Mục lục

Giới thiệu về Git

Git là một hệ thống quản lý mã nguồn phân phối và kiểm soát phiên bản phân tán với sự nhấn mạnh về tốc độ. Git lần đầu được thiết kế và phát triển bởi Linus Torvalds cho phát triển hạt nhân Linux. Nó là một phần mềm miễn phí được phân phối theo các điều khoản của GNU phiên bản 2.

Phần hướng dẫn này giải thích cách để sử dụng Git cho điều khiển phiên bản dự án trong môi trường phân tán trong khi làm việc về phát triển các ứng dụng dựa trên hoặc không dựa trên web.

Loạt bài hướng dẫn của chúng tôi dựa trên nguồn tài liệu của: **Tutorialspoint**

Đối với độc giả

Các phần hướng dẫn này sẽ giúp người mới bắt đầu tìm hiểu các chức năng cớ bản của hệ thống quản lý phiên bản Git. Sau khi hoàn thành các phần hướng dẫn này, bạn sẽ thấy trình độ của mình ở mức vừa phải trong việc sử dụng hệ thống kiểm soát phiên bản Git, và từ đó bạn có điều kiện để nâng cao trình độ chuyên môn.

Điều kiện tiền đề

Chúng tôi giả định rằng bạn đang sử dụng Git để xử lý tất cả các dự án về Java hoặc Non-Java. Vì vậy nó sẽ tốt nếu bạn đã tiếp xúc với vòng đời phát triển phần mềm và kiến thức về phát triển các ứng dụng dựa trên web hoặc không dựa trên web.

Cơ bản về Git

VCS – hệ thống quản lý phiên bản

Version Control System (VCS) là một phần mềm mà giúp các nhà phát triển phần mềm làm việc cùng nhau và duy trì một lịch sử đầy đủ các công việc mà họ đã làm.

Dưới đây là các chức năng của một VCS:

- Cho phép các nhà phát triển phần mềm cùng làm việc với nhau
- Không cho phép ghi đè lên các thay đổi của nhau
- Duy trì một lịch sử của mọi phiên bản.

Dưới đây là các loại VCS:

- Hệ thống kiểm soát phiên bản tập trung (CVCS).
- Hệ thống kiểm soát phiên bản phân phối/phân cấp (DVCS).

Trong chương này chúng ta sẽ chỉ tập trung vào hệ thống quản lý phiên bản phân phối và đặc biệt trên Git.

Hệ thống kiểm soát phiên bản phân phối

Hệ thống kiểm soát phiên bản tập trung (CVCS) sử dụng một máy chủ để lưu giữ tất cả các file và cho phép các team cộng tác với nhau. Nhưng nhược điểm lớn nhất của CVCS cũng là điểm thất bại của nó, tức là, sự thất bại của các máy chủ trung tâm. Thật không may là, nếu máy chủ trung tâm bị hỏng trong một giờ, thì trong suốt quãng thời gian đó không ai có thể cộng tác được với ai cả. Và ngay cả trong trường hợp xấu nhất, nếu đĩa của máy chủ trung tâm bị hỏng và sự sao lưu không được thực hiện, bạn sẽ mất toàn bộ lịch sử của dự án. Tại đây, hệ thống quản lý phiên bản phân phối xuất hiện.

Các client DVCS không chỉ kiểm tra được các ảnh chụp mới nhất của các thư mục mà họ còn quan sát được tất cả repository trữ của dự án. Nếu server bị hỏng, các kho dự trữ của các client có thể sao một bản sao đầy đủ cho server để khôi phục lại nó. Git không phụ thuộc vào server trung tâm và đó là lý do tại sao bạn có thể thực hiện nhiều thao tác khi bạn đang offline. Bạn có thể ủy thác các thay đổi, tạo các nhánh, xem các bản ghi và thực hiện các hoạt động khác khi bạn đang offline. Bạn cần kết nối mạng chỉ để công bố những thay đổi của bạn và đưa những thay đổi mới nhất vào dự án.

Các lợi thế của Git

Nguồn miễn phí và mở

Git được công bố dưới giấy phép nguồn mở của GPL. Nó có sẵn miễn phí trên mạng. Bạn có thể sử dụng Git để quản lý các dự án thích hợp mà không phải trả bất kỳ đồng nào. Như là một nguồn mở, bạn có thể tải mã nguồn của nó và cũng có thể thực hiện các thay đổi theo yêu cầu của bạn.

Tốc độ nhanh và nhỏ gọn

Khi hầu hết các thao tác được thực hiện trong nội bộ, nó mang lại lợi ích rất lớn về tốc độ. Git không phụ thuộc vào server, đó là lý do tại sao mà không cần sự tương tác với server từ xa cho mọi thao tác. Phần cốt lõi của Git được viết bằng C, mà có thể tránh được các chi phí liên quan đến thời gian chạy với ngôn ngữ bậc cao khác. Mặc dù Git phản ánh toàn bộ repository trữ, kích thước của các dữ liệu trên các client là nhỏ. Điều này cho thấy sự hiệu quả của Git trong việc nén và lưu trữ dữ liệu trên các client.

Dự phòng (sao lưu) ẩn

Việc mất dữ liệu là hiếm khi xảy ra khi mà có rất nhiều bản sao của nó. Dữ liệu hiện diện ở bất kỳ client nào, do đó nó có thể được sử dụng trong trường hợp hỏng hoặc ngừng ở server.

An toàn cao

Git sử dụng một hàm băm (hash funtion) mật mã chung được gọi là hàm băm an toàn (SHA1), để đặt tên và xác định các đối tượng trong cơ sở dữ liệu của nó. Mỗi tập tin và commit được kiểm tra tóm tắt và thu được kết quả tại thời gian kiểm tra. Điều này ngụ ý rằng, nó không thể thay đổi tập tin, ngày tháng và thông báo commit và bất kỳ dữ liệu khác từ cơ sở dữ liệu Git mà không hiểu biết về Git.

Không yêu cầu một phần cứng mạnh

Trong trường hợp CVCS, server trung tâm cần đủ mạnh để phục vụ các yêu cầu của toàn team. Đối với những team nhỏ, nó không phải là một vấn đề, nhưng khi kích thước team phát triển, thì những hạn chế của phần cứng server có thể làm hiệu suất của công việc thay đổi theo hướng tiêu cực. Trong trường hợp DVCS, các nhà phát triển không tương tác với server trừ khi họ cần công bố những thay đổi đã thực hiện. Tất cả các việc đều diễn ra trên các client, vì thế phần cứng của server có thể thực sự là vấn đề đơn giản.

Phân nhánh dễ dàng hơn

CVCS sử dụng kỹ thuật sao chép rẻ, nếu chúng ta tạo ra một nhánh mới, nó sẽ sao chép tất cả các code tới nhánh mới, vì thế nó tốn thời gian, không hiệu suất. Ngoài ta, việc xóa và sáp nhập của các nhánh trong CVCS là phức tạp và tốn thời gian. Nhưng quản lý nhánh với Git là rất đơn giản. Nó chỉ mất một vài giây để tạo, xoa, và nhập các nhánh.

Các thuật ngữ của DVCS

Kho commit nội bộ

Mỗi công cụ VCS cung cấp một nơi làm việc riêng như là bản sao làm việc. Các nhà phát triển đã thực hiện những thay đổi trong nơi làm việc riêng và sau khi commit của họ, những thay đổi trở thành một phần của kho. Git có một bước tiến xa hơn bằng cách cung cấp cho họ một bản sao riêng của toàn bộ kho. Người sử dụng có thể thực hiện nhiều thao tác với kho này như thêm, di chuyển, đổi tên file, commit thay đổi và nhiều thao tác khác.

Thư mục làm việc và Staging hoặc Index

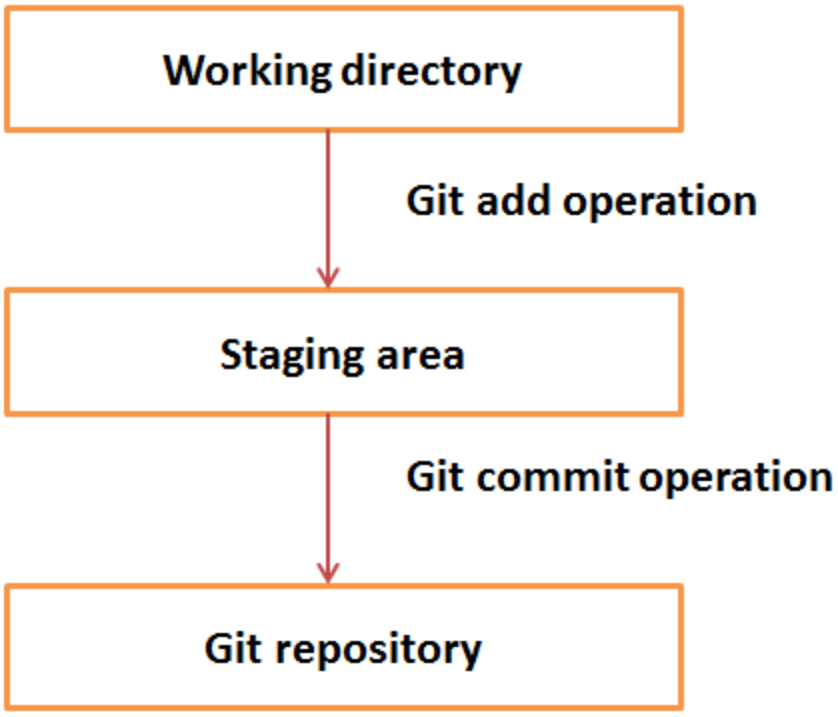
Thư mục làm việc là nơi các file được kiểm tra. Trong CVCS, các nhà phát triển thường tạo các thay đổi và commit các thay đổi của họ một cách trực tiếp tới kho chứa. Nhưng Git sử dụng một phương thức khác. Git không theo dõi từng file sửa đổi. Bất cứ khi nào bạn thự hiện commit một thao tác, Git tìm kiếm file trong khu vực tổ chức (staging area). Chỉ khi nào những file có mặt trong khu vực tổ chức này được xem xét để commit mà không phải tất cả các file sửa đổi.

Sau đây là tiến trình làm việc cơ bản của Git:

Bước 1 : Bạn sửa đổi một file từ thư mục làm việc

Bước 2 : Bạn thêm file đó vào khu vực tổ chức

Bước 3 : Bạn thực hiện các hoạt động commit mà di chuyển các file từ khu vực tổ chức. Sau thao tác đẩy (push), nó lưu các thay đổi cố định tới kho chứa Git.



Giả sử bạn sửa đổi 2 file, tên là sort.c và search.c và bạn muốn thực hiện hai commit khác nhau cho mỗi hoạt động. Bạn có thể thêm một file vào khu vực tổ chức và thực hiện commit. Sau hoạt động commit đầu tiên, làm lại theo phương thức tương tự cho file còn lại.

```
# First commit
[bash]$ git add sort.c

# adds file to the staging area
[bash]$ git commit –m “Added sort operation”

# Second commit
[bash]$ git add search.c

# adds file to the staging area
[bash]$ git commit –m “Added search operation”
```

Blobs

Blob là viết tắt của **B**inary **L**arge **Ob**ject. Mỗi phiên bản của một file được đại diện bởi blob. Một blob chứa dữ liệu file nhưng không chứa bất kỳ siêu dữ liệu nào về file. Nó là một tập tin nhị phân,

và trong cơ sở dữ liệu Git, nó được đặt trên là SHA1 hash của file đó. Trong Git, các file không được đặt bằng tên. Tất cả mọi thứ được đặt địa chỉ theo nội dung.

Cây - Trees

Cây (Tree) là một đối tượng, mà biểu diễn một thư mục. Nó giữ các blob cũng như các thư mục phụ khác. Một cây là một file nhị phân mà giữ các thứ liên quan đến blob và các cây cũng được đặt tên là SHA1 hash của đối tượng cây.

Ký thác - Commits

Hoạt động commit giữ trạng thái hiện tại của repository. Một commit cũng được đặt tên là SHA1 hash. Bạn có thể xem xét một đối tượng commit như là một nút của danh sách liên kết. Mỗi đối tượng commit có một điểm con trỏ tới đối tượng commit gốc. Từ một commit đã cho, bạn có thể truy xét trở lại bằng cách nhìn vào điểm con trỏ gốc để xem lịch sử của commit đó. Nếu một commit có nhiều commit gốc, thì khi đó các commit cụ thể sẽ được tạo bởi cách sáp nhập hai nhánh.

Các nhánh - Branches

Các nhánh được sử dụng để tạo ra các tuyến khác của sự phát triển. Theo mặc định, Git có một nhánh chủ, mà tương tự như thân (trunk) trong Subversion. Thông thường, một nhánh được tạo để làm việc về một điểm mới. Một khi điểm này được hoàn thành, nó được sáp nhập lại với nhánh chủ và chúng ta xóa nhánh đó đi. Mỗi nhánh được ám chỉ bởi HEAD, mà các điểm con trỏ tới commit mới nhất trong nhánh. Bất cứ khi nào bạn thực hiện một commit, HEAD được cập nhật bởi các commit mới nhất đó.

Thẻ - Tags

Các thẻ chỉ một tên có nghĩa với một phiên bản xác định trong kho chứa. Các thể là tương tự như các nhánh, nhưng sự khác nhau là các thẻ không thay đổi được. Nó có nghĩa là, thẻ là một nhánh, mà không ai có ý định sửa chúng. Một khi một thẻ được tạo ra cho các commit cụ thể, ngay cả khi bạn tạo một commit mới, nó sẽ không được cập nhật. Thông thường, các nhà phát triển tạo các thẻ cho công bố sản phẩm.

Mô phỏng - Clone

Hoạt động mô phỏng tạo bản sao của repository. Hoạt động này không chỉ kiểm tra việc sao chép, mà còn phản ánh kho toàn bộ repository. Người sử dụng có thể thực hiện rất nhiều thao tác với repository nội bộ. Các mạng thời gian chỉ được tham gia khi các repository instance đang được đồng bộ.

Pull

Hoạt động pull sao chép những thay đổi từ một repository instance xa tới kho nội bộ. Hoạt động này được sử dụng để đồng bộ giữa hai repository instance. Điều này tương tự như hoạt động cập nhật trong Subversion.

Push

Thao tác đẩy (push) sao chép các thay đổi từ repository nội bộ tới một kho xa. Nó được sử dụng để lưu các thay đổi vĩnh viễn trong repository Git. Nó tương tự như hoạt động commit trong Subversion.

HEAD

HEAD là một điểm con trỏ, mà thường trỏ vào commit mới nhất trong nhánh. Bất cứ khi nào bạn thực hiện một commit, HEAD được cập nhật với commit mới nhất đó. Đầu của các nhánh được lưu trong **.git/refs/heads/**directory.

```
[CentOS]$ ls -1 .git/refs/heads/
master

[CentOS]$ cat .git/refs/heads/master
570837e7d58fa4bccd86cb575d884502188b0c49
```

Revision

Revision đại diện cho phiên bản của một mã nguồn. Revision trong Git được đại diện bởi các commit. Những commit này được xác định bởi SHA1 secure hash.

URL

URL đại diện cho vị trí của repository Git. Git URL được giữ trong một tệp config.

```
[tom@CentOS tom_repo]$ pwd
/home/tom/tom_repo

[tom@CentOS tom_repo]$ cat .git/config
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
[remote "origin"]
```

```
url = gituser@git.server.com:project.git
fetch = +refs/heads/*:refs/remotes/origin/*
```

Cài đặt môi trường Git

Trước khi bạn có thể sử dụng Git, bạn phải cài đặt và thực hiện một vài thay đổi về cấu hình. Dưới đây là các bước để cài đặt Git client trên Ubuntu và Centos Linux.

Cài đặt Git client

Nếu bạn đang sử dụng Debian do GNU/Linux phân phối, lệnh apt-get sau là cần thiết thực hiện.

```
[ubuntu ~]$ sudo apt-get install git-core
[sudo] password for ubuntu:

[ubuntu ~]$ git --version
git version 1.8.1.2
```

Và nếu bạn đang sử dụng RPM do GNU/Linux phân phối, bạn sử dụng lệnh yum như dưới:

```
[CentOS ~]$
su -
Password:

[CentOS ~]# yum -y install git-core

[CentOS ~]# git --version
git version 1.7.1
```

Tùy chỉnh môi trường Git

Git cung cấp công cụ cấu hình git cho phép bạn thiết lập các cấu hình đa dạng. Git lưu tất cả các cấu hình chung trong tệp .gitconfig được đặt trong thư mục home của bạn. Để thiết lập những giá trị cấu hình như global, bạn thêm tùy chọn --global, và nếu bạn muốn bỏ tùy chọn này đi, thì các cấu hình của bạn được xác định riêng cho repository Git hiện tại.

Bạn cũng có thể thiết lập cấu hình rỗng rãi cho hệ thống. Git lưu các giá trị trong tệp /etc/gitconfig, mà chứa cấu hình cho mọi người sử dụng và kho chứa trên hệ thống. Để thiết lập những giá trị này, bạn phải có root đúng và sử dụng tùy chọn --system.

Khi code trên được biên dịch và thực hiện, nó tạo kết quả sau:

Thiết lập tên người sử dụng

Thông tin này được sử dụng bởi Git cho mỗi commit.

```
[jerry@CentOS project]$ git config --global user.name "Jerry Mouse"
```

Thiết lập email id

Thông tin này được sử dụng bởi Git cho mỗi commit.

```
[jerry@CentOS project]$ git config --global user.email "jerry@tutorialspoint.com"
```

Tránh các commit sáp nhập khi pull

Bạn pull thay đổi mới nhất từ repository xa và nếu những thay đổi này là khác nhau hoặc phân kỳ, thì khi đó theo mặc định Git tạo các commit sáp nhập. Chúng ta có thể tránh được điều này theo các thiết lập sau:

```
jerry@CentOS project]$ git config --global branch.autosetuprebase always
```

Màu nổi

Các lệnh sau tạo màu nổi trong bảng điều khiển Git.

```
[jerry@CentOS project]$ git config --global color.ui true

[jerry@CentOS project]$ git config --global color.status auto

[jerry@CentOS project]$ git config --global color.branch auto
```

Thiết lập bộ soạn mặc định

Theo mặc định, Git sử dụng hệ thống bộ soạn mặc định, mà được lấy từ VISUAL hoặc EDITOR. Bạn có thể định một bộ soạn khác bằng cách sử dụng config.

```
[jerry@CentOS project]$ git config --global core.editor vim
```

Thiết lập công cụ sáp nhập mặc định

Git không cung cấp một công cụ sáp nhập mặc định cho việc tương tác các thay đổi đối lập nhau trong cây làm việc. Bạn có thể thiết lập công cụ này theo các thiết lập sau:

```
[jerry@CentOS project]$ git config --global merge.tool vimdiff
```

Liệt kê các thiết lập Git

Để thẩm tra lại các thiết lập Git trong repository nội bộ, sử dụng lệnh **git config –list** như dưới đây.

```
[jerry@CentOS ~]$ git config --list
```

Lệnh trên sẽ cho kết quả sau:

```
user.name=Jerry Mouse
user.email=jerry@tutorialspoint.com
push.default=nothing
branch.autosetuprebase=always
color.ui=true
color.status=auto
color.branch=auto
core.editor=vim
merge.tool=vimdiff
```

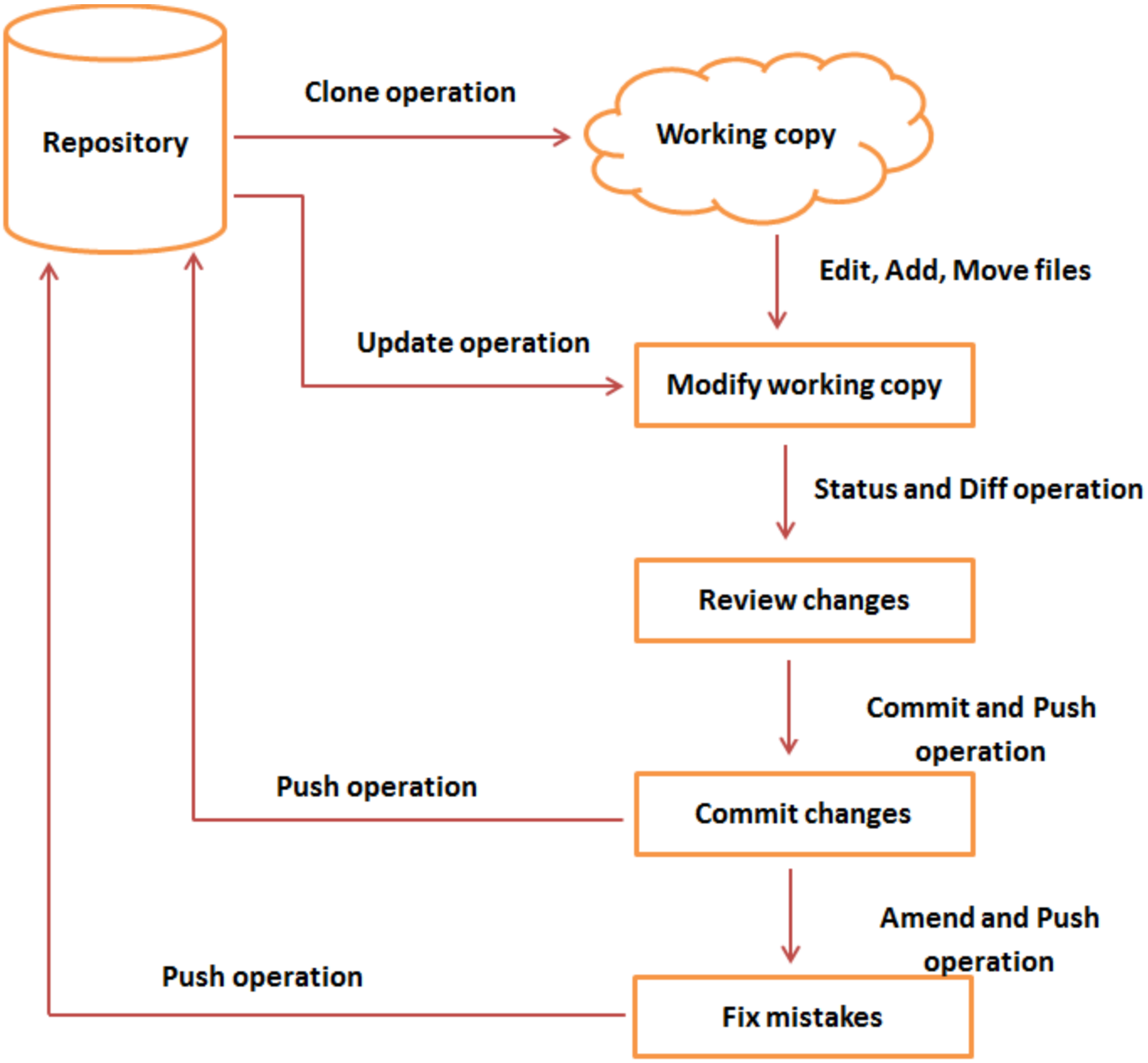
Vòng đời Git

Trong chương này, chúng ta sẽ bàn luận về vòng đời của Git. Và ở chương sau, chúng ta tìm hiểu qua các lệnh Git cho mỗi hoạt động.

Tiến trình làm việc tổng quát như sau:

- Bạn mô phỏng repository Git như là bản sao làm việc.
- Bạn chỉnh sửa bản sao làm việc bằng việc thêm/sửa các file.
- Nếu cần thiết, bạn có thể cập nhật bản sao làm việc bằng cách thực hiện các thay đổi của các nhà phát triển khác.
- Bạn xem lại các thay đổi trước khi commit.
- Bạn commit các thay đổi. Nếu mọi thứ là tốt, sau đó bạn đẩy những thay đổi này tới repository.
- Sau khi commit, nếu bạn nhận ra một vài thứ là sai, khi đó bạn có thể chỉnh lại cho đúng các commit đó và đẩy các thay đổi này tới repository.

Dưới đây là hình ảnh biểu thị về tiến trình làm việc:



Hoạt động Create trong Git

Trong chương này, chúng ta sẽ học cách tạo một repository git từ xa, từ đó chúng ta sẽ đề cập nó như một Git server. Chúng ta cần một Git server để cho phép team cộng tác với nhau.

Tạo một tài khoản sử dụng mới

```
# add new group
[root@CentOS ~]# groupadd dev
```

```
# add new user
[root@CentOS ~]# useradd -G devs -d /home/gituser -m -s /bin/bash gituser

# change password
[root@CentOS ~]# passwd gituser
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Changing password for user gituser.
New password:
Retype new password:
passwd: all authentication token updated successfully.
```

Tạo một repository rỗng

Hãy cùng chúng tôi khởi chạy một repository mới bằng cách sử dụng lệnh init theo sau bởi tùy chọn --bare. Nó khởi chạy repository mà không là một thư mục làm việc. Theo quy ước, repository rỗng này phải được đặt tên như **.git**.

```
[gituser@CentOS ~]$ pwd
/home/gituser

[gituser@CentOS ~]$ mkdir project.git

[gituser@CentOS ~]$ cd project.git/

[gituser@CentOS project.git]$ ls

[gituser@CentOS project.git]$ git --bare init
Initialized empty Git repository in /home/gituser-m/project.git/

[gituser@CentOS project.git]$ ls
branches config description HEAD hooks info objects refs
```

Cặp khóa chung/riêng

Bạn hãy cùng chúng tôi xem qua tiến trình của việc định cấu hình của một Git server, tiện ích ssh-keygen công cộng/private RSA key pair, mà chúng ta sẽ sử dụng để xác định người sử dụng.

Mở một terminal và nhập lệnh sau và nhấn enter cho mỗi lần nhập. Sau khi hoàn thành, nó sẽ tạo một thư mục .ssh bên trong thư mục home.

```
tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ ssh-keygen
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tom/.ssh/id_rsa): Press Enter Only
Created directory '/home/tom/.ssh'.
Enter passphrase (empty for no passphrase): ----------------> Press Enter Only
Enter same passphrase again: -----------------------------> Press Enter Only
Your identification has been saved in /home/tom/.ssh/id_rsa.
Your public key has been saved in /home/tom/.ssh/id_rsa.pub.
The key fingerprint is:
df:93:8c:a1:b8:b7:67:69:3a:1f:65:e8:0e:e9:25:a1 tom@CentOS
The key's randomart image is:
+--[ RSA 2048]----+
| |
| |
| |
|
.
|
| Soo |
| o*B. |
| E = *.= |
| oo==. . |
| ..+Oo
|
+-----------------+
```

ssh-keygen đã tạo hai khóa, đầu tiên là private (i.e., id_rsa) và thứ hai là public (i.e., id_rsa.pub).

Ghi chú: Bạn đừng bao giờ chia sẻ khóa Private của bạn với ai khác.

Thêm các khóa vào các khóa được ủy quyền

Giả sử có hai nhà lập trình làm việc trên một dự án, tên là Tom và Jerry. Cả hai có khóa generate public. Hãy cùng xem cách để sử dụng các khóa này để xác nhận.

Tom nhập khóa chung của anh ta tới server bằng cách sử dụng lệnh ssh-copy-id như dưới đây:

```
[tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa.pub gituser@git.server.com
```

Lệnh trên sẽ tạo ra kết quả sau:

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh 'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

Theo cách tương tự, Jerry nhập khóa chung tới server bằng cách sử dụng lệnh ssh-copy-id.

```
[jerry@CentOS ~]$ pwd
/home/jerry

[jerry@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa gituser@git.server.com
```

Lệnh trên sẽ tạo ra kết quả sau:

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh 'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

Đẩy (Push) các thay đổi tới repository

Chúng ta đã tạo ra một repository rỗng trên server và cho phép sự truy cập của hai người. Từ đó, Tom và Jerry có thể đẩy các thay đổi của họ tới repository bằng cách thêm nó vào như một điều khiển từ xa.

Lệnh init tạo một thư mục .git để giữ siêu dữ liệu về repository mọi lúc nó đọc cấu hình từ tệp .git/config.

Tom tạo một thư mục mới, thêm tệp README, và commit thay đổi của anh ta như là commit đầu tiên. Sau khi ký tác, anh ta thẩm tra các thông báo commit bằng cách chạy lệnh git log.

```
[tom@CentOS ~]$ pwd
```

```
/home/tom

[tom@CentOS ~]$ mkdir tom_repo

[tom@CentOS ~]$ cd tom_repo/

[tom@CentOS tom_repo]$ git init
Initialized empty Git repository in /home/tom/tom_repo/.git/

[tom@CentOS tom_repo]$ echo 'TODO: Add contents for README' > README

[tom@CentOS tom_repo]$ git status -s
?? README

[tom@CentOS tom_repo]$ git add .

[tom@CentOS tom_repo]$ git status -s
A README

[tom@CentOS tom_repo]$ git commit -m 'Initial commit'
```

Lệnh trên tạo kết quả sau:

```
[master (root-commit) 19ae206] Initial commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 README
```

Tom kiểm tra thông báo log bằng cách thực hiện lệnh git log.

```
[tom@CentOS tom_repo]$ git log
```

Lệnh trên sẽ tạo kết quả sau:

```
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Tom commit thay đổi của anh ta tới repository nội bộ. Bây giờ là thời gian để đẩy những thay đổi tới repository từ xa. Nhưng trước đó, chúng ta phải thêm repository này như một điều khiển từ xa,

đây là một hoạt động một lần. Sau đó, anh ta có thể đẩy những thay đổi này tới repository từ xa một cách an toàn.

Ghi chú: Theo mặc định, Git chỉ đẩy tới những nhánh kết nối: cho mỗi nhánh mà tồn tại trên side nội bộ, side điều khiển từ xa được cập nhật nếu một nhánh với cùng tên đã tồn tại trên đó. Trong phần hướng dẫn của chúng tôi, mỗi khi chúng tôi đẩy những thay đổi tới nhánh origin master, chúng tôi sử dụng chính xác tên nhánh.

```
[tom@CentOS tom_repo]$ git remote add origin gituser@git.server.com:project.git

[tom@CentOS tom_repo]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả:

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 242 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
master -> master
```

Bây giờ, những thay đổi này đã được commit thành công tới repository điều khiển từ xa.

Hoạt động Clone trong Git

Chúng ta có một repository rỗng trên máy chủ và Tom cũng đẩy phiên bản đầu tiên của anh ta. Bây giờ, Jerry có thể quan sát những thay đổi của anh ta. Hoạt động clone (tạo bản sao) tạo một instance của repository từ xa.

Jerry tạo một thư mục mới trong thư mục home của anh ta và thực hiện hoạt động mô phỏng.

```
[jerry@CentOS ~]$ mkdir jerry_repo

[jerry@CentOS ~]$ cd jerry_repo/

[jerry@CentOS jerry_repo]$ git clone gituser@git.server.com:project.git
```

Lệnh trên sẽ tạo kết quả sau:

```
Initialized empty Git repository in /home/jerry/jerry_repo/project/.git/
```

```
remote: Counting objects: 3, done.
Receiving objects: 100% (3/3), 241 bytes, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
```

Jerry thay đổi thư mục này tới thư mục nội bộ mới và liệt kê nội dung thư mục của nó.

```
[jerry@CentOS jerry_repo]$ cd project/

[jerry@CentOS jerry_repo]$ ls
README
```

Thực hiện thay đổi trong Git

Jerry tạo một bản sao repository trên máy anh ta và quyết định thực hiện các chuỗi hoạt động cơ bản. Vì thế anh ta tạo tệp string.c. Sau khi thêm nội dung, string.c sẽ trông giống như sau:

```
#include <stdio.h>

int my_strlen(char *s)
{
   char *p = s;

   while (*p)
      ++p;

   return (p - s);
}

int main(void)
{
   int i;
   char *s[] =
   {
      "Git tutorials",
      "Tutorials Point"
   };
```

```
   for (i = 0; i < 2; ++i)

   printf("string lenght of %s = %d\n", s[i], my_strlen(s[i]));

   return 0;
}
```

Anh ta biên dịch và kiểm tra code của anh ta và mọi thứ làm việc bình thường. Bây giờ anh ta có thể thêm những thay đổi này tới repository một cách an toàn.

Git thêm hoạt động thêm file tới khu vực staging.

```
[jerry@CentOS project]$ git status -s
?? string
?? string.c

[jerry@CentOS project]$ git add string.c
```

Git đang chỉ một câu hỏi đánh dấu trước khi đặt tên file. Rõ ràng là các file này không là một phần của Git, và đó là tại sao Git không biết phải làm điều gì với những file này. Đó là tại sao, Git đang chỉ một câu hỏi đánh dấu trước khi đặt tên file.

Jerry đã thêm file tới khu vực staging, lệnh git status sẽ chỉ các file có trong khu vực này.

```
[jerry@CentOS project]$ git status -s
A string.c
?? string
```

Để commit những thay đổi, anh ta sử dụng lệnh git commit theo sau bởi tùy chọn –m. Nếu chúng ta quên tùy chọn –m. Git sẽ mở một text editor, tại đó chúng ta có thể viết thông tin commit nhiều dòng.

```
[jerry@CentOS project]$ git commit -m 'Implemented my_strlen function'
```

Lệnh trên sẽ tạo ra kết quả:

```
[master cbe1249] Implemented my_strlen function
1 files changed, 24 insertions(+), 0 deletions(-)
```

```
create mode 100644 string.c
```

Sau khi commit để xem chi tiết log, anh ta chạy lệnh git log. Nó sẽ hiển thị thông tin của tất cả các commit với ID ký thác, tác giả ký thác, ngày commit và SHA-1 hash của ký thác.

```
[jerry@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả:

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Review thay đổi trong Git

Sau khi kiểm tra lại các chi tiết ký thác, Jerry nhận thấy rằng độ dài chuỗi không thể âm, do đó anh ta quyết định thay đổi kiểu của chức năng my_strlen.

Jerry sử dụng lệnh git log để quan sát các chi tiết log.

```
[jerry@CentOS project]$ git log
```

Lệnh trên sẽ tạo kết quả sau:

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function
```

Jerry sử dụng lệnh git show để kiểm tra các chi tiết ký thác. Git chỉ rõ lệnh được lấy từ ID ký thác SHA-1 như là một tham số.

```
[jerry@CentOS project]$ git show cbe1249b140dad24b2c35b15cc7e26a6f02d2277
```

Lệnh trên sẽ tạo kết quả sau:

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function



diff --git a/string.c b/string.c
new file mode 100644
index 0000000..187afb9
--- /dev/null
+++ b/string.c
@@ -0,0 +1,24 @@
+#include <stdio.h>
+
+int my_strlen(char *s)
+{
   +
   char *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p -s );
   +
}
+
```

Anh ta thay đổi kiểu của chức năng từ int đến size_t. Sau khi kiểm tra code, anh ta duyệt lại các thay đổi bằng cách chạy lệnh git diff.

```
[jerry@CentOS project]$ git diff
```

Lệnh trên sẽ tạo kết quả sau:

```
diff --git a/string.c b/string.c
```

```
index 187afb9..7da2992 100644
--- a/string.c
+++ b/string.c
@@ -1,6 +1,6 @@
#include <stdio.h>

-int my_strlen(char *s)
+size_t my_strlen(char *s)
{
   char *p = s;
   @@ -18,7 +18,7 @@ int main(void)
};
for (i = 0; i < 2; ++i)
{
   - printf("string lenght of %s = %d\n", s[i], my_strlen(s[i]));
   + printf("string lenght of %s = %lu\n", s[i], my_strlen(s[i]));
   return 0;
}
```

Git diff chỉ ký hiệu '+' trước các dòng mà thêm mới và thêm '-' cho các dòng bị xóa.

Commit trong Git

Jerry đã ký thác các thay đổi và anh ta muốn chỉnh sửa cho đúng các ký thác vừa qua của anh ta. Trong trường hợp này, chức năng git amend sẽ giúp làm điều này. Chức năng này thay đổi ký thác vừa qua bao gồm cả thông báo ký thác của bạn; nó tạo ra một ID ký thác mới.

Trước khi hoạt động tùy chỉnh, anh ta kiểm tra ký thác log.

```
[jerry@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function
```

```
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Jerry ký thác các thay đổi mới với chức năng --amend, và quan sát ký thác log.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git add string.c

[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git commit --amend -m 'Changed return type of my_strlen to size_t'
[master d1e19d3] Changed return type of my_strlen to size_t
1 files changed, 24 insertions(+), 0 deletions(-)
create mode 100644 string.c
```

Bây giờ git log sẽ chỉ một thông báo ký thác mới với ID ký thác mới:

```
[jerry@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
```

```
Initial commit
```

Hoạt động Push trong HTML

Jerry chỉnh sửa các commit trước bằng cách sử dụng hoạt động amend và anh ta sẵn sàng để push các thay đổi. Hoạt động push lưu dữ liệu vĩnh cửu tới repository Git. Sau một hoạt động push thành công, các nhà lập trình khác có thể quan sát các thay đổi của Jerry.

Anh ta thực hiện lệnh git log để quan sát các chi tiết commit .

```
[jerry@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t
```

Trước khi thực hiện thao tác push, anh ta muốn duyệt lại các thay đổi của anh ta, vì thế anh ta sử dụng lệnh git log để duyệt các thay đổi.

```
[jerry@CentOS project]$ git show d1e19d316224cddc437e3ed34ec3c931ad803958
```

Lệnh trên sẽ tạo ra kết quả:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t

diff --git a/string.c b/string.c
new file mode 100644
index 0000000..7da2992
--- /dev/null
+++ b/string.c
@@ -0,0 +1,24 @@
+#include <stdio.h>
+
```

```
+size_t my_strlen(char *s)
+
{
   +
   char *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p -s );
   +
}
+
+int main(void)
+
{
   + int i;
   + char *s[] =
   {
      + "Git tutorials",
      + "Tutorials Point"
      +
   };
   +
   +
   +
   for (i = 0; i < 2; ++i)
   printf("string lenght of %s = %lu\n", s[i], my_strlen(s[i]));
   +
   +
   return 0;
   +
}
```

Jerry vui mừng với những thay đổi của anh ta và sẵn sàng để push những thay đổi này.

```
[jerry@CentOS project]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 517 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
19ae206..d1e19d3 master -> master
```

Các thay đổi của Jerry đã được push thành công đến repository; bây giờ các nhà lập trình khác có thể quan sát những thay đổi này bằng cách thực hiện hoạt động mô phỏng hoặc cập nhật.

Hoạt động Update trong Git

Tùy chỉnh các chức năng đang tồn tại

Tom thực hiện hoạt động mô phỏng và thấy một file mới string.c. Anh ta muốn biết ai đã thêm file này vào kho chứa và với mục đích gì, vì thế anh ta chạy lệnh git log.

```
[tom@CentOS ~]$ git clone gituser@git.server.com:project.git
```

Lệnh này sẽ tạo ra kết quả sau:

```
Initialized empty Git repository in /home/tom/project/.git/
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
Receiving objects: 100% (6/6), 726 bytes, done.
remote: Total 6 (delta 0), reused 0 (delta 0)
```

Hoạt động mô phỏng sẽ tạo một thư mục mới bên trong thư mục làm việc hiện tại. Anh ta thay đổi thư mục tới thư mục vừa tạo mới và chạy lệnh git log.

```
[tom@CentOS ~]$ cd project/

[tom@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t
```

```
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Sau khi quan sát log, anh ta nhận thấy rằng tệp string.c được thêm bởi Jerry để thực hiện các hoạt động chuỗi cơ bản. Anh ta muốn biết về đoạn code của Jerry. Vì thế anh ta mở string.c trong text editor và ngay lập tức tìm thấy một bug (lỗi). Trong chức năng my_strlen, Jerry không sử dụng điểm trỏ hằng số. Vì thế, anh ta quyết định chỉnh sửa code của Jerry. Sau khi thực hiện chỉnh sửa, đoạn code trông như sau:

```
[tom@CentOS project]$ git diff
```

Lệnh trên sẽ tạo ra kết quả sau:

```
diff --git a/string.c b/string.c
index 7da2992..32489eb 100644
--- a/string.c
+++ b/string.c
@@ -1,8 +1,8 @@
#include <stdio.h>
-size_t my_strlen(char *s)
+size_t my_strlen(const char *s)
{
   - char *p = s;
   + const char *p = s;
   while (*p)
   ++p;
}
```

Sau khi kiểm tra, anh ta commit các thay đổi của anh ta.

```
[tom@CentOS project]$ git status -s
M string.c
?? string

[tom@CentOS project]$ git add string.c
```

```
[tom@CentOS project]$ git commit -m 'Changed char pointer to const char pointer'
[master cea2c00] Changed char pointer to const char pointer
1 files changed, 2 insertions(+), 2 deletions(-)

[tom@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit cea2c000f53ba99508c5959e3e12fff493b
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 08:32:07 2013 +0530

Changed char pointer to const char pointer


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit
```

Tom sử dụng lệnh git push để push các thay đổi của anh ta.

```
[tom@CentOS project]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
d1e19d3..cea2c00 master -> master
```

Thêm chức năng mới

Trong khi ấy, Jerry quyết định thực hiện chức năng string compare. Vì thế anh ta chỉnh sửa string.c. Sau khi tùy chỉnh, file trông như sau:

```
[jerry@CentOS project]$ git diff
```

Lệnh trên sẽ tạo ra kết quả sau:

```
index 7da2992..bc864ed 100644
--- a/string.c
+++ b/string.c
30Git Tutorials
@@ -9,9 +9,20 @@ size_t my_strlen(char *s)
return (p -s );
}
+char *my_strcpy(char *t, char *s)
+
{
   +
   char *p = t;
   +
   + while (*t++ = *s++)
   + ;
   +
   +
   return p;
   +
}
+
int main(void)
{
   int i;
   +
   char p1[32];
   char *s[] =
   {
      "Git tutorials",
      "Tutorials Point"
      @@ -20,5 +31,7 @@ int main(void)
```

```
      for (i = 0; i < 2; ++i)
      printf("string lenght of %s = %lu\n", s[i], my_strlen(s[i]));
      +
      printf("%s\n", my_strcpy(p1, "Hello, World !!!"));
      +
      return 0;
   }
}
```

Sau khi kiểm tra, anh ta sẵn sàng push thay đổi này.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git add string.c

[jerry@CentOS project]$ git commit -m "Added my_strcpy function"
[master e944e5a] Added my_strcpy function
1 files changed, 13 insertions(+), 0 deletions(-)
```

Trước khi thực hiện hoạt động push, anh ta chỉnh sửa commit bằng cách quan sát các thông báo log.

```
[jerry@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit e944e5aab74b26e7447d3281b225309e4e59efcd
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:41:42 2013 +0530

Added my_strcpy function


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t
```

```
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Jerry vui mừng với những thay đổi này và anh ta muốn thực hiện push những thay đổi đó.

```
[jerry@CentOS project]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả sau:

```
To gituser@git.server.com:project.git
! [rejected]
master -> master (non-fast-forward)
error: failed to push some refs to 'gituser@git.server.com:project.git'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes before pushing again. See the 'Note about
fast-forwards' section of 'git push --help' for details.
```

Nhưng Git không cho phép Jerry push những thay đổi của anh ta. Bởi vì Git nhận định rằng repository từ xa và repository nội bộ của Jerry là không đồng bộ. Vì điều đó, anh ta có thể lạc mất lịch sử của dự án. Để tránh mớ hỗn độn này, Git quên hoạt động này. Bây giờ, Jerry đầu tiên phải cập nhật repository nội bộ và chỉ khi sau đó, anh ta có thể push nhưng thay đổi riêng này.

Gọi ra những thay đổi mới nhất

Jerry chạy lệnh git pull để đồng bộ hóa repository nội bộ của anh ta với repository từ xa.

```
[jerry@CentOS project]$ git pull
```

Lệnh trên sẽ tạo ra kết quả sau:

```
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git.server.com:project
d1e19d3..cea2c00 master -> origin/master
```

```
First, rewinding head to replay your work on top of it...
Applying: Added my_strcpy function
```

Sau khi thực hiện hoạt động pull, Jerry kiểm tra các thông báo log và tìm thấy các chi tiết commit của Tom với ID commit cea2c000f53ba99508c5959e3e12fff493ba6f69.

```
[jerry@CentOS project]$ git log
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit e86f0621c2a3f68190bba633a9fe6c57c94f8e4f
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:41:42 2013 +0530

Added my_strcpy function


commit cea2c000f53ba99508c5959e3e12fff493ba6f69
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 08:32:07 2013 +0530

Changed char pointer to const char pointer


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit
```

Bây giờ repository nội bộ của Jerry đã được đồng bộ đầy đủ với repository từ xa. Vì thế anh ta có thể push những thay đổi một cách an toàn.

```
[jerry@CentOS project]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả:

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 455 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
cea2c00..e86f062 master -> master
```

Hoạt động Stash trong Git

Giả sử bạn đang thực hiện một tính năng mới của sản phẩm của bạn. Code của bạn đang trong tiến trình thực hiện thì đột nhiên một vị khách đến thăm. Bởi vì điều này, bạn phải đi ra ngoài trong một vài giờ. Bạn không thể commit phần code này và cũng không thể quăng nó đi đâu những thay đổi của bạn. Vì thế bạn cần một vài không gian tạm thời, mà tại nơi đó bạn có thể giữ những thay đổi cục bộ này và sau đó quay lại commit nó.

Trong Git, hoạt động stash giúp bạn stash những file đã được chỉnh sửa, những thay đổi, và lưu chúng trên một stack của những thay đổi chưa được hoàn thành, và từ đó bạn có thể ứng dụng lại bất cứ lúc nào.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string
```

Bây giờ bạn muốn chuyển branch do một vị khách đến bất ngờ, nhưng bạn không muốn commit những gì bạn đã làm trên đó, vì thế bạn stash những thay đổi đi. Để đẩy những cái stash mới này vào trong stack, bạn chạy lệnh git stash.

```
[jerry@CentOS project]$ git stash
Saved working directory and index state WIP on master: e86f062 Added my_strcpy function
HEAD is now at e86f062 Added my_strcpy function
```

Bây giờ thư mục làm việc của bạn đã sạch trơn và tất cả những thay đổi được giữ trên một stack. Hãy cùng thẩm tra lại nó với lệnh git status.

```
[jerry@CentOS project]$ git status -s
?? string
```

Bây giờ bạn có thể chuyển branch một cách an toàn và làm bất cứ điều gì. Chúng ta có thể quan sát danh sách của những thay đổi được stash bằng cách sử dụng lệnh git stash list.

```
[jerry@CentOS project]$ git stash list
stash@{0}: WIP on master: e86f062 Added my_strcpy function
```

Giả sử bạn đã giải quyết xong việc với vị khách hàng mới đến và bạn quay trở lại làm đoạn code đang dở dang của mình, bạn chỉ cần thực hiện lện git stash pop, để di chuyển những thay đổi từ stack và đặt chúng vào thư mục làm việc hiện tại.

```
[jerry@CentOS project]$ git status -s
?? string

[jerry@CentOS project]$ git stash pop
```

Lệnh trên sẽ tạo ra kết quả sau:

```
# On branch master
# Changed but not updated:
# (use "git add ..." to update what will be committed)
# (use "git checkout -- ..." to discard changes in working directory)
#
#
modified: string.c
#
# Untracked files:
# (use "git add ..." to include in what will be committed)
#
#
string
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (36f79dfedae4ac20e2e8558830154bd6315e72d4)

[jerry@CentOS project]$ git status -s
M string.c
?? string
```

Hoạt động Move trong Git

Như tên gọi đã đề cập, hoạt động move dời một thư mục hoặc một file từ một vị trí này đến một vị trí khác. Tom quyết định di chuyển code nguồn vào trong thư mục src. Cấu trúc thư mục được chỉnh sửa này sẽ xuất hiện như sau:

```
[tom@CentOS project]$ pwd
/home/tom/project

[tom@CentOS project]$ ls
README string string.c

[tom@CentOS project]$ mkdir src

[tom@CentOS project]$ git mv string.c src/

[tom@CentOS project]$ git status -s
R string.c -> src/string.c
?? string
```

Để tạo ra các thay đổi cố định (vĩnh viễn), chúng tôi push cấu trúc thư mục được chỉnh sửa tới repository từ xa để mà các nhà lập trình khác có thể thấy nó.

```
[tom@CentOS project]$ git commit -m "Modified directory structure"

[master 7d9ea97] Modified directory structure
1 files changed, 0 insertions(+), 0 deletions(-)
rename string.c => src/string.c (100%)

[tom@CentOS project]$ git push origin master
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
e86f062..7d9ea97 master -> master
```

Trong thư mục nội bộ của Jerry, trước khi thực hiện hoạt động pull, nó sẽ chỉ cấu trúc thư mục cũ.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project
```

```
[jerry@CentOS project]$ ls
README string string.c
```

Nhưng sau khi pull xong, cấu trúc thư mục sẽ được cập nhật. Bây giờ, Jerry có thể nhìn thấy thư mục src và các file có mặt bên trong thư mục đó.

```
[jerry@CentOS project]$ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git.server.com:project
e86f062..7d9ea97 master -> origin/master
First, rewinding head to replay your work on top of it...
Fast-forwarded master to 7d9ea97683da90bcdb87c28ec9b4f64160673c8a.

[jerry@CentOS project]$ ls
README src string

[jerry@CentOS project]$ ls src/
string.c
```

Hoạt động Rename trong Git

Tới giờ, cả Tom và Jerry đều đang sử dụng các lệnh bằng tay để biên dịch dự án của họ. Bây giờ, Jerry quyết định tạo Makefile cho dự án và cũng đặt một tên cho file " string.c".

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project

[jerry@CentOS project]$ ls
README src

[jerry@CentOS project]$ cd src/

[jerry@CentOS src]$ git add Makefile

[jerry@CentOS src]$ git mv string.c string_operations.c
```

```
[jerry@CentOS src]$ git status -s
A Makefile
R string.c -> string_operations.c
```

Git đang chỉ R trước tên file để chỉ rằng file đã được đặt lại tên.

Đối với hoạt động commit, Jerry sử dụng đuôi -a, mà làm cho git commit tự động tìm các file đã được chỉnh sửa.

```
[jerry@CentOS src]$ git commit -a -m 'Added Makefile and renamed strings.c to
string_operations.c '

[master 94f7b26] Added Makefile and renamed strings.c to string_operations.c
1 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 src/Makefile
rename src/{string.c => string_operations.c} (100%)
```

Sau khi commit, anh ta push những thay đổi của anh ta tới repository.

```
[jerry@CentOS src]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 6, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 396 bytes, done.
Total 4 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
7d9ea97..94f7b26 master -> master
```

Bây giờ, các nhà lập trình khác có thể quan sát những thay đổi bằng cách cập nhật repository nội bộ của họ.

Hoạt động Delete trong Git

Tom cập nhật repository nội bộ và tìm mã nhị phân được biên dịch trong thư mục src. Sau khi quan sát thông báo commit, anh ta nhận ra rằng mã này được thêm bởi Jerry.

```
[tom@CentOS src]$ pwd
/home/tom/project/src
```

```
[tom@CentOS src]$ ls
Makefile string_operations string_operations.c

[tom@CentOS src]$ file string_operations
string_operations: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked
(uses
shared libs), for GNU/Linux 2.6.18, not stripped

[tom@CentOS src]$ git log
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary
```

VCS chỉ được sử dụng để giữ các code nguồn và không biểu diễn được các mã nhị phân. Vì thế, Tom quyết định dọn các file này từ trong repository. Với các thao tác xa hơn, anh ta sử dụng lệnh git rm.

```
[tom@CentOS src]$ ls
Makefile string_operations string_operations.c

[tom@CentOS src]$ git rm string_operations
rm 'src/string_operations'

[tom@CentOS src]$ git commit -a -m "Removed executable binary"

[master 5776472] Removed executable binary
1 files changed, 0 insertions(+), 0 deletions(-)
delete mode 100755 src/string_operations
```

Sau khi commit, anh ta push những thay đổi tới repository.

```
[tom@CentOS src]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 310 bytes, done.
```

```
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
29af9d4..5776472 master -> master
```

Sửa lỗi trong Git

Đã là con người thì ai cũng phạm sai lầm. Vì thế mỗi VCS đều cung cấp một tính năng để sửa lỗi tại một điểm nào đó. Git cung cấp một tính năng mà chúng ta có thể sử dụng để undo các chỉnh sửa mà đã làm trên repository nội bộ.

Giả sử người sử dụng tình cờ thực hiện một vài thay đổi tới repository nội bộ và sau đó muốn undo những thay đổi này. Trong những trường hợp như vậy, hoạt động revert đóng một vai trò quan trọng.

Trả lại những thay đổi chưa được commit

Giả sử Jerry tình cờ chỉnh sửa một file từ trong repository nội bộ của anh ta. Nhưng anh ta muốn undo lại tùy chỉnh này. Để giải quyết tình huống này, chúng ta sử dụng lệnh git checkout. Chúng ta có thể sử dụng lệnh này để trả lại những nội dung của một file.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git checkout string_operations.c

[jerry@CentOS src]$ git status –s
```

Xa hơn nữa, chúng ta có thể sử dụng lệnh git checkout để đạt được các file bị xóa từ repository nội bộ. Giả sử Tom xóa một file từ repository nội bộ và anh ta muốn sử dụng file này sau đó. Chúng ta có thể thực hiện được điều này bằng cách sử dụng lệnh giống trên.

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src

[tom@CentOS src]$ ls -1
Makefile
string_operations.c
```

```
[tom@CentOS src]$ rm string_operations.c

[tom@CentOS src]$ ls -1
Makefile

[tom@CentOS src]$ git status -s
D string_operations.c
```

Git đang chỉ ký tự D trước tên file. Điều này chỉ rằng file đã được xóa từ repository nội bộ.

```
[tom@CentOS src]$ git checkout string_operations.c

[tom@CentOS src]$ ls -1
Makefile
string_operations.c

[tom@CentOS src]$ git status -s
```

Ghi chú: Chúng ta có thể thực hiện tất cả những hoạt động này trước hoạt động commit.

Dỡ bỏ những thay đổi từ khu vực tổ chức

Chúng ta đã nhìn thấy khi chúng ta chúng ta thực hiện một hoạt động thêm, các file di chuyển từ repository nội bộ tới khu vực tổ chức. Nếu một người sử dụng tình cờ chỉnh sửa một file và thêm nó vào trong khu vực tổ chức, anh ta có thể trả lại những thay đổi, bằng cách sử dụng lệnh git checkout.

Trong Git, có một điểm con trỏ HEAD mà luôn luôn trỏ tại commit mới nhất. Nếu bạn muốn undo một sự thay đổi từ khu vực tổ chức, thì khi đó bạn có thể sử dụng lệnh git checkout, nhưng với lệnh này, bạn phải cung cấp thêm một tham số, i.e., điểm trỏ HEAD. tham số con trỏ commit thêm vào này chỉ thị lệnh git checkout để reset cây làm việc và cũng để dỡ bỏ những thay đổi được tổ chức.

Giả sử Tom chỉnh sửa một file từ repository nội bộ. Nếu chúng ta quan sát trạng thái của file này, nó sẽ chỉ rằng file này được chỉnh sửa nhưng không thêm vào trong khu vực tổ chức.

```
tom@CentOS src]$ pwd
/home/tom/top_repo/project/src
# Unmodified file
```

```
[tom@CentOS src]$ git status -s

# Modify file and view it's status.
[tom@CentOS src]$ git status -s
M string_operations.c

[tom@CentOS src]$ git add string_operations.c
```

Git status chỉ rằng file hiện diện trong khu vực tổ chức, bây giờ trả lại nó bằng cách sử dụng lệnh git command và quan sát trạng thái của file được trả lại.

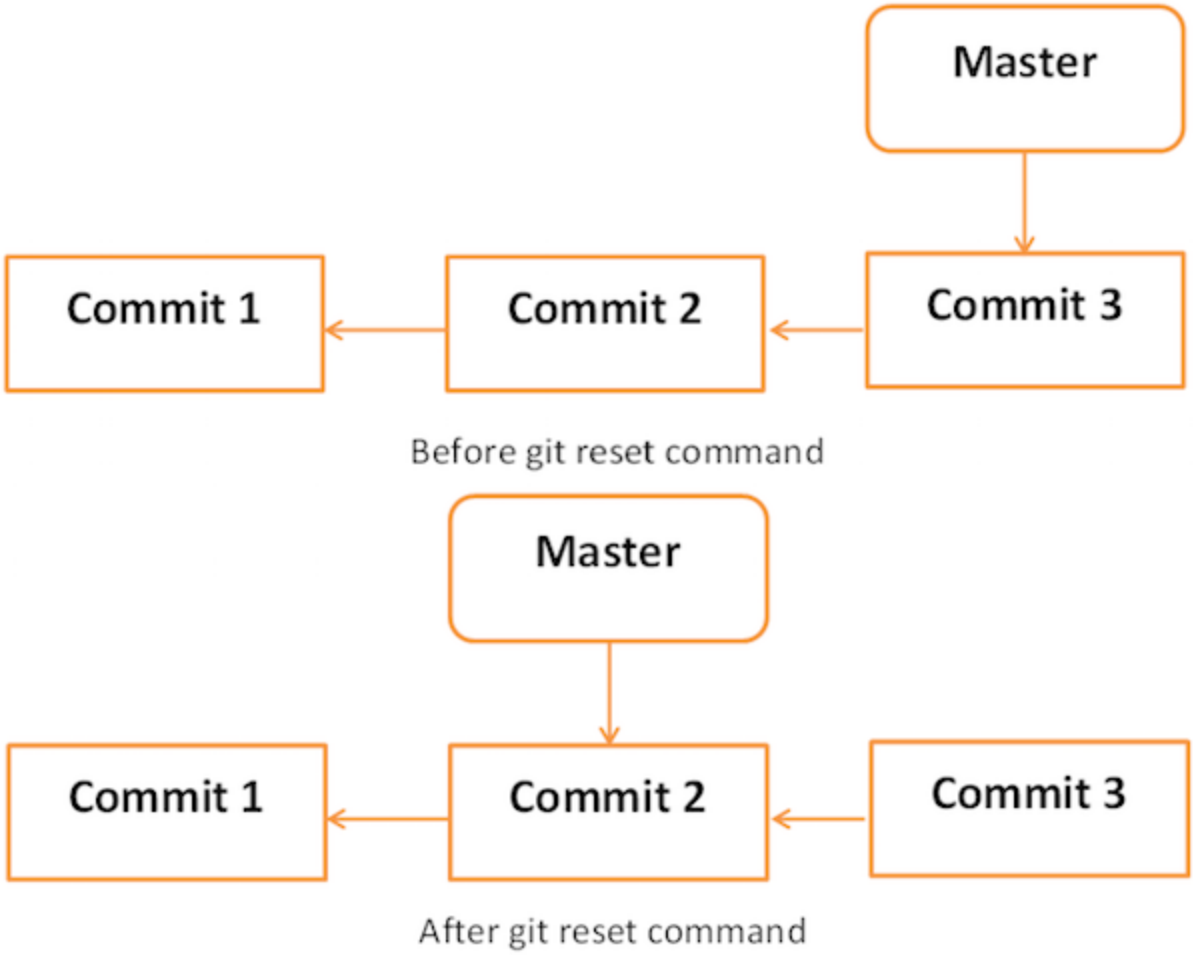
```
[tom@CentOS src]$ git checkout HEAD -- string_operations.c

[tom@CentOS src]$ git status -s
```

Di chuyển điểm trỏ HEAD với git reset

Sau khi thực hiện một số thay đổi, bạn có thể quyết định dỡ bỏ những thay đổi này. Lệnh git reset được sử dụng để reset hoặc trả lại những thay đổi. Chúng ta có thể thực hiện ba kiểu khác nhau của hoạt động reset.

Sơ đồ dưới chỉ tiến trình của lệnh git reset.



Soft

Mỗi nhánh có một điểm trỏ HEAD, mà trỏ vào commit mới nhất. Nếu chúng ta sử dụng lệnh git reset với tùy chọn --soft theo sau bằng ID commit, thì khi đó nó sẽ chỉ reset điểm trỏ HEAD mà không phá hủy bất cứ thứ gì.

Tệp **.git/refs/heads/master** giữ ID commit của điểm trỏ HEAD. Chúng ta có thể thẩm tra nó bằng cách sử dụng lệnh git log-1.

```
[jerry@CentOS project]$ cat .git/refs/heads/master
577647211ed44fe2ae479427a0668a4f12ed71a1
```

Bây giờ, quan sát các ID commit mới nhất, mà sẽ kết nối với ID commit trên.

```
[jerry@CentOS project]$ git log -2
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary


commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary
```

Hãy cùng chúng tôi reset điểm trỏ HEAD.

```
[jerry@CentOS project]$ git reset --soft HEAD~
```

Bây giờ chúng ta vừa reset điểm trỏ HEAD trở lại sau một vị trí. Chúng ta kiểm tra lại nội dung của tệp **.git/refs/heads/master file**.

```
[jerry@CentOS project]$ cat .git/refs/heads/master
29af9d45947dc044e33d69b9141d8d2dad37cc62
```

ID commit từ file được thay đổi, bây giờ thẩm tra nó bằng cách kiểm tra thông báo commit.

```
jerry@CentOS project]$ git log -2
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary


commit 94f7b26005f856f1a1b733ad438e97a0cd509c1a
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:08:01 2013 +0530

Added Makefile and renamed strings.c to string_operations.c
```

Tùy chọn mixed

Lệnh git reset với tùy chọn --mixed trả lại những thay đổi từ trong khu vực tổ chức mà chưa được commit. Nó chỉ trả lại những thay đổi từ khu vực tổ chức. Những thay đổi thực sự đối với bản sao làm việc của một file không bị ảnh hưởng. Lệnh git reset mặc định tương đương với git reset --mixed.

Tùy chọn hard

Nếu bạn sử dụng tùy chọn --hard với lệnh git reset, nó sẽ xóa khu vực tổ chức; nó sẽ reset điểm trỏ HEAD tới những commit mới nhất của ID commit cụ thể và xóa các thay đổi file nội bộ.

Chúng ta kiểm tra ID commit.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git log -1
```

Lệnh trên sẽ tạo kết quả sau:

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary
```

Jerry chỉnh sửa một file bằng cách thêm một lời bình dòng đơn tại vị trí đầu của file.

```
[jerry@CentOS src]$ head -2 string_operations.c
/* This line be removed by git reset operation */
#include <stdio.h>
```

Anh ta thẩm tra nó bằng cách sử dụng lệnh git status.

```
[jerry@CentOS src]$ git status -s
M string_operations.c
```

Jerry thêm file được chỉnh sửa này tới khu vực tổ chức và thẩm tra nó với lệnh git status.

```
[jerry@CentOS src]$ git add string_operations.c
[jerry@CentOS src]$ git status
```

Lệnh trên sẽ tạo kết quả sau:

```
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
#
modified: string_operations.c
#
```

Git status đang chỉ rằng file đang hiện diện trong khu vực tổ chức. Bây giờ, reset điểm trỏ HEAD với tùy chọn --hard.

```
[jerry@CentOS src]$ git reset --hard 577647211ed44fe2ae479427a0668a4f12ed71a1

HEAD is now at 5776472 Removed executable binary
```

Lệnh git reset thực hiện thành công, mà sẽ trả lại file từ khu vực tổ chức cũng như dỡ bỏ bất kỳ những thay đổi cục bộ nào đã thực hiện với file.

```
[jerry@CentOS src]$ git status -s
```

Git status đang chỉ rằng file đã được dỡ bỏ khỏi khu vực tổ chức.

```
[jerry@CentOS src]$ head -2 string_operations.c
#include <stdio.h>
```

Lệnh head cũng chỉ rằng hoạt động reset đã dỡ bỏ những thay đổi cục bộ.

Hoạt động Tag trong Git

Hoạt động tag cho phép cung cấp tên ý nghĩa cho các phiên bản cụ thể trong repository. Giả sử Tom và Jerry quyết định ghép tag ghi vào code dự án của họ để mà họ sau đó có thể truy cập nó dễ dàng.

Tạo các tag

Hãy cùng chúng tôi ghi tag HEAD hiện tại bằng cách sử dụng lệnh git tag. Tom cung cấp một tên tag với tùy chọn -a và cung cấp một thông tin tag với tùy chọn -m.

```
tom@CentOS project]$ pwd
```

```
/home/tom/top_repo/project

[tom@CentOS project]$ git tag -a 'Release_1_0' -m 'Tagged basic string operation code' HEAD
```

Nếu bạn muốn ghi tag một commit cụ thể, thì khi đó bạn sử dụng ID commit chính xác thay vì điểm trỏ HEAD. Tom sử dụng lệnh sau để push tag vào trong khu vực dỡ bỏ.

```
[tom@CentOS project]$ git push origin tag Release_1_0
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 1, done.
Writing objects: 100% (1/1), 183 bytes, done.
Total 1 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new tag]
Release_1_0 -> Release_1_0
```

Quan sát các tag

Tom tạo ra các tag. Bây giờ, Jerry có thể quan sát tất cả các tag có sẵn bằng cách sử dụng lệnh git tag với tùy chọn -l.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git pull
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (1/1), done.
From git.server.com:project
* [new tag]
Release_1_0 -> Release_1_0
Current branch master is up to date.

[jerry@CentOS src]$ git tag -l
Release_1_0
```

Jerry sử dụng lệnh git show theo sau bởi tên tag của nó để quan sát chi tiết hơn về tag.

```
[jerry@CentOS src]$ git show Release_1_0
```

Lệnh trên sẽ tạo ra kết quả sau:

```
tag Release_1_0
Tagger: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 13:45:54 2013 +0530

Tagged basic string operation code



commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary

diff --git a/src/string_operations b/src/string_operations
deleted file mode 100755
index 654004b..0000000
Binary files a/src/string_operations and /dev/null differ
```

Xóa các tag

Tom sử dụng lệnh sau để xóa các tag từ repository lưu nội bộ cũng như từ xa.

```
[tom@CentOS project]$ git tag
Release_1_0

[tom@CentOS project]$ git tag -d Release_1_0
Deleted tag 'Release_1_0' (was 0f81ff4)
# Remove tag from remote repository.

[tom@CentOS project]$ git push origin :Release_1_0
To gituser@git.server.com:project.git
- [deleted]
Release_1_0
```

Hoạt động Patch trong Git

Patch là một file văn bản, mà nội dung của nó tương tự với git diff, nhưng song song với code, nó cũng có siêu dữ liệu về các commit như ID commit, ngày tháng, thông báo commit…. Chúng ta có thể tạo ra một patch từ các commit và người khác có thể áp dụng chúng vào repository của họ.

Jerry thực hiện chức năng strcat cho dự án của anh ta. Jerry có thể tạo một patch của code của anh ta và gửi nó cho Tom. Sau đó, anh ta có thể áp dụng patch nhận được vào code của anh ta.

Jerry sử dụng lệnh git format-patch để tạo một patch cho các commit mới nhất. Nếu bạn muốn tạo một patch cho một commit cụ thể, thì khi đó bạn sử dụng COMMIT_ID với lệnh format-patch.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git status -s
M string_operations.c
?? string_operations

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m "Added my_strcat function"

[master b4c7f09] Added my_strcat function
1 files changed, 13 insertions(+), 0 deletions(-)

[jerry@CentOS src]$ git format-patch -1
0001-Added-my_strcat-function.patch
```

Lệnh trên tạo ra các tệp .patch bên trong thư mục làm việc hiện tại. Tom có thể sử dụng patch để chỉnh sửa file của anh ta. Git cung cấp hai lệnh để áp dụng các patch là git am và git apply, theo cách riêng biệt. Git apply chỉnh sửa các file nội bộ mà không tạo ra commit, trong khi git am chỉnh sửa file và cũng tạo commit.

Để áp dụng patch và tạo commit, bạn sử dụng lệnh sau:

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src

[tom@CentOS src]$ git diff
```

```
[tom@CentOS src]$ git status –s

[tom@CentOS src]$ git apply 0001-Added-my_strcat-function.patch

[tom@CentOS src]$ git status -s
M string_operations.c
?? 0001-Added-my_strcat-function.patch
```

Patch này được áp dụng thành công, bây giờ chúng ta có thể quan sát các chỉnh sửa bằng cách sử dụng lệnh git diff.

```
[tom@CentOS src]$ git diff
```

Lệnh trên tạo ra kết quả sau:

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 8ab7f42..f282fcf 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,5 +1,16 @@
#include <stdio.h>
+char *my_strcat(char *t, char *s)
diff --git a/src/string_operations.c b/src/string_operations.c
index 8ab7f42..f282fcf 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,5 +1,16 @@
#include <stdio.h>
+char *my_strcat(char *t, char *s)
+
{
   +
   char *p = t;
   +
   +
   +
   while (*p)
   ++p;
   +
   while (*p++ = *s++)
```

```
   + ;
   + return t;
   +
}
+
size_t my_strlen(const char *s)
{
   const char *p = s;
   @@ -23,6 +34,7 @@ int main(void)
   {
```

Quản lý nhánh trong Git

Hoạt động nhánh cho phép tạo các tuyến khác nhau của sự phát triển. Chúng ta có thể sử dụng hoạt động này để phân nhánh tiến trình phát triển vào hai hướng khác nhau. Ví dụ, chúng tôi công bố một sản phẩm phiên bản 6 và chúng tôi muốn tạo ra một nhánh để phát triển các tính năng 7.0 mà có thể được giữ cách biệt với sự sửa lỗi trong phiên bản 6.0.

Tạo nhánh

Tom tạo một nhánh mới bằng cách sử dụng lệnh git branch . Chúng ta có thể tạo một nhánh mới từ một nhánh đã tồn tại. Chúng ta có thể sử dụng một commit hoặc một thẻ cụ thể như là điểm bắt đầu. Nếu bất kỳ ID commit cụ thể nào không được cung cấp, thì khi đó nhánh sẽ được tạo ra với HEAD như là điểm bắt đầu.

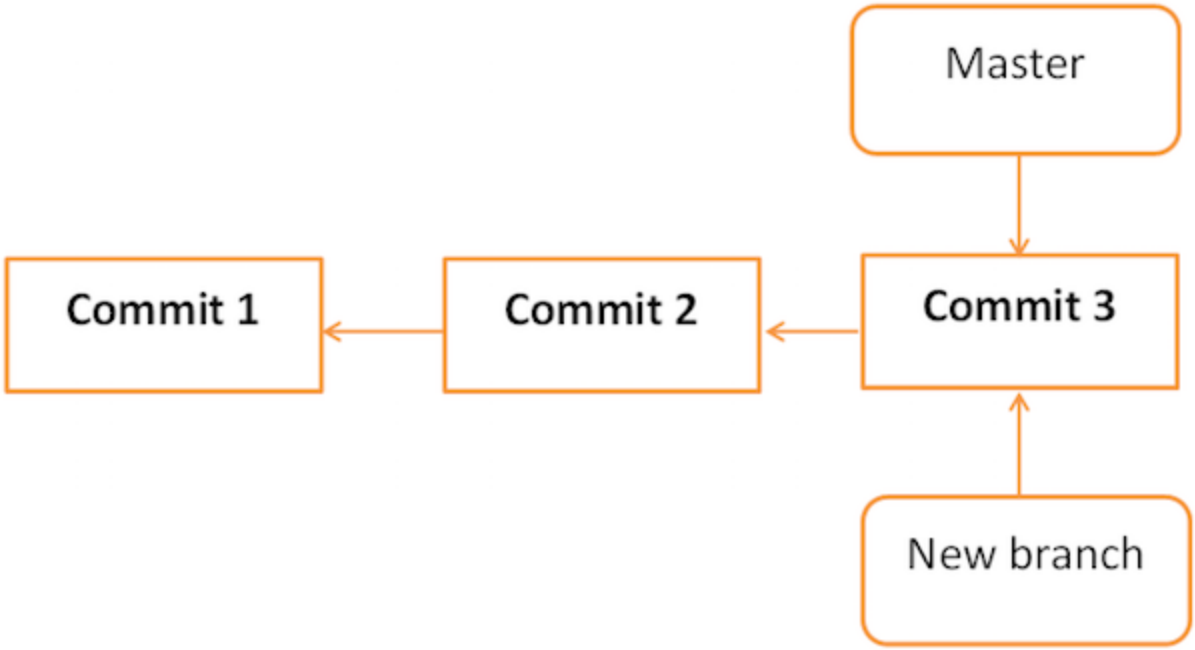
```
[jerry@CentOS src]$ git branch new_branch


[jerry@CentOS src]$ git branch
* master
new_branch
```

Một nhánh mới được tạo ra; Tom sử dụng lệnh git branch để liệt kê các nhánh có sẵn. Git chỉ một dấu hoa thị trước khi kiểm tra nhánh hiện tại.

Hình dưới đây miêu tả hoạt động tạo nhánh:

Before create branch command



After create branch operation

Chuyển đổi giữa các nhánh

Jerry sử dụng lệnh git checkout để chuyển đổi giữa các nhánh:

```
[jerry@CentOS src]$ git checkout new_branch
Switched to branch 'new_branch'
[jerry@CentOS src]$ git branch
master
* new_branch
```

Cách tắt để tạo nhánh và chuyển đổi giữa các nhánh

Ở ví dụ trên, chúng ta đã sử dụng hai lệnh riêng rẽ để tạo và chuyển đổi giữa các nhánh. Git cung cấp tùy chọn -b với lệnh checkout; hoạt động này tạo một nhánh mới và ngay lập tức chuyển đổi đến nhánh mới.

```
[jerry@CentOS src]$ git checkout -b test_branch
Switched to a new branch 'test_branch'

[jerry@CentOS src]$ git branch
master
new_branch
* test_branch
```

Xóa một nhánh

Một nhánh có thể được xóa bằng cách sử dụng tùy chọn -D với lệnh git branch. Nhưng trước khi xóa một nhánh đang tồn tại, bạn chuyển tới nhánh khác.

Jerry hiện tại đang trên nhánh test_branch và anh ta muốn dỡ bỏ nhánh đó. Vì thế anh ta chuyển sang nhánh khác và xóa nhánh như dưới đây:

```
[jerry@CentOS src]$ git branch
master
new_branch
* test_branch

[jerry@CentOS src]$ git checkout master
Switched to branch 'master'

[jerry@CentOS src]$ git branch -D test_branch
Deleted branch test_branch (was 5776472).
```

Bây giờ, Git sẽ chỉ có hai nhánh.

```
[jerry@CentOS src]$ git branch
* master
new_branch
```

Đặt lại tên cho một nhánh

Jerry quyết định thêm sự hỗ trợ cho các ký tự rộng rãi trong dự án các chuỗi hoạt động của anh ta. Anh ta đã tạo một nhánh mới, nhưng tên nhánh không được cung cấp chính xác. Vì thế anh ta thay đổi tên nhánh bằng cách sử dụng tùy chọn -m theo sau bởi old branch name và new branch name.

```
[jerry@CentOS src]$ git branch
* master
new_branch

[jerry@CentOS src]$ git branch -m new_branch wchar_support
```

Bây giờ, lệnh git branch sẽ chỉ tên nhánh mới.

```
[jerry@CentOS src]$ git branch
* master
wchar_support
```

Sáp nhập hai nhánh

Jerry thực hiện một chức năng để trả lại độ dài chuỗi của một chuỗi ký tự rộng. Một code mới sẽ xuất hiện như dưới đây:

```
[jerry@CentOS src]$ git branch
master
* wchar_support

[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git diff
```

Lệnh trên sẽ tạo ra kết quả sau:

```
t a/src/string_operations.c b/src/string_operations.c
index 8ab7f42..8fb4b00 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,4 +1,14 @@
#include <stdio.h>
+#include <wchar.h>
```

```
+
+size_t w_strlen(const wchar_t *s)
+
{
   +
   const wchar_t *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p - s);
   +
}
```

Sau khi kiểm tra, anh ta commit và push những thay đổi của anh ta tới nhánh mới.

```
[jerry@CentOS src]$ git status -s
M string_operations.c
?? string_operations

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Added w_strlen function to return string lenght of wchar_t
string'

[wchar_support 64192f9] Added w_strlen function to return string lenght of wchar_t string
1 files changed, 10 insertions(+), 0 deletions(-)
```

Ghi chú rằng Jerry đang push những thay đổi này tới nhánh mới, đó là tại sao anh ta sử dụng nhánh tên là wchar_support thay cho nhánh master.

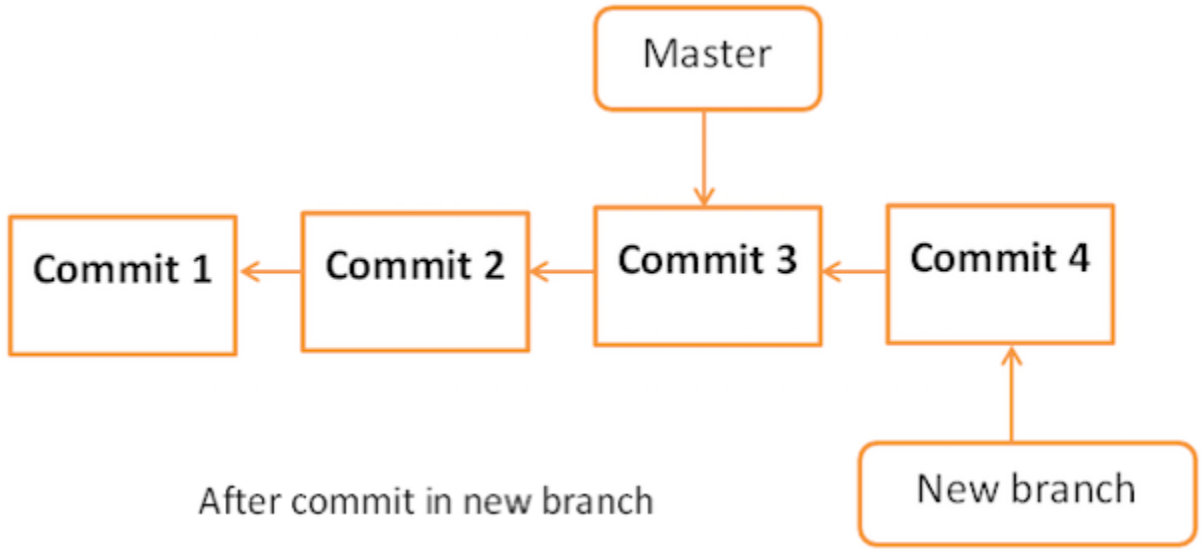
```
[jerry@CentOS src]$ git push origin wchar_support   <--- Observer branch_name
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 507 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
```

```
* [new branch]
wchar_support -> wchar_support
```

Sau khi commit những thay đổi, nhánh mới sẽ xuất hiện như sau:



Tom tò mò về những gì Jerry đang làm trong nhánh tư nhân của cậu ta và anh ta kiểm tra log từ nhánh wchar_support.

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src


[tom@CentOS src]$ git log origin/wchar_support -2
```

Lệnh trên sẽ tạo ra kết quả sau:

```
commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 16:10:06 2013 +0530

Added w_strlen function to return string lenght of wchar_t string


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530
```

```
Removed executable binary
```

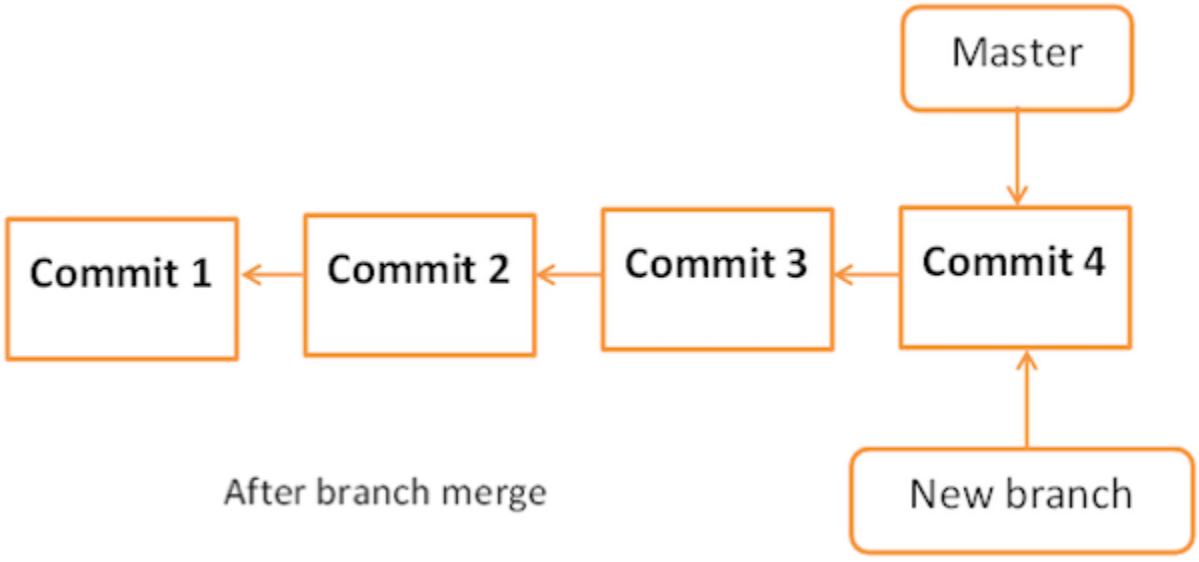
Bằng cách quan sát các thông báo commit, Tom nhận ra rằng Jerry thực hiện chức năng strlen cho ký tự mở rộng và anh ta muốn có chức năng tương tự trong nhánh master. Thay vì thực hiện lại các bước trên, anh ta quyết định lấy code của Jerry bằng cách sáp nhập nhánh của anh ta với nhánh master.

```
[tom@CentOS project]$ git branch
* master

[tom@CentOS project]$ pwd
/home/tom/top_repo/project

[tom@CentOS project]$ git merge origin/wchar_support
Updating 5776472..64192f9
Fast-forward
src/string_operations.c | 10 ++++++++++
1 files changed, 10 insertions(+), 0 deletions(-)
```

Sau hoạt động sáp nhập, nhánh master sẽ xuất hiện như sau:



Bây giờ, nhánh wchar_support đã được nhập với nhánh master. Chúng ta thẩm tra nó bằng cách quan sát thông tin commit hoặc quan sát các chỉnh sửa được thực hiện trong tệp string_operation.c.

```
[tom@CentOS project]$ cd src/
```

```
[tom@CentOS src]$ git log -1

commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3
Author: Jerry Mouse
Date: Wed Sep 11 16:10:06 2013 +0530

Added w_strlen function to return string lenght of wchar_t string

[tom@CentOS src]$ head -12 string_operations.c
```

Lệnh trên sẽ tạo ra kết quả sau:

```
#include <stdio.h>
#include <wchar.h>
size_t w_strlen(const wchar_t *s)
{
   const wchar_t *p = s;

   while (*p)
      ++p;

   return (p - s);
}
```

Sau khi kiểm tra, anh ta push những thay đổi code của anh ta tới nhánh master.

```
[tom@CentOS src]$ git push origin master
Total 0 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
5776472..64192f9 master -> master
```

Rebase các nhánh

Lệnh git rebase là một lệnh sáp nhập nhánh, nhưng điểm khác biệt ở đây là nó chỉnh sửa thứ tự của các commit.

Lệnh git merge cố gắng để đặt các commit từ nhánh khác lên trên đầu của HEAD của nhánh nội bộ hiện tại. Ví dụ, nhánh nội bộ của bạn có các commit A–>B–>C–>D và nhánh sáp nhập có các commit là A–>B–>X–>Y, thì sau đó lệnh git merge sẽ biến đổi nhánh nội bộ hiện tại thành một nhánh giống như A–>B–>C–>D–>X–>Y

Lệnh git rebase cố gắng để tìm ra gốc giữa nhánh nội bộ hiện tại và nhánh sáp nhập. Nó push các commit tới nhánh nội bộ bằng cách chỉnh sửa thứ tự của các commit trong nhánh nội bộ hiện tại. Ví dụ, hai nhánh có các commit như trên, thì lệnh git rebase sẽ chuyển đổi nhánh nội bộ hiện tại thành một nhánh giống như A>B>X>Y>C>D.

Khi có nhiều nhà lập trình cùng làm việc trên một repository từ xa, bạn không thể chỉnh sửa thứ tự của các commit trong repository này. Trong tình huống này, bạn có thể sử dụng hoạt động rebase để đặt các commit nội bộ của bạn trên phần đầu của các commit ở repository từ xa và bạn có thể push những thay đổi.

Xử lý Conflict trong Git

Thực hiện các thay đổi trong nhánh wchar_support

Jerry đang làm việc trên nhánh wchar_support. Anh ta thay đổi tên của các tính năng và sau khi kiểm tra, anh ta repository những thay đổi của anh ta.

```
[jerry@CentOS src]$ git branch
 master
* wchar_support
[jerry@CentOS src]$ git diff
```

Lệnh trên sẽ tạo ra kết quả sau:

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 8fb4b00..01ff4e0 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,7 +1,7 @@
#include <stdio.h>
#include <wchar.h>
-size_t w_strlen(const wchar_t *s)
+size_t my_wstrlen(const wchar_t *s)
{
```

```
   const wchar_t *p = s;
```

Sau khi thẩm tra lại code, anh ta repository những thay đổi vừa thực hiện.

```
[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Changed function name'
[wchar_support 3789fe8] Changed function name
1 files changed, 1 insertions(+), 1 deletions(-)

[jerry@CentOS src]$ git push origin wchar_support
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 409 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
64192f9..3789fe8 wchar_support -> wchar_support
```

Thực hiện các thay đổi trong nhánh master

Trong khi ấy trong nhánh master, Tom cũng thay đổi tên của cùng chức năng và push những thay đổi đó tới nhánh master.

```
[tom@CentOS src]$ git branch
* master
[tom@CentOS src]$ git diff
```

Lệnh trên sẽ tạo ra kết quả:

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 8fb4b00..52bec84 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,7 +1,8 @@
#include <stdio.h>
```

```
#include <wchar.h>
-size_t w_strlen(const wchar_t *s)
+/* wide character strlen fucntion */
+size_t my_wc_strlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Sau khi thẩm tra lại diff, anh ta repository những thay đổi này.

```
[tom@CentOS src]$ git status -s
M string_operations.c

[tom@CentOS src]$ git add string_operations.c

[tom@CentOS src]$ git commit -m 'Changed function name from w_strlen to my_wc_strlen'
[master ad4b530] Changed function name from w_strlen to my_wc_strlen
1 files changed, 2 insertions(+), 1 deletions(-)

[tom@CentOS src]$ git push origin master
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 470 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
64192f9..ad4b530 master -> master
```

Trên nhánh wchar_support, Jerry thực hiện chức năng strchr cho chuỗi ký tự mở rộng. Sau khi kiểm tra, anh ta repository và push những thay đổi này tới nhánh wchar_support.

```
[jerry@CentOS src]$ git branch
master
* wchar_support
[jerry@CentOS src]$ git diff
```

Lệnh trên sẽ tạo ra kết quả sau:

```
diff --git a/src/string_operations.c b/src/string_operations.c
index 01ff4e0..163a779 100644
```

```
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,6 +1,16 @@
#include <stdio.h>
#include <wchar.h>
+wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   while (*ws)
   {
      +
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Sau khi thẩm tra, anh ta repository những thay đổi này.

```
[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Addded strchr function for wide character string'
[wchar_support 9d201a9] Addded strchr function for wide character string
1 files changed, 10 insertions(+), 0 deletions(-)

[jerry@CentOS src]$ git push origin wchar_support
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 516 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
3789fe8..9d201a9 wchar_support -> wchar_support
```

Xử trí các conflict

Tom muốn thấy những gì mà Jerry đang làm trên nhánh tư nhân của cậu ta, vì thế anh ta cố gắng để pull những thay đổi mới nhất từ nhánh wchar_support, nhưng git hủy bỏ hoạt động này với thông báo lỗi sau:

```
[tom@CentOS src]$ git pull origin wchar_support
```

Lệnh trên sẽ tạo ra kết quả sau:

```
remote: Counting objects: 11, done.
63Git Tutorials
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (8/8), done.
From git.server.com:project
* branch
wchar_support -> FETCH_HEAD
Auto-merging src/string_operations.c
CONFLICT (content): Merge conflict in src/string_operations.c
Automatic merge failed; fix conflicts and then commit the result.
```

Xử lý các conflict

Từ thông báo lỗi, dễ nhận ra rằng có một conflict trong src/string_operations.c. Anh ta chạy lệnh git command để quan sát chi tiết hơn.

```
[tom@CentOS src]$ git diff
```

Lệnh trên sẽ tạo ra kết quả sau:

```
diff --cc src/string_operations.c
```

```
index 52bec84,163a779..0000000
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@@ -1,8 -1,17 +1,22 @@@
#include <stdio.h>
#include <wchar.h>
++<<<<<<< HEAD
+/* wide character strlen fucntion */
+size_t my_wc_strlen(const wchar_t *s)
++=======
+ wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   +
   while (*ws)
   {
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
+ size_t my_wstrlen(const wchar_t *s)
++>>>>>>>9d201a9c61bc4713f4095175f8954b642dae8f86
{
   const wchar_t *p = s;
```

Khi cả Tom và Jerry đều thay đổi tên của cùng một chức năng, git trong trạng thái rối loạn và nó hỏi người sử dụng giải quyết vấn đề này.

Tom quyết định giữ tên chức năng từ đề nghị của Jerry, nhưng anh ta giữ lời bình được thêm bởi anh ta. Sau dỡ bỏ mâu thuẫn, git diff sẽ trông như sau:

```
[tom@CentOS src]$ git diff
```

Lệnh trên sẽ tạo ra kết quả sau:

```
diff --cc src/string_operations.c
diff --cc src/string_operations.c
index 52bec84,163a779..0000000
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@@ -1,8 -1,17 +1,18 @@@
#include <stdio.h>
#include <wchar.h>
+ wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   while (*ws)
   {
      +
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
+/* wide character strlen fucntion */
- size_t my_wc_strlen(const wchar_t *s)
+ size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Khi Tom đã chỉnh sửa các file, đầu tiên anh ta phải repository những thay đổi này và sau đó anh ta có thể pull những thay đổi đó.

```
[tom@CentOS src]$ git commit -a -m 'Resolved conflict'
```

```
[master 6b1ac36] Resolved conflict


[tom@CentOS src]$ git pull origin wchar_support.
```

Tom đã giải quyết conflict, bây giờ hoạt động pull sẽ thành công.

Các Platform khác nhau trong Git

GNU/Linux và Mac OS sử dụng **line-feed (LF)**, hoặc một dòng mới khi ký tự kết thúc dòng (line-ending character) trong khi Windows sử dụng kết hợp **line-feed và carriage-return (LFCR)** để biểu diễn dòng ký tự kết thúc.

Để tránh các commit không cần thiết bởi vì sự khác nhau về dòng kết thúc, chúng ta phải định cấu hình trên client để viết cùng một dạng line-ending tới repository.

Đối với hệ điều hành Windows, chúng ta có thể định hình Git client để chuyển đổi line-ending sang định dạng CRLF trong khi kiểm tra, và chuyển đổi chúng trở lại định dạng LF trong suốt quá trình hoạt động commit. Các thiết lập sau là cần thiết để thực hiện.

```
[tom@CentOS project]$ git config --global core.autocrlf true
```

Đối với hệ điều hành GNU/Linux hoặc Mac OS, chúng ta có thể định hình client git chuyển đổi line-ending từ CRLF sang LF trong khi thực hiện hoạt động checkout.

```
[tom@CentOS project]$ git config --global core.autocrlf input
```

Repository trực tuyến trong Git

GitHub là mạng xã hội dành cho các nhà lập trình cho các dự án phát triển phần mềm sử dụng hệ thống quản lý git revision. Nó cũng có ứng dụng tiêu chuẩn GUI có cho tải trực tiếp (Windows, Mac, GNU/Linux) về máy từ các trang web. Nhưng trong phần hướng dẫn này, chúng tôi chỉ xem xét phần CLI.

Tạo repository GitHub

Bạn vào trang **github.com**. Nếu bạn đã có tài khoản **GitHub**, thì sau đó đăng nhập vào hệ thống bằng tài khoản hoặc tạo ra một tài khoản mới. Theo các bước trong**github.com** để tạo một repository mới.

Hoạt động push

Tom quyết định sử dụng máy chủ GitHub. Để bắt đầu một dự án mới, anh ta tạo một thư mục mới và một file trong đó.

```
[tom@CentOS]$ mkdir github_repo

[tom@CentOS]$ cd github_repo/

[tom@CentOS]$ vi hello.c

[tom@CentOS]$ make hello
cc hello.c -o hello

[tom@CentOS]$ ./hello
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Hello, World !!!
```

Sau khi thẩm tra lại code của mình, anh ta bắt đầu làm việc với thư mục với lệnh git init và commit các thay đổi nội bộ của anh ta.

```
[tom@CentOS]$ git init
Initialized empty Git repository in /home/tom/github_repo/.git/

[tom@CentOS]$ git status -s
?? hello
?? hello.c

[tom@CentOS]$ git add hello.c

[tom@CentOS]$ git status -s
A hello.c
?? hello

[tom@CentOS]$ git commit -m 'Initial commit'
```

Sau đó, anh ta thêm địa chỉ URL repository GitHub như là một điều khiển từ xa và push những thay đổi của mình tới repository từ xa này.

```
[tom@CentOS]$ git remote add origin https://github.com/kangralkar/testing_repo.git

[tom@CentOS]$ git push -u origin master
```

Hoạt động push sẽ yêu cầu tài khoản sử dụng và mật khẩu trên **GitHub** . Sau khi đăng nhập thành công, hoạt động push sẽ được thực hiện thành công.

Lệnh trên sẽ tạo ra kết quả sau:

```
Username for 'https://github.com': kangralkar
Password for 'https://kangralkar@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 214 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/kangralkar/test_repo.git
 * [new branch]      master -> master
 Branch master set up to track remote branch master from origin.
```

Bắt đầu từ giờ, Tom có thể push bất cứ thay đổi nào tới repository GitHub. Anh ta có thể sử dụng tất cả các lệnh được đề cập trong phần hướng dẫn này với repository GitHub.

Hoạt động pull

Tom đã push thành công tất cả những thay đổi tới repository GitHub. Bây giờ, các nhà lập trình khác có thể quan sát những thay đổi đó bằng cách thực hiện hoạt động mô phỏng hoặc cập nhật repository nội bộ của họ.

Jerry tạo một thư mục mới trong thư mục home và mô phỏng repository GitHub **GitHub**bằng cách sử dụng lệnh git clone.

```
[jerry@CentOS]$ pwd
/home/jerry

[jerry@CentOS]$ mkdir jerry_repo

[jerry@CentOS]$ git clone https://github.com/kangralkar/test_repo.git
```

Lệnh trên sẽ tạo ra kết quả sau:

```
Cloning into 'test_repo'...
remote: Counting objects: 3, done.
```

```
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
```

Anh ta thẩm tra nội dung thư mục bằng cách chạy lệnh ls.

```
[jerry@CentOS]$ ls
test_repo

[jerry@CentOS]$ ls test_repo/
hello.c
```

Tài liệu tham khảo về Git

ác tài liệu sau chứa các thông tin hữu ích về Git. Bạn nên sử dụng chúng để nâng cáo kiến thức của mình và hiểu sâu hơn về các chủ đề trong loạt bài hướng dẫn này.

Các đường link hữu ích về Git

- Tutorialspoint − Loạt bài hướng dẫn của chúng tôi xây dựng dựa trên nguồn này.
- Git Website - Trang web chính thức về Git
- Git Documentation - Tài liệu Git chính thức.
- Git Wikipedia - Wikipedia tham khảo về Git.