảnh hưởng đến dự án.

Tổ chức dữ liệu trong Git

Các tập tin trong Git tồn tại ở một trong ba trạng thái: modified, staged, committed.

- Modified: tập tin được sửa nhưng chưa được đánh dấu để commit (nằm trong mục Unstaged files).
- Staged: tập tin được đánh dấu sẽ được commit (nằm trong mục Staged files).
- Committed: tập tin đã được commit và lưu trữ trong cơ sở dữ liệu.

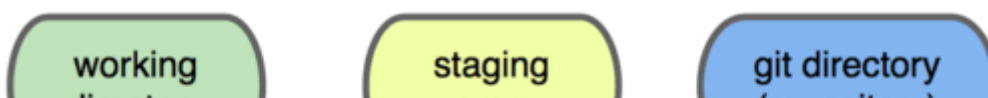
Một tập tin có thể vừa ở trạng thái modified vừa ở trạng thái staged hoặc committed. Điều này xảy ra khi người dùng chỉ staged một số dòng trong tập tin.

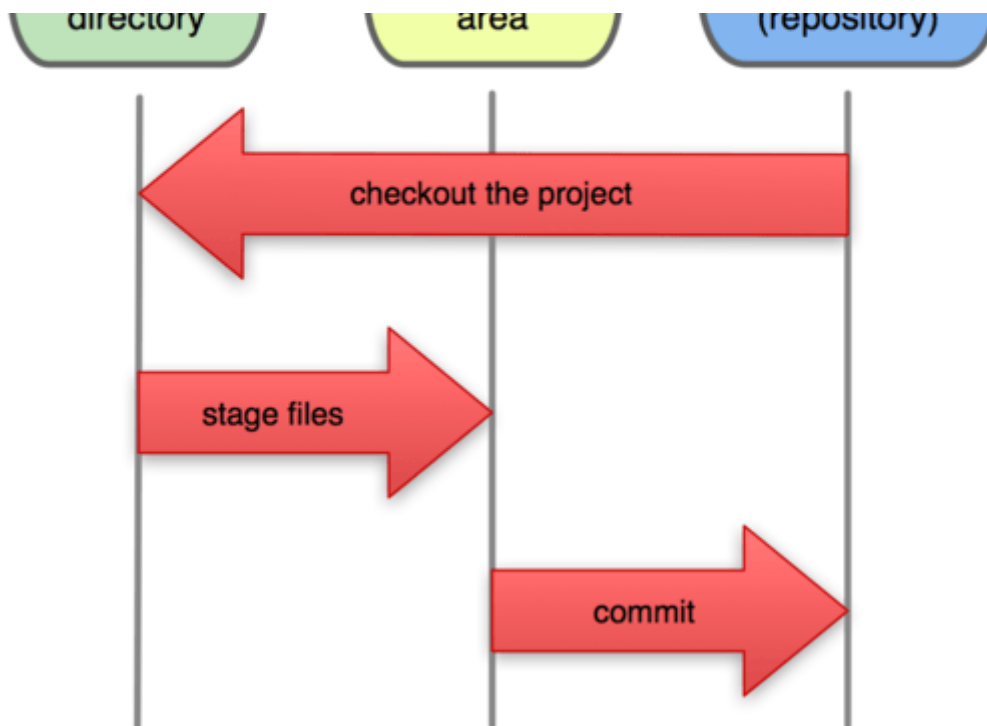
Ba trạng thái này tạo ra ba phần riêng biệt của dự án:

- Working directory: là bản sao của một phiên bản của dự án. Người dùng sẽ làm việc với các tập tin ở khu vực này. Mọi thay đổi của các tập tin sẽ được hiển thị ở đây.
- Staging area: là một tập tin trong thư mục Git. Tập tin này chứa thông tin về những thay đổi sẽ được commit.
- Git directory: đây là nơi Git lưu trữ các siêu dữ liệu (metadata) và cơ sở dữ liệu của toàn bộ dự án. Đây cũng là phần quan trọng nhất của Git, nó là một bản sao của dự án được sao chép (clone) từ repository trên server Git.

Mô hình làm việc và tổ chức của Git như trong hình dưới đây:

Local Operations





(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_local_area.png)

Mô hình tổ chức trong Git (Nguồn: <http://git-scm.com/book/v1/v1/>)

3. Làm việc với Git

Khởi tạo

- Khởi tạo repository từ thư mục cũ:

Để khởi tạo một repository từ thư mục cũ, cần thêm thư mục “.git” vào thư mục đó bằng cách chạy lệnh:

```
git init
```

- Sao chép một repository đã tồn tại:

Để sao chép từ một repository đã tồn tại cần sử dụng lệnh:

```
git clone [url] [folder name]
```

Trong đó:

url là đường dẫn đến repository. Đường dẫn này tùy thuộc vào giao thức truyền tải mà server cung cấp. Một số loại giao thức thường dùng:

- Giao thức git://
- Giao thức http hoặc https
- Giao thức SSH: user@server:/path.git

folder name là tên folder ở máy cục bộ mà repository sẽ được sao chép về. Tham số này là không bắt buộc, nếu không dùng tham số này thì repository sẽ được sao chép vào thư mục mặc định có tên như tên repository.

Lưu trữ

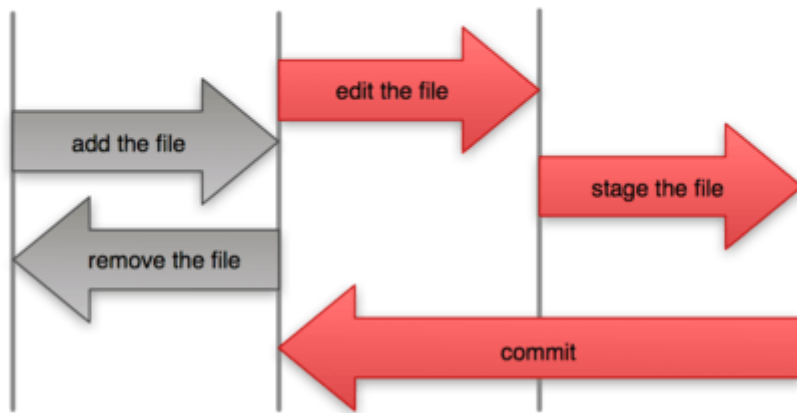
Trước khi tìm hiểu cách lưu trữ các tập tin trong thư mục Git, chúng ta cần hiểu về vòng đời của các tập tin này. Vòng đời của một tập tin gồm có 4 trạng thái: untracked, unmodified, modified, staged. Sơ đồ vòng đời của một tập tin được mô tả trong hình dưới đây.

- untracked: là trạng thái của tập tin mà không có trong bất kỳ ảnh nào trước đó.
- unmodified: là trạng thái của tập tin chưa được sửa gì so với ảnh cuối cùng.
- modified: là trạng thái của tập tin đã được sửa so với ảnh cuối cùng.
- staged: là trạng thái của tập tin đã được đánh dấu và chuyển vào staging area.

Vòng đời của tập tin trong Git được mô tả trong hình dưới:

File Status Lifecycle





(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_file_life_cycle.png)

Vòng đời của tập tin trong Git (Nguồn: <http://git-scm.com/book/v1/v1/>)

Bỏ qua các tập tin: việc này khá cần thiết trong dự án khi có một số loại tập tin mà người dùng không muốn theo dõi. Những tập tin này có thể là các tập tin được tạo ra tự động, các tập tin tạm thời hay các tập tin được tạo ra khi biên dịch. Để bỏ qua các tập tin này có thể tạo ra một tập tin có tên là `.gitignore` và liệt kê tên các tập tin hoặc các mẫu tên tập tin muốn bỏ qua. Gitignore sử dụng filename glob pattern

([https://en.wikipedia.org/wiki/Glob_\(programming\)](https://en.wikipedia.org/wiki/Glob_(programming))) (tương tự như một regular expression đơn giản) để lọc các tập tin.

Ví dụ:

*.[abc] : bỏ qua các tập tin có đuôi là `.a` hoặc `.b` hoặc `.c`

*.~ : bỏ qua các tập tin có đuôi là `~`

*.[0-9] : bỏ qua các tập tin có đuôi nằm trong khoảng từ 0 đến 9

Phục hồi

Tại bất kì thời điểm nào, người dùng đều có thể phục hồi một phần của dự án về trạng thái trước đó. Phục hồi ở đây không phải là lấy lại dữ liệu đã mất mà nó được hiểu là đưa dữ liệu về trạng thái trước, trạng thái này dĩ nhiên đã được lưu lại trong cơ sở dữ liệu. Tuy nhiên, việc phục hồi dữ liệu khá nguy hiểm vì người dùng có thể bị mất dữ liệu nếu không thực hiện

phục hồi ưu tiên khả năng miễn vì người dùng có thể bị mất ưu tiên nếu không thực hiện đúng. Có một số công cụ để phục hồi các thay đổi tùy theo trường hợp áp dụng việc phục hồi này.

- Thay đổi commit cuối cùng: phương pháp này có thể hiểu một cách đơn giản là ghi nối vào commit cuối cùng. Trong lịch sử commit vẫn chỉ hiển thị một commit, chỉ có dữ liệu hoặc thông điệp của commit là thay đổi. Để thực hiện phương pháp này, sử dụng lệnh:

```
Git commit --amend
```

- Loại bỏ tập tin khỏi staging area: phương pháp này dùng để chuyển tập tin từ staging area sang working directory. Nội dung thay đổi của tập tin không bị mất đi mà nó chỉ không được commit. Để thực hiện phương pháp này có thể sử dụng lệnh:

```
Git reset HEAD [file name]
```

- Phục hồi tập tin đã thay đổi (trong working directory): khi muốn phục hồi nội dung tập tin về trạng thái trước khi thay đổi (ảnh cuối cùng trong cơ sở dữ liệu có chứa tập tin đó) có thể sử dụng phương pháp này. Câu lệnh được dùng:

```
Git checkout -- [file name]
```

- Loại bỏ commit, phục hồi về một commit trước đó (khi chưa đẩy lên repository từ xa):

```
Git reset --[option] [commit]
```

[Option] có thể là soft, mixed hoặc hard.

- Soft reset: giữ nguyên các tập tin ở working directory và staging area.
- Mixed reset: giữ nguyên các tập tin ở working directory, các tập tin ở staging area được thay đổi khớp với commit.
- Hard reset: các tập tin ở cả 2 khu vực trên đều bị thay đổi.
- Loại bỏ commit đã được đẩy lên repository từ xa:

```
Git revert [commit]
```


Git revert [COMMIT]

Thực chất của lệnh này là tạo một commit khác giống với commit được phục hồi về. Đây là một cách an toàn để phục hồi các commit đã được công khai vì nó không sửa lịch sử commit.

Tổng kết lại, việc phục hồi thay đổi trong Git có thể được tóm tắt trong bảng sau:

Lệnh	Đối tượng tác động	Ý nghĩa
Git reset	Commit – level	Xóa các commit ở nhánh riêng tư hoặc phục hồi thay đổi chưa commit
Git reset	File – level	Loại bỏ tập tin khỏi khu vực staging area
Git checkout	Commit – level	Chuyển nhánh
Git checkout	File – level	Phục hồi thay đổi của tập tin trong working directory
Git revert	Commit – level	Phục hồi commit ở nhánh công khai

4. Làm việc với remote repository

Một trong những yêu cầu đối với Git đó là khả năng hợp tác với các thành viên khác trong dự án. Để thực hiện được việc này, Git cung cấp các remote repository, đây là các phiên bản của dự án, được lưu trữ trên server và người dùng có quyền truy cập vào đó. Một dự án có thể có nhiều repository, tuy nhiên nó luôn có một repository chính là repository mà người dùng tạo

một repository, tuy nhiên nó luôn có một repository chính, là repository mà người dùng sau chép về. Repository chính của dự án có tên mặc định là origin.

Để thêm các repository, có thể sử dụng lệnh:

```
Git remote add [repository name] [repository url]
```

Repository name là tên đại diện của repository, nó được sử dụng trong tất cả các câu lệnh thao tác với repository đó.

Để lấy dữ liệu về từ repository, có thể sử dụng một trong hai lệnh:

```
Git fetch [repository name]
```

```
Git pull [repository name]
```

Lệnh fetch chỉ lấy dữ liệu từ repository về mà không tích hợp vào dữ liệu của repository cục bộ trong khi lệnh pull sẽ thực hiện tích hợp với máy cục bộ luôn.

Để đẩy dữ liệu cục bộ lên máy chủ, có thể sử dụng lệnh:

```
Git push [repository name] [branch name]
```

Để xóa một repository, sử dụng lệnh:

```
Git remote rm [repository name]
```

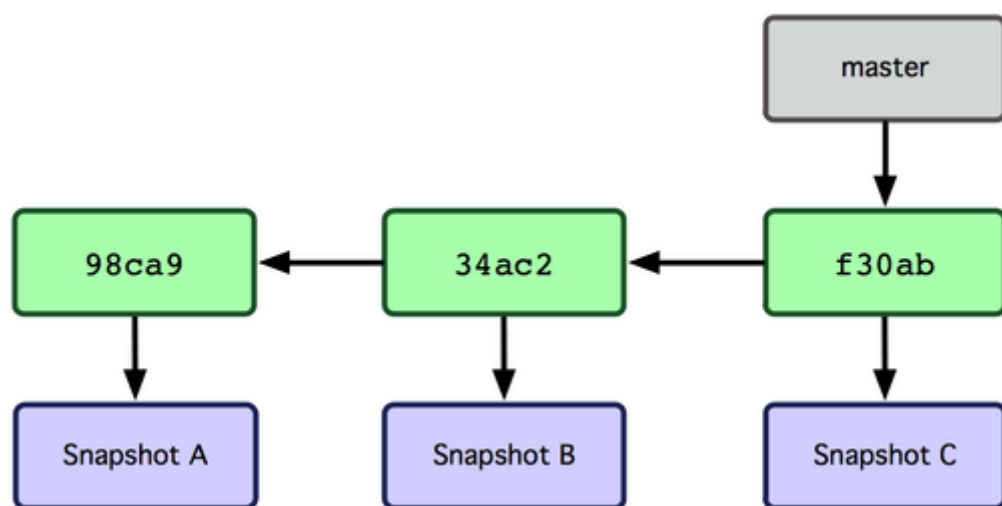
5. Git branch

Khái niệm nhánh

Trước khi tìm hiểu khái niệm nhánh trong Git, chúng ta nhắc lại cách Git lưu trữ dữ liệu. Như đã nói trong phần trước, Git lưu dữ liệu dưới dạng một chuỗi các ảnh. Vậy làm thế nào để Git có thể liên kết được chuỗi các ảnh này và biết được quan hệ giữa chúng? Git sử dụng đối tượng commit. Đối tượng này chứa một con trỏ tới ảnh đã lưu và 0 hoặc nhiều con trỏ tới

tuýng commit, đoi tuýng này chứa một con trỏ tới định địa iau và 0 hoặc nhiều con trỏ tới commit cha của nó. Một commit có thể không có cha (commit gốc), có một cha (commit thông thường) hoặc có nhiều cha (commit được tích hợp từ nhiều nhánh).

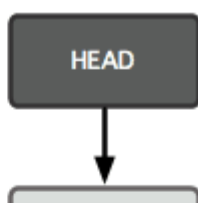
Như vậy chúng ta có thể định nghĩa nhánh như sau: nhánh là một con trỏ có khả năng di chuyển được, trỏ đến một trong những commit của repository.

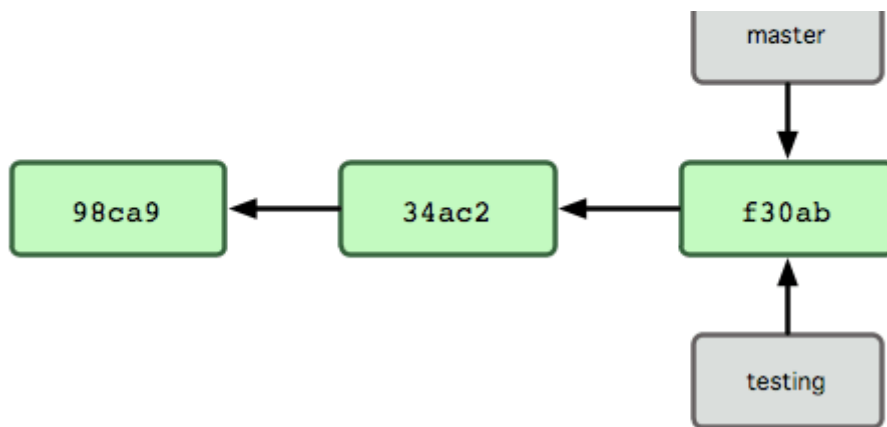


(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_master_branch.png)

Nhánh mặc định trong Git (Nguồn: <http://git-scm.com/book/v1/v1/>)

Tên nhánh mặc định là master, người dùng có thể thêm nhiều nhánh khác. Khi tạo một nhánh mới, một con trỏ mới sẽ được tạo ra, trỏ vào commit hiện tại. Để có thể biết được nhánh nào đang được sử dụng, Git sử dụng một con trỏ đặc biệt gọi là HEAD, con trỏ này trỏ vào nhánh đang được sử dụng để làm việc. Do đó việc chuyển nhánh thực chất là cho con trỏ HEAD trỏ sang nhánh khác.





(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_new_branch.png)

Lịch sử commit sau khi tạo nhánh mới (Nguồn: <http://git-scm.com/book/v1/v1/>)

Quản lý nhánh (tạo, xóa, chuyển nhánh)

Tạo nhánh mới: sử dụng lệnh

```
Git branch [branch name]
```

Chuyển sang một nhánh đã có sử dụng lệnh:

```
Git checkout [branch name]
```

Ngoài ra, có thể kết hợp hai lệnh trên để tạo một nhánh mới và chuyển luôn sang nhánh đó:

```
Git checkout -b [branch name]
```

Để xóa một nhánh, có thể sử dụng lệnh:

```
Git branch -d [branch name]
```

Khi chuyển nhánh, nhằm tránh xung đột giữa các nhánh, Git thường yêu cầu người dùng làm “sạch” working directory và staging area trước. Làm sạch ở đây được hiểu là không có bất kỳ thay đổi nào chưa được commit. Để làm sạch working directory và staging area có 2 cách:

- Commit tất cả các thay đổi.
- Lưu thay đổi lại mà không cần commit. Để thực hiện cách này, sử dụng lệnh:

```
Git stash
```

Ngược lại, nếu muốn khôi phục những thay đổi này để tiếp tục làm việc, có thể sử dụng lệnh:

```
Git stash apply
```

Hoặc:

```
Git stash pop
```

Với “apply”, stash sau khi được khôi phục thì vẫn còn trong cơ sở dữ liệu, ngược lại, với “pop” thì stash sẽ bị xóa ngay sau khi khôi phục.

Quy trình làm việc với nhánh: nhánh lâu đời (Long-Running Branches) và nhánh chủ đề (Topic Branches)

Khi làm việc với nhánh, thường có hai cách phổ biến là tạo nhánh lâu đời và tạo nhánh theo chủ đề.

Nhánh lâu đời (Long-Running Branches): đây là một cách làm việc với nhánh khá phổ biến. Mã nguồn ổn định được chứa ở nhánh master hoặc ở các nhánh theo phiên bản (ví dụ 1.0, 2.0) đối với dự án có nhiều phiên bản. Đồng thời với nhánh ổn định là các nhánh song song, công việc được thực hiện ở trên các nhánh này cho đến khi đạt trạng thái ổn định sẽ được tích hợp vào nhánh master. Trạng thái ổn định thường là khi hoàn thành một chức năng và đã kiểm thử xong.

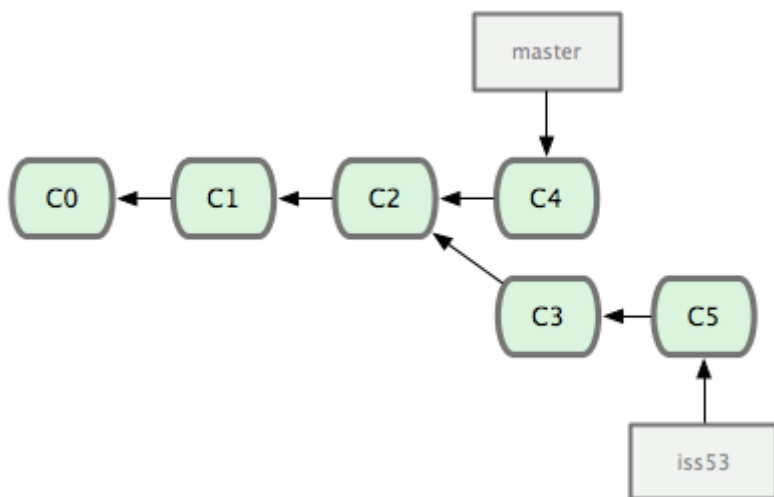
Nhánh chủ đề (Topic Branches): nhánh chủ đề là các nhánh có vòng đời ngắn, thường được tạo ra để phát triển một tính năng nào đấy và xóa đi khi hoàn thành công việc.

Tích hợp nhánh: Merge và Rebase

Tích hợp nhánh: Merge và Rebase

Để tích hợp một nhánh vào một nhánh khác, người dùng Git có hai lựa chọn: merging và rebasing. Hai cách này có cùng một mục đích, cho kết quả giống nhau nhưng cách làm lại khác nhau hoàn toàn.

Lịch sử trước khi tích hợp:



(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_before_merge.png)

Lịch sử commit trước khi tích hợp (Nguồn: <http://git-scm.com/book/v1/v1/>)

- Merge:

Để thực hiện việc tích hợp 2 nhánh sử dụng lệnh merge như sau:

```
Git checkout [destination branch]
```

```
Git merge
```

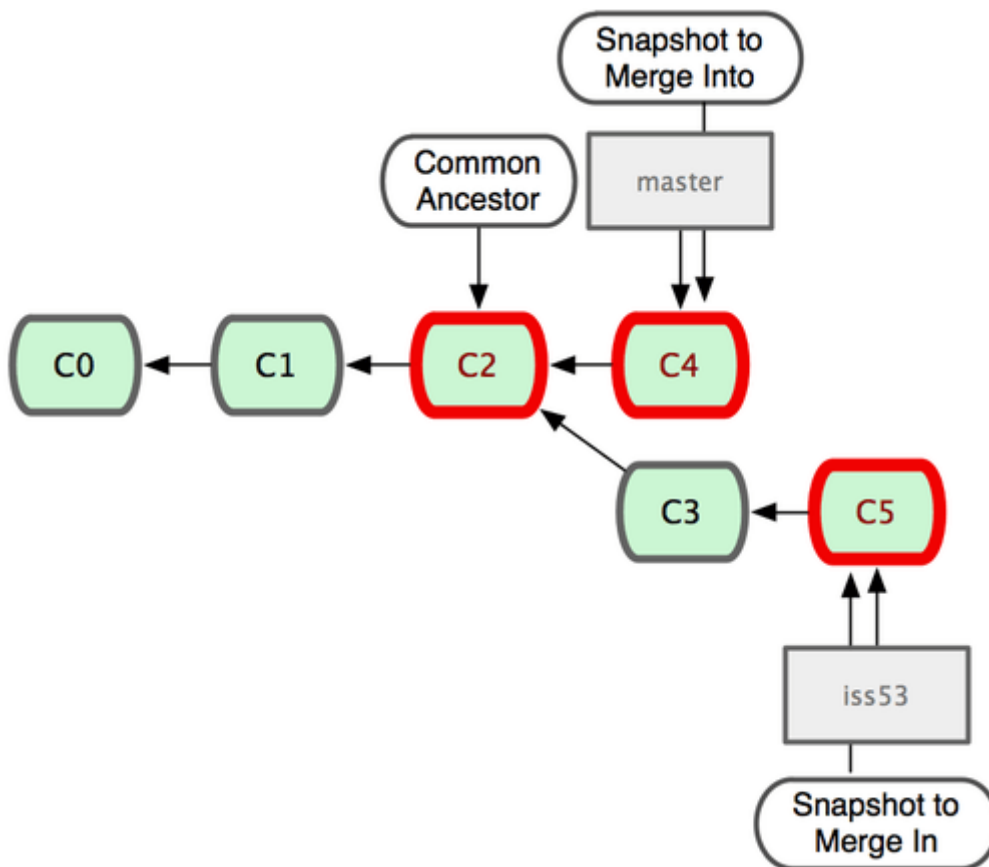
Hoặc có thể dùng câu lệnh rút gọn như sau:

```
Git merge [destination branch]
```

Cơ chế tích hợp của lệnh merge: chúng ta chỉ xét trường hợp 2 nhánh cần tích hợp song song với nhau và có một cha chung. Trong trường hợp này, Git thực hiện một tích hợp 3 chiều, gồm

có: cha chung của 2 nhánh và 2 ảnh được trỏ tới bởi 2 commit ở đầu mút của 2 nhánh. Git tạo một ảnh mới, là kết quả từ việc tích hợp 3 chiều này, đồng thời tạo ra một commit trỏ tới ảnh đó, commit này được gọi là merge-commit. Các commit được tích hợp từ nhánh nguồn vẫn được lưu trữ lại.

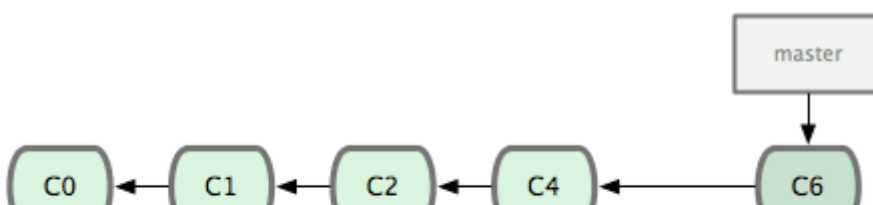
Quá trình tích hợp sử dụng merge:

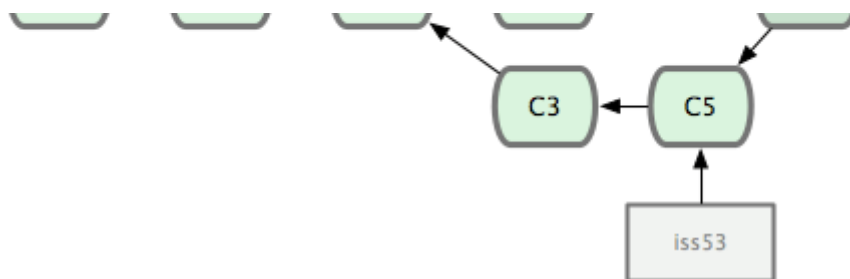


(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_merge_process.png)

Quá trình tích hợp trong Git (Nguồn: <http://git-scm.com/book/v1/v1/>)

Kết quả của merge:





(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_merge_result.png)

Kết quả tích hợp (Nguồn: <http://git-scm.com/book/v1/>)

- Rebase:

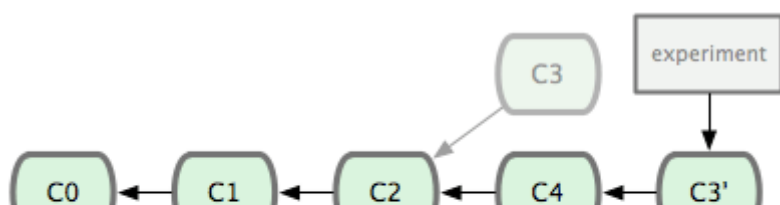
Để tích hợp 2 nhánh sử dụng rebase có thể làm như sau:

Git checkout

Git rebase [destination branch]

Cơ chế hoạt động của rebase: Git tìm commit là cha chung của 2 nhánh, tìm sự khác biệt trong từng commit của nhánh đang làm việc (source branch), lưu nó vào một tập tin tạm thời và khôi phục nhánh đang làm việc về cùng commit với nhánh đang rebase. Cuối cùng các thay đổi được lưu lại tạm thời vừa rồi sẽ được áp dụng lần lượt vào nhánh đang làm việc.

Lịch sử commit sau khi rebase:

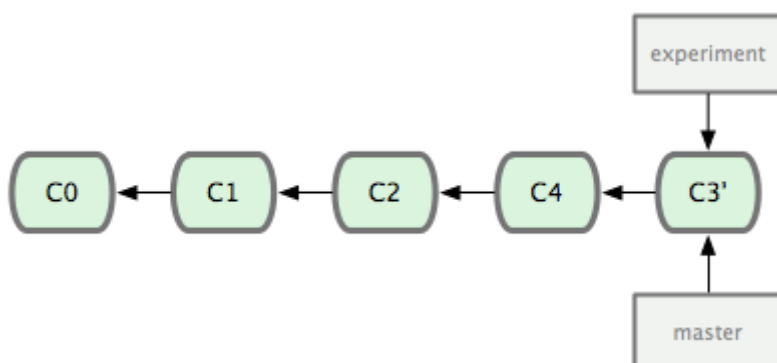




(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_rebase_before_fast-forward.png)

Lịch sử commit sau khi rebase, trước khi fast-forward

(Nguồn: <http://git-scm.com/book/v1/v1/>)



(http://blog.rikkeisoft.com/wp-content/uploads/2016/03/git_rebase_after_fast-forward.png)

Lịch sử commit sau khi rebase và fast-forward (Nguồn: <http://git-scm.com/book/v1/v1/>)

Trong cả 2 trường hợp rebase và merge, ảnh được trở tới bởi commit cuối cùng đều giống nhau. Tuy nhiên lịch sử commit của 2 cách không giống nhau. Lịch sử của rebase là một đường thẳng trong khi lịch sử của merge là 2 hoặc nhiều đường thẳng song song và được gộp lại ở commit cuối.

Lưu ý khi dùng rebase: không được dùng rebase với các commit đã được đẩy lên repository công khai. Nếu thực hiện rebase với các commit này có thể gây ra rối loạn trong việc tích hợp các commit từ nhánh từ xa.

6. Sử dụng submodules trong git

Submodules được sử dụng khi cần sử dụng một dự án khác trong dự án đang làm việc. Dự án cần dùng có thể là một thư viện hoặc một dự án riêng biệt. Các dự án này được coi là

an bản đang có thể là một thư viện hoặc một dự án riêng biệt. Các dự án này được coi là riêng biệt với nhau nhưng vẫn có thể sử dụng được các chức năng của nhau.

Để thêm một submodule, sử dụng câu lệnh:

```
Git submodule add [url]
```

Submodule được mặc định thêm vào thư mục có tên giống tên của repository. File cấu hình submodule có tên là .gitmodules.

Sao chép một dự án đã có Submodules:

- Sao chép dự án sử dụng lệnh git clone
- Truy cập vào thư mục submodule của dự án.
- Khởi tạo tập tin cấu hình submodule sử dụng lệnh “git submodule init”
- Lấy toàn bộ dữ liệu của submodule về sử dụng lệnh “git submodule update”

Với submodule, người dùng có thể thực hiện các lệnh lấy dữ liệu, tích hợp, commit và đẩy dữ liệu lên repository trên server tương tự như với dự án chính.

Để tìm hiểu sâu hơn về cách sử dụng submodule, các vấn đề gặp phải khi dùng submodule, có thể tham khảo tài liệu của git trên: <https://git-scm.com/book/en/v2/Git-Tools-Submodules> (<https://git-scm.com/book/en/v2/Git-Tools-Submodules>)

7. Các tài liệu tham khảo

<https://git-scm.com/> (<https://git-scm.com/>)

<https://www.atlassian.com/git/tutorials/> (<https://www.atlassian.com/git/tutorials/>)

<http://stackoverflow.com/questions/tagged/git> (<http://stackoverflow.com/questions/tagged/git>)

5 Comments

Rikkeisoft Blog

 Nguyễn Văn Dực ▾

 Recommend

 Share

Sort by Best ▾

Rikkeisoft Blog requires you to verify your email address before posting. Send verification email to nguyenvanduoc071994@gmail.com



Join the discussion...



Oanh Nguyen Ngoc Mod • 2 years ago

> The .gitignore (and other) files use filename globs, not regular expressions.

File .gitignore nó sử dụng glob pattern để lọc filename (sử dụng ký tự wildcard) chứ không phải dùng biểu thức chính quy nhé Hà.

[https://en.wikipedia.org/wiki/Glob_\(programming\)](https://en.wikipedia.org/wiki/Glob_(programming))

3 ^ | v • Reply • Share >



Ha Nguyen Duc ➔ Oanh Nguyen Ngoc • 2 years ago

Cảm ơn anh Oanh. Em đã sửa lại rồi.

^ | v • Reply • Share >



Huy Nguyen Quang Mod ➔ Oanh Nguyen Ngoc • 2 years ago

Chắc là đọc lướt nên nhìn nhầm rồi anh ạ :D

^ | v • Reply • Share >



Ánh Kiếu • a year ago

Em chào anh ạ. Anh cho em hỏi lệnh diff trong Git với ạ.

^ | v • Reply • Share >



Huy Nguyen Quang Mod ➔ Ánh Kiếu • a year ago

Approve![([https://link.nylas.com/open...2-e5f9?](https://link.nylas.com/open...2-e5f9?r=bm90aWZ5LTE5QzE5RDU0LTdBRkQtMTFFNi1CMkU5LTAwMjU5MDg1MzA4MEBkaXNxdXMu)

r=bm90aWZ5LTE5QzE5RDU0LTdBRkQtMTFFNi1CMkU5LTAwMjU5MDg1MzA4MEBkaXNxdXMu
mV0)

^ | v • Reply • Share >

ALSO ON RIKKEISOFT BLOG

[Seminar] Giới thiệu về Websocket và Node.js

8 comments • 3 years ago •

AvatarNguyễn Xuân Bình — Chào Anh/Chị, Cho em ví dụ về socket io rooms được không ạ!, chỉ những người trong rooms, mới nhận được message từ server. ...

Những Lưu Ý Trong Lập Trình Unity

1 comment • a year ago •

AvatarNguyễn Giáp — Bài viết rất hữu ích (y

How to make unit test with PHPUnit (p1)

3 comments • a year ago •

AvatarHuy Nguyen Quang — @anh Oanh: Với 1 số micro-framework như Slim, Lumen thì nên viết test cho phần nào hả anh? Giả sử em có cả Model, ...

Htaccess File Tutorial and Tips

1 comment • 3 years ago •

AvatarHuy Nguyen Quang — Cảm ơn anh Oanh nhé :D

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy

Copyright © 2017 · [Genesis Framework \(http://www.studiopress.com/\)](http://www.studiopress.com/) · [WordPress \(http://wordpress.org/\)](http://wordpress.org/)
· [Log in \(http://blog.rikkeisoft.com/wp-login.php\)](http://blog.rikkeisoft.com/wp-login.php)