## Bắt đầu với Git

### Tạo kho lưu trữ đầu tiên của bạn, sau đó thêm và cam kết các tập tin

Tại dòng lệnh, trước tiên hãy xác minh rằng bạn đã cài đặt Git:

* Trên tất cả các hệ điều hành:

bash

Sao chép mã

git --version

* Trên các hệ điều hành tương tự UNIX:

bash

Sao chép mã

which git

Nếu không có gì được trả về, hoặc lệnh không được nhận diện, bạn có thể cần cài đặt Git trên hệ thống của mình bằng cách tải xuống và chạy trình cài đặt. Xem trang chủ Git để có hướng dẫn cài đặt rõ ràng và dễ hiểu.

Sau khi cài đặt Git, hãy cấu hình tên người dùng và địa chỉ email của bạn. Hãy thực hiện việc này trước khi thực hiện bất kỳ cam kết nào.

Khi Git đã được cài đặt, điều hướng đến thư mục mà bạn muốn đặt dưới sự kiểm soát phiên bản và tạo một kho lưu trữ Git trống:

bash

Sao chép mã

git init

Điều này sẽ tạo một thư mục ẩn, .git, chứa các cấu trúc cần thiết để Git hoạt động.

Tiếp theo, kiểm tra các tập tin mà Git sẽ thêm vào kho lưu trữ mới của bạn; bước này cần được chú ý đặc biệt:

bash

Sao chép mã

git status

Xem xét danh sách các tập tin kết quả; bạn có thể cho Git biết tập tin nào sẽ đưa vào kiểm soát phiên bản (tránh thêm các tập tin chứa thông tin bí mật như mật khẩu, hoặc các tập tin chỉ làm rối kho lưu trữ):

bash

Sao chép mã

git add <tên\_tập\_tin/thư\_mục #1> <tên\_tập\_tin/thư\_mục #2> < ... >

Nếu tất cả các tập tin trong danh sách nên được chia sẻ với tất cả những ai có quyền truy cập vào kho lưu trữ, một lệnh duy nhất sẽ thêm tất cả các tập tin trong thư mục hiện tại và các thư mục con của nó:

bash

Sao chép mã

git add .

Lệnh này sẽ "chuẩn bị" tất cả các tập tin để được đưa vào kiểm soát phiên bản, chuẩn bị cho chúng được cam kết trong cam kết đầu tiên của bạn.

Đối với các tập tin mà bạn không muốn đưa vào kiểm soát phiên bản, hãy tạo và điền nội dung cho một tập tin có tên là .gitignore trước khi thực hiện lệnh add.

Cam kết tất cả các tập tin đã được thêm, cùng với một thông điệp cam kết:

bash

Sao chép mã

git commit -m "Initial commit"

Điều này tạo ra một cam kết mới với thông điệp đã cho. Một cam kết giống như một lần lưu hoặc một ảnh chụp của toàn bộ dự án của bạn. Bạn có thể đẩy, hoặc tải lên, nó vào kho lưu trữ từ xa, và sau đó bạn có thể quay lại nếu cần.

Nếu bạn bỏ qua tham số -m, trình soạn thảo mặc định của bạn sẽ mở ra và bạn có thể chỉnh sửa và lưu thông điệp cam kết ở đó.

#### Thêm một kho lưu trữ từ xa

Để thêm một kho lưu trữ từ xa mới, hãy sử dụng lệnh git remote add trong terminal, trong thư mục nơi kho lưu trữ của bạn được lưu trữ. Lệnh git remote add nhận hai đối số:

1. Tên từ xa, ví dụ: origin
2. URL từ xa, ví dụ: https://<địa\_chỉ\_git\_service>/user/repo.git

bash

Sao chép mã

git remote add origin https://<địa\_chỉ\_git\_service>/owner/repository.git

LƯU Ý: Trước khi thêm kho lưu trữ từ xa, bạn phải tạo kho lưu trữ cần thiết trong dịch vụ git của bạn. Bạn sẽ có thể đẩy/kéo cam kết sau khi thêm kho lưu trữ từ xa.

#### Mục 1.2: Nhân bản (clone) một kho lưu trữ

Lệnh git clone được sử dụng để sao chép một kho lưu trữ Git hiện có từ một máy chủ về máy tính cục bộ. Ví dụ, để nhân bản một dự án trên GitHub:

bash

Sao chép mã

cd <đường\_dẫn\_nơi\_bạn\_muốn\_tạo\_thư\_mục>

git clone https://github.com/username/projectname.git

Để nhân bản một dự án trên BitBucket:

bash

Sao chép mã

cd <đường\_dẫn\_nơi\_bạn\_muốn\_tạo\_thư\_mục>

git clone https://yourusername@bitbucket.org/username/projectname.git

Điều này tạo ra một thư mục có tên là projectname trên máy tính cục bộ, chứa tất cả các tập tin trong kho lưu trữ Git từ xa. Điều này bao gồm các tập tin nguồn cho dự án, cũng như một thư mục con .git chứa toàn bộ lịch sử và cấu hình cho dự án.

Để chỉ định một tên khác cho thư mục, ví dụ: MyFolder:

bash

Sao chép mã

git clone https://github.com/username/projectname.git MyFolder

Hoặc để nhân bản vào thư mục hiện tại:

bash

Sao chép mã

git clone https://github.com/username/projectname.git .

Lưu ý:

1. Khi nhân bản vào một thư mục được chỉ định, thư mục phải trống hoặc không tồn tại.
2. Bạn cũng có thể sử dụng phiên bản ssh của lệnh:

bash

Sao chép mã

git clone git@github.com:username/projectname.git

Phiên bản https và phiên bản ssh là tương đương. Tuy nhiên, một số dịch vụ lưu trữ như GitHub khuyến nghị bạn sử dụng https thay vì ssh.

#### Mục 1.3: Chia sẻ mã nguồn

Để chia sẻ mã nguồn của bạn, bạn tạo một kho lưu trữ trên một máy chủ từ xa mà bạn sẽ sao chép kho lưu trữ cục bộ của bạn vào đó.

Để giảm thiểu việc sử dụng không gian trên máy chủ từ xa, bạn tạo một kho lưu trữ trần (bare repository): một kho lưu trữ chỉ có các đối tượng .git và không tạo một bản sao làm việc trong hệ thống tập tin. Thêm vào đó, bạn đặt máy chủ từ xa này làm máy chủ upstream để dễ dàng chia sẻ các cập nhật với các lập trình viên khác.

Trên máy chủ từ xa:

bash

Sao chép mã

git init --bare /path/to/repo.git

Trên máy tính cục bộ:

bash

Sao chép mã

git remote add origin ssh://username@server:/path/to/repo.git

(Lưu ý rằng ssh: chỉ là một cách có thể để truy cập kho lưu trữ từ xa.)

Bây giờ sao chép kho lưu trữ cục bộ của bạn lên máy chủ từ xa:

bash

Sao chép mã

git push --set-upstream origin master

Việc thêm --set-upstream (hoặc -u) tạo ra một tham chiếu upstream (theo dõi) được sử dụng bởi các lệnh Git không có tham số, ví dụ: git pull.

#### Mục 1.4: Cài đặt tên người dùng và email của bạn

Bạn cần thiết lập danh tính của mình trước khi tạo bất kỳ cam kết nào. Điều này sẽ cho phép các cam kết có tên và email của tác giả được liên kết đúng. Việc này không liên quan đến xác thực khi đẩy đến một kho lưu trữ từ xa (ví dụ: khi đẩy đến kho lưu trữ từ xa bằng tài khoản GitHub, BitBucket hoặc GitLab của bạn).

Để khai báo danh tính đó cho tất cả các kho lưu trữ, sử dụng git config --global. Điều này sẽ lưu thiết lập trong tệp .gitconfig của người dùng: ví dụ $HOME/.gitconfig hoặc đối với Windows, %USERPROFILE%\.gitconfig.

bash

Sao chép mã

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

git config --global user.email mail@example.com

Để khai báo danh tính cho một kho lưu trữ đơn lẻ, sử dụng git config bên trong một kho lưu trữ. Điều này sẽ lưu thiết lập bên trong kho lưu trữ cá nhân, trong tệp $GIT\_DIR/config, ví dụ /path/to/your/repo/.git/config.

bash

Sao chép mã

cd /path/to/my/repo

git config user.name "Tên đăng nhập của bạn tại công việc"

git config user.email mail\_at\_work@example.com

Các thiết lập lưu trong tệp config của kho lưu trữ sẽ ưu tiên hơn so với cấu hình toàn cục khi bạn sử dụng kho lưu trữ đó.

Mẹo: nếu bạn có các danh tính khác nhau (một cho dự án mã nguồn mở, một cho công việc, một cho kho lưu trữ cá nhân, ...), và bạn không muốn quên thiết lập danh tính đúng cho mỗi kho lưu trữ khác nhau mà bạn đang làm việc:

* Xóa một danh tính toàn cục

bash

Sao chép mã

git config --global --remove-section user.name

git config --global --remove-section user.email

Phiên bản ≥ 2.8

Để buộc git chỉ tìm danh tính của bạn trong các thiết lập của kho lưu trữ, không trong cấu hình toàn cục:

bash

Sao chép mã

git config --global user.useConfigOnly true

Bằng cách này, nếu bạn quên thiết lập user.name và user.email cho một kho lưu trữ cụ thể và cố gắng thực hiện cam kết, bạn sẽ thấy thông báo:

perl

Sao chép mã

no name was given and auto-detection is disabled

no email was given and auto-detection is disabled

#### Mục 1.5: Thiết lập upstream remote

Nếu bạn đã nhân bản một nhánh (ví dụ: một dự án mã nguồn mở trên Github), bạn có thể không có quyền đẩy vào kho lưu trữ upstream, vì vậy bạn cần cả nhánh của mình và có thể lấy dữ liệu từ kho lưu trữ upstream.

Trước tiên kiểm tra tên từ xa:

bash

Sao chép mã

git remote -v

perl

Sao chép mã

origin https://github.com/myusername/repo.git (fetch)

origin https://github.com/myusername/repo.git (push)

upstream # dòng này có thể hoặc không có ở đây

Nếu upstream đã có sẵn (nó có trong một số phiên bản Git), bạn cần thiết lập URL (hiện tại nó trống):

bash

Sao chép mã

git remote set-url upstream https://github.com/projectusername/repo.git

Nếu upstream không có ở đó, hoặc nếu bạn cũng muốn thêm nhánh của một người bạn/đồng nghiệp (hiện tại chúng không tồn tại):

bash

Sao chép mã

git remote add upstream https://github.com/projectusername/repo.git

git remote add dave https://github.com/dave/repo.git

#### Mục 1.6: Tìm hiểu về một lệnh

Để có thêm thông tin về bất kỳ lệnh git nào – tức là chi tiết về những gì lệnh làm, các tùy chọn có sẵn và tài liệu khác – sử dụng tùy chọn --help hoặc lệnh help.

Ví dụ, để có tất cả thông tin có sẵn về lệnh git diff, sử dụng:

bash

Sao chép mã

git diff --help

git help diff

Tương tự, để có tất cả thông tin có sẵn về lệnh status, sử dụng:

bash

Sao chép mã

git status --help

git help status

Nếu bạn chỉ muốn một trợ giúp nhanh hiển thị ý nghĩa của các cờ dòng lệnh được sử dụng nhiều nhất, sử dụng -h:

bash

Sao chép mã

git checkout -h

#### Mục 1.7: Thiết lập SSH cho Git

Nếu bạn đang sử dụng Windows, mở Git Bash. Nếu bạn đang sử dụng Mac hoặc Linux, mở Terminal của bạn.

Trước khi bạn tạo khóa SSH, bạn có thể kiểm tra xem bạn có bất kỳ khóa SSH hiện có nào không. Liệt kê nội dung của thư mục ~/.ssh của bạn:

bash

Sao chép mã

ls -al ~/.ssh

Danh sách tất cả các tập tin trong thư mục ~/.ssh của bạn.

Kiểm tra danh sách thư mục để xem bạn đã có một khóa SSH công khai chưa. Theo mặc định, tên tập tin của các khóa công khai là một trong những tên sau:

* id\_dsa.pub
* id\_ecdsa.pub
* id\_ed25519.pub
* id\_rsa.pub

Nếu bạn thấy một cặp khóa công khai và khóa riêng hiện có mà bạn muốn sử dụng trên Bitbucket, GitHub (hoặc tương tự), bạn có thể sao chép nội dung của tập tin id\_\*.pub.

Nếu không, bạn có thể tạo một cặp khóa công khai và khóa riêng mới với lệnh sau:

bash

Sao chép mã

ssh-keygen

Nhấn Enter hoặc Return để chấp nhận vị trí mặc định. Nhập và nhập lại cụm mật khẩu khi được nhắc, hoặc để trống.

Đảm bảo rằng khóa SSH của bạn được thêm vào ssh-agent. Khởi động ssh-agent trong nền nếu nó chưa chạy:

bash

Sao chép mã

eval "$(ssh-agent -s)"

Thêm khóa SSH của bạn vào ssh-agent. Lưu ý rằng bạn cần thay thế id\_rsa trong lệnh bằng tên của tập tin khóa riêng của bạn:

bash

Sao chép mã

ssh-add ~/.ssh/id\_rsa

Nếu bạn muốn thay đổi upstream của một kho lưu trữ hiện có từ HTTPS sang SSH, bạn có thể chạy lệnh sau:

bash

Sao chép mã

git remote set-url origin ssh://git@bitbucket.server.com:7999/projects/your\_project.git

Để nhân bản một kho lưu trữ mới qua SSH, bạn có thể chạy lệnh sau:

bash

Sao chép mã

git clone ssh://git@bitbucket.server.com:7999/projects/your\_project.git

#### Mục 1.8: Cài đặt Git

Bây giờ chúng ta hãy bắt đầu sử dụng Git. Trước tiên, bạn phải cài đặt nó. Bạn có thể cài đặt Git theo nhiều cách; hai cách chính là cài đặt từ mã nguồn hoặc cài đặt gói có sẵn cho nền tảng của bạn.

**Cài đặt từ mã nguồn**

Nếu bạn có thể, việc cài đặt Git từ mã nguồn thường hữu ích, vì bạn sẽ có được phiên bản mới nhất. Mỗi phiên bản của Git thường bao gồm các cải tiến giao diện người dùng hữu ích, vì vậy việc có phiên bản mới nhất thường là lựa chọn tốt nhất nếu bạn cảm thấy thoải mái khi biên dịch phần mềm từ mã nguồn. Cũng có trường hợp nhiều bản phân phối Linux chứa các gói rất cũ; do đó, trừ khi bạn đang sử dụng bản phân phối rất cập nhật hoặc sử dụng backports, việc cài đặt từ mã nguồn có thể là lựa chọn tốt nhất.

Để cài đặt Git, bạn cần có các thư viện sau mà Git phụ thuộc vào: curl, zlib, openssl, expat, và libiconv. Ví dụ, nếu bạn đang sử dụng một hệ thống có yum (như Fedora) hoặc apt-get (như một hệ thống dựa trên Debian), bạn có thể sử dụng một trong những lệnh sau để cài đặt tất cả các phụ thuộc:

bash

Sao chép mã

yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel

apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev

Khi bạn đã có tất cả các phụ thuộc cần thiết, bạn có thể tiến hành lấy bản snapshot mới nhất từ trang web Git: <http://git-scm.com/download>

Sau đó, biên dịch và cài đặt:

bash

Sao chép mã

tar -zxf git-1.7.2.2.tar.gz

cd git-1.7.2.2

make prefix=/usr/local all

sudo make prefix=/usr/local install

Sau khi hoàn tất, bạn cũng có thể lấy Git qua chính Git để cập nhật:

bash

Sao chép mã

git clone git://git.kernel.org/pub/scm/git/git.git

**Cài đặt trên Linux**

Nếu bạn muốn cài đặt Git trên Linux qua một trình cài đặt nhị phân, bạn thường có thể làm như vậy thông qua công cụ quản lý gói cơ bản đi kèm với bản phân phối của bạn. Nếu bạn đang sử dụng Fedora, bạn có thể sử dụng yum:

bash

Sao chép mã

yum install git

Hoặc nếu bạn đang sử dụng một bản phân phối dựa trên Debian như Ubuntu, hãy thử apt-get:

bash

Sao chép mã

apt-get install git

**Cài đặt trên Mac**

Có ba cách dễ dàng để cài đặt Git trên Mac. Cách dễ nhất là sử dụng trình cài đặt Git đồ họa, bạn có thể tải xuống từ trang SourceForge: <http://sourceforge.net/projects/git-osx-installer/>

Cách khác là cài đặt Git qua MacPorts (<http://www.macports.org>). Nếu bạn đã cài đặt MacPorts, cài đặt Git qua:

bash

Sao chép mã

sudo port install git +svn +doc +bash\_completion +gitweb

Bạn không cần thêm tất cả các tiện ích bổ sung, nhưng có lẽ bạn sẽ muốn bao gồm +svn trong trường hợp bạn cần sử dụng Git với các kho lưu trữ Subversion.

Homebrew (<http://brew.sh/>) là một lựa chọn khác để cài đặt Git. Nếu bạn đã cài đặt Homebrew, cài đặt Git qua:

bash

Sao chép mã

brew install git

**Cài đặt trên Windows**

Cài đặt Git trên Windows rất dễ dàng. Dự án msysGit có một trong những quy trình cài đặt dễ dàng nhất. Chỉ cần tải xuống tệp trình cài đặt exe từ trang GitHub, và chạy nó: <http://msysgit.github.io>

Sau khi cài đặt, bạn có cả phiên bản dòng lệnh (bao gồm một trình khách SSH sẽ hữu ích sau này) và GUI tiêu chuẩn.

Lưu ý về việc sử dụng trên Windows: bạn nên sử dụng Git với shell msysGit được cung cấp (kiểu Unix), nó cho phép sử dụng các dòng lệnh phức tạp được đưa ra trong sách này. Nếu bạn cần, vì lý do nào đó, sử dụng shell Windows gốc/console dòng lệnh, bạn phải sử dụng dấu ngoặc kép thay vì dấu ngoặc đơn (cho các tham số có dấu cách trong đó) và bạn phải trích dẫn các tham số kết thúc bằng dấu mũ (^) nếu chúng là cuối cùng trên dòng, vì nó là ký hiệu tiếp tục trên Windows.

## Chương 2: Duyệt lịch sử

| **Tham số** | **Giải thích** |
| --- | --- |
| -q, --quiet | Yên lặng, ngăn xuất kết quả khác biệt |
| --source | Hiển thị nguồn của cam kết |
| --use-mailmap | Sử dụng tệp mail map (thay đổi thông tin người dùng khi cam kết) |
| --decorate=[...] | Tùy chọn trang trí |
| --L <n,m  > | Hiển thị log cho phạm vi cụ thể của các dòng trong một tệp, bắt đầu từ 1. Bắt đầu từ dòng n, đi đến dòng m. Cũng hiển thị khác biệt. |
| --show-signature | Hiển thị chữ ký của các cam kết đã ký |
| -i, --regexp-ignore-case | Khớp biểu thức chính quy giới hạn mẫu mà không quan tâm đến chữ hoa chữ thường |

#### Mục 2.1: Git Log "Thông thường"

git log sẽ hiển thị tất cả các cam kết của bạn với tác giả và mã băm. Điều này sẽ được hiển thị trên nhiều dòng cho mỗi cam kết. (Nếu bạn muốn hiển thị một dòng duy nhất cho mỗi cam kết, hãy xem phần oneline). Sử dụng phím q để thoát khỏi log.

Mặc định, không có tham số nào, git log liệt kê các cam kết được thực hiện trong kho lưu trữ đó theo thứ tự ngược thời gian - nghĩa là các cam kết mới nhất sẽ xuất hiện trước. Như bạn có thể thấy, lệnh này liệt kê mỗi cam kết với mã băm SHA-1, tên và email của tác giả, ngày viết, và thông điệp cam kết.

Ví dụ (từ kho lưu trữ Free Code Camp):

plaintext

Sao chép mã

commit 87ef97f59e2a2f4dc425982f76f14a57d0900bcf

Merge: e50ff0d eb8b729

Author: Brian

Date: Thu Mar 24 15:52:07 2016 -0700

Merge pull request #7724 from BKinahan/fix/where-art-thou

Fix 'its' typo in Where Art Thou description

commit eb8b7298d516ea20a4aadb9797c7b6fd5af27ea5

Author: BKinahan

Date: Thu Mar 24 21:11:36 2016 +0000

Fix 'its' typo in Where Art Thou description

commit e50ff0d249705f41f55cd435f317dcfd02590ee7

Merge: 6b01875 2652d04

Author: Mrugesh Mohapatra

Date: Thu Mar 24 14:26:04 2016 +0530

Merge pull request #7718 from deathsythe47/fix/unnecessary-comma

Remove unnecessary comma from CONTRIBUTING.md

Nếu bạn muốn giới hạn lệnh của mình chỉ hiển thị log của các cam kết gần nhất, bạn có thể đơn giản truyền vào tham số. Ví dụ, nếu bạn muốn liệt kê log của 2 cam kết gần nhất:

bash

Sao chép mã

git log -2

#### Mục 2.2: Log Đẹp Hơn

Để xem log trong cấu trúc dạng đồ thị đẹp hơn, sử dụng:

bash

Sao chép mã

git log --decorate --oneline --graph

Ví dụ đầu ra:

plaintext

Sao chép mã

\* e0c1cea (HEAD -> maint, tag: v2.9.3, origin/maint) Git 2.9.3

\* 9b601ea Merge branch 'jk/difftool-in-subdir' into maint

|\

| \* 32b8c58 difftool: use Git::\* functions instead of passing around state

| \* 98f917e difftool: avoid $GIT\_DIR and $GIT\_WORK\_TREE

| \* 9ec26e7 difftool: fix argument handling in subdirs

\* | f4fd627 Merge branch 'jk/reset-ident-time-per-commit' into maint

...

Vì đây là một lệnh khá dài, bạn có thể tạo một bí danh (alias):

bash

Sao chép mã

git config --global alias.lol "log --decorate --oneline --graph"

Để sử dụng phiên bản alias:

* Lịch sử của nhánh hiện tại:

bash

Sao chép mã

git lol

* Lịch sử kết hợp của nhánh đang hoạt động (HEAD), nhánh develop và nhánh origin/master:

bash

Sao chép mã

git lol HEAD develop origin/master

* Lịch sử kết hợp của tất cả mọi thứ trong kho lưu trữ của bạn:

bash

Sao chép mã

git lol --all

#### Mục 2.3: Tô Màu Log

bash

Sao chép mã

git log --graph --pretty=format:'%C(red)%h%Creset -%C(yellow)%d%Creset %s %C(green)(%cr) %C(yellow)<%an>%Creset'

Tùy chọn format cho phép bạn chỉ định định dạng đầu ra log của riêng mình:

Thông tin chi tiết:

* %C(color\_name): Tùy chọn màu sắc cho đầu ra tiếp theo
* %h hoặc %H: Rút gọn mã cam kết (sử dụng %H cho mã đầy đủ)
* %Creset: Đặt lại màu về màu mặc định của terminal
* %d: Tên tham chiếu
* %s: Chủ đề [thông điệp cam kết]
* %cr: Ngày cam kết, so với ngày hiện tại
* %an: Tên tác giả

#### Mục 2.4: Log Một Dòng

bash

Sao chép mã

git log --oneline

sẽ hiển thị tất cả các cam kết của bạn với chỉ phần đầu của mã băm và thông điệp cam kết. Mỗi cam kết sẽ nằm trên một dòng duy nhất, như cờ oneline gợi ý.

Tùy chọn oneline in mỗi cam kết trên một dòng, hữu ích nếu bạn đang xem xét nhiều cam kết.

Ví dụ (từ kho lưu trữ Free Code Camp, với cùng một đoạn mã từ ví dụ khác):

plaintext

Sao chép mã

87ef97f Merge pull request #7724 from BKinahan/fix/where-art-thou

eb8b729 Fix 'its' typo in Where Art Thou description

e50ff0d Merge pull request #7718 from deathsythe47/fix/unnecessary-comma

2652d04 Remove unnecessary comma from CONTRIBUTING.md

6b01875 Merge pull request #7667 from zerkms/patch-1

766f088 Fixed assignment operator terminology

d1e2468 Merge pull request #7690 from BKinahan/fix/unsubscribe-crash

bed9de2 Merge pull request #7657 from Rafase282/fix/

Nếu bạn muốn giới hạn lệnh của mình chỉ hiển thị log của các cam kết gần nhất, bạn có thể đơn giản truyền vào tham số. Ví dụ, nếu bạn muốn liệt kê log của 2 cam kết gần nhất:

bash

Sao chép mã

git log -2 --oneline

#### Mục 2.5: Tìm Kiếm Log

bash

Sao chép mã

git log -S"#define SAMPLES"

Tìm kiếm sự thêm vào hoặc loại bỏ của chuỗi cụ thể hoặc chuỗi khớp với REGEXP đã cung cấp. Trong trường hợp này, chúng ta đang tìm kiếm sự thêm vào/loại bỏ của chuỗi #define SAMPLES.

Ví dụ:

plaintext

Sao chép mã

+#define SAMPLES 100000

hoặc

plaintext

Sao chép mã

-#define SAMPLES 100000

bash

Sao chép mã

git log -G"#define SAMPLES"

Tìm kiếm sự thay đổi trong các dòng chứa chuỗi cụ thể hoặc chuỗi khớp với REGEXP đã cung cấp.

Ví dụ:

plaintext

Sao chép mã

-#define SAMPLES 100000

+#define SAMPLES 100000000

#### Mục 2.6: Liệt Kê Tất Cả Đóng Góp Theo Tên Tác Giả

git shortlog tóm tắt git log và nhóm theo tác giả. Nếu không có tham số nào được đưa ra, danh sách tất cả các cam kết được thực hiện bởi từng người cam kết sẽ được hiển thị theo thứ tự thời gian.

bash

Sao chép mã

git shortlog

Ví dụ:

plaintext

Sao chép mã

Committer 1 (<number\_of\_commits>):

Commit Message 1

Commit Message 2

...

Committer 2 (<number\_of\_commits>):

Commit Message 1

Commit Message 2

...

Để chỉ đơn giản xem số lượng cam kết và bỏ qua mô tả cam kết, sử dụng tùy chọn summary:

bash

Sao chép mã

git shortlog -s

Ví dụ:

plaintext

Sao chép mã

<number\_of\_commits> Committer 1

<number\_of\_commits> Committer 2

Để sắp xếp đầu ra theo số lượng cam kết thay vì theo thứ tự chữ cái của tên người cam kết, sử dụng tùy chọn numbered:

bash

Sao chép mã

git shortlog -n

Để thêm email của người cam kết, sử dụng tùy chọn email:

bash

Sao chép mã

git shortlog -e

Tùy chọn định dạng tùy chỉnh cũng có thể được cung cấp nếu bạn muốn hiển thị thông tin khác ngoài chủ đề cam kết:

bash

Sao chép mã

git shortlog --format

Xem phần Tô Màu Log ở trên để biết thêm thông tin về điều này.

#### Mục 2.7: Tìm Kiếm Chuỗi Cam Kết Trong Git Log

Tìm kiếm git log sử dụng một chuỗi trong log:

bash

Sao chép mã

git log [options] --grep "search\_string"

Ví dụ:

bash

Sao chép mã

git log --all --grep "removed file"

Sẽ tìm kiếm chuỗi removed file trong tất cả các log trên tất cả các nhánh.

Bắt đầu từ git 2.4+, tìm kiếm có thể bị đảo ngược bằng cách sử dụng tùy chọn --invert-grep. Ví dụ:

bash

Sao chép mã

git log --grep="add file" --invert-grep

Sẽ hiển thị tất cả các cam kết không chứa add file.

#### Mục 2.8: Log Cho Một Phạm Vi Dòng Trong Một Tệp

bash

Sao chép mã

git log -L 1,20:index.html

Ví dụ:

plaintext

Sao chép mã

commit 6a57fde739de66293231f6204cbd8b2feca3a869

Author: John Doe <john@doe.com>

Date: Tue Mar 22 16:33:42 2016 -0500

commit message

diff --git a/index.html b/index.html

--- a/index.html

+++ b/index.html

@@ -1,17 +1,20 @@

<!DOCTYPE HTML>

<html>

- <head>

- <meta charset="utf-8">

+

+<head>

+ <meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

#### Mục 2.9: Lọc Log

bash

Sao chép mã

git log --after '3 days ago'

Các ngày cụ thể cũng hoạt động:

bash

Sao chép mã

git log --after 2016-05-01

Giống như các lệnh và cờ khác chấp nhận tham số ngày, định dạng ngày được cho phép là định dạng được hỗ trợ bởi GNU date (rất linh hoạt).

Một bí danh cho --after là --since.

Các cờ tương ứng cho ngược lại cũng tồn tại: --before và --until.

Bạn cũng có thể lọc log theo tác giả. Ví dụ:

bash

Sao chép mã

git log --author=author

#### Mục 2.10: Log Với Thay Đổi Trong Dòng

Để xem log với các thay đổi trong dòng, sử dụng tùy chọn -p hoặc --patch.

bash

Sao chép mã

git log --patch

Ví dụ (từ kho lưu trữ Trello Scientist):

plaintext

Sao chép mã

commit 8ea1452aca481a837d9504f1b2c77ad013367d25

Author: Raymond Chou <info@raychou.io>

Date: Wed Mar 2 10:35:25 2016 -0800

fix readme error link

diff --git a/README.md b/README.md

index 1120a00..9bef0ce 100644

--- a/README.md

+++ b/README.md

@@ -134,7 +134,7 @@ the control function threw, but \*after\* testing the other functions and

readying

the logging. The criteria for matching errors is based on the constructor and

message.

-You can find this full example at [examples/errors.js](examples/error.js).

+You can find this full example at [examples/errors.js](examples/errors.js).

#### Mục 2.11: Log Hiển Thị Các Tệp Đã Cam Kết

bash

Sao chép mã

git log --stat

Ví dụ:

plaintext

Sao chép mã

commit 4ded994d7fc501451fa6e233361887a2365b91d1

Author: Manassés Souza <manasses.inatel@gmail.com>

Date: Mon Jun 6 21:32:30 2016 -0300

MercadoLibre java-sdk dependency

mltracking-poc/.gitignore | 1 +

mltracking-poc/pom.xml | 14 ++++++++++++--

2 files changed, 13 insertions(+), 2 deletions(-)

commit 506fff56190f75bc051248770fb0bcd976e3f9a5

Author: Manassés Souza <manasses.inatel@gmail.com>

Date: Sat Jun 4 12:35:16 2016 -0300

[manasses] generated by SpringBoot initializr

.gitignore | 42 ++++++++++++

mltracking-poc/mvnw | 233 +++++++++++++++++++++++++++++++++++++++++++++++++++++

mltracking-poc/mvnw.cmd | 145 ++++++++++++++++++++++++++++++++++++++

mltracking-poc/pom.xml | 74 +++++++++++++++++++++

mltracking-poc/src/main/java/br/com/mls/mltracking/MltrackingPocApplication.java | 12 ++++

mltracking-poc/src/main/resources/application.properties | 0

mltracking-poc/src/test/java/br/com/mls/mltracking/MltrackingPocApplicationTests.java | 18 +++++

7 files changed, 524 insertions(+)

#### Mục 2.12: Hiển Thị Nội Dung Một Cam Kết

Sử dụng git show chúng ta có thể xem một cam kết duy nhất

bash

Sao chép mã

git show 48c83b3

git show 48c83b3690dfc7b0e622fd220f8f37c26a77c934

Ví dụ:

plaintext

Sao chép mã

commit 48c83b3690dfc7b0e622fd220f8f37c26a77c934

Author: Matt Clark <mrclark32493@gmail.com>

Date: Wed May 4 18:26:40 2016 -0400

The commit message will be shown here.

diff --git a/src/main/java/org/jdm/api/jenkins/BuildStatus.java

b/src/main/java/org/jdm/api/jenkins/BuildStatus.java

index 0b57e4a..fa8e6a5 100755

--- a/src/main/java/org/jdm/api/jenkins/BuildStatus.java

+++ b/src/main/java/org/jdm/api/jenkins/BuildStatus.java

@@ -50,7 +50,7 @@ public enum BuildStatus {

colorMap.put(BuildStatus.UNSTABLE, Color.decode( "#FFFF55" ));

- colorMap.put(BuildStatus.SUCCESS, Color.decode( "#55FF55" ));

+ colorMap.put(BuildStatus.SUCCESS, Color.decode( "#33CC33" ));

colorMap.put(BuildStatus.BUILDING, Color.decode( "#5555FF" ));

#### Mục 2.13: Git Log Giữa Hai Nhánh

git log master..foo sẽ hiển thị các cam kết có trên nhánh foo và không có trên nhánh master. Hữu ích để xem những cam kết nào bạn đã thêm kể từ khi tạo nhánh!

#### Mục 2.14: Một Dòng Hiển Thị Tên Người Cam Kết và Thời Gian Kể Từ Cam Kết

plaintext

Sao chép mã

tree = log --oneline --decorate --source --pretty=format:'"%Cblue %h %Cgreen %ar %Cblue %an %C(yellow) %d %Creset %s"' --all --graph

Ví dụ:

plaintext

Sao chép mã

\* 40554ac 3 months ago Alexander Zolotov Merge pull request #95 from gmandnepr/external\_plugins

|\

| \* e509f61 3 months ago Ievgen Degtiarenko Documenting new property

| \* 46d4cb6 3 months ago Ievgen Degtiarenko Running idea with external plugins

| \* 6253da4 3 months ago Ievgen Degtiarenko Resolve external plugin classes

| \* 9fdb4e7 3 months ago Ievgen Degtiarenko Keep original artifact name as this may be important for intellij

| \* 22e82e4 3 months ago Ievgen Degtiarenko Declaring external plugin in intellij section

|/

\* bc3d2cb 3 months ago Alexander Zolotov Ignore DTD in plugin.xml

## Chương 3: Làm việc với các kho lưu trữ từ xa

#### Mục 3.1: Xóa một nhánh từ xa

Để xóa một nhánh từ xa trong Git:

bash

Sao chép mã

git push [tên\_remote] --delete [tên\_nhánh]

hoặc

bash

Sao chép mã

git push [tên\_remote] :[tên\_nhánh]

#### Mục 3.2: Thay đổi URL của Remote Git

Kiểm tra remote hiện có:

bash

Sao chép mã

git remote -v

# origin https://github.com/username/repo.git (fetch)

# origin https://github.com/username/repo.git (push)

Thay đổi URL của kho lưu trữ:

bash

Sao chép mã

git remote set-url origin https://github.com/username/repo2.git

# Thay đổi URL của remote 'origin'

Xác minh URL remote mới:

bash

Sao chép mã

git remote -v

# origin https://github.com/username/repo2.git (fetch)

# origin https://github.com/username/repo2.git (push)

#### Mục 3.3: Liệt kê các Remote hiện có

Liệt kê tất cả các remote hiện có liên kết với kho lưu trữ này:

bash

Sao chép mã

git remote

Liệt kê chi tiết tất cả các remote hiện có liên kết với kho lưu trữ này bao gồm cả URL fetch và push:

bash

Sao chép mã

git remote --verbose

hoặc đơn giản là:

bash

Sao chép mã

git remote -v

#### Mục 3.4: Loại bỏ các bản sao cục bộ của các nhánh từ xa đã bị xóa

Nếu một nhánh từ xa đã bị xóa, kho lưu trữ cục bộ của bạn cần được thông báo để loại bỏ tham chiếu tới nó.

Để loại bỏ các nhánh đã bị xóa từ một remote cụ thể:

bash

Sao chép mã

git fetch [tên\_remote] --prune

Để loại bỏ các nhánh đã bị xóa từ tất cả các remotes:

bash

Sao chép mã

git fetch --all --prune

#### Mục 3.5: Cập nhật từ kho lưu trữ Upstream

Giả sử bạn đã thiết lập upstream (như trong phần "thiết lập kho lưu trữ upstream"):

bash

Sao chép mã

git fetch [tên\_remote]

git merge [tên\_remote]/[tên\_nhánh]

Lệnh pull kết hợp một lệnh fetch và một lệnh merge:

bash

Sao chép mã

git pull

Lệnh pull với cờ --rebase kết hợp một lệnh fetch và một lệnh rebase thay vì merge:

bash

Sao chép mã

git pull --rebase [tên\_remote] [tên\_nhánh]

#### Mục 3.6: ls-remote

git ls-remote là một lệnh đặc biệt cho phép bạn truy vấn một kho lưu trữ từ xa mà không cần phải clone/fetch nó trước. Nó sẽ liệt kê các refs/heads và refs/tags của kho lưu trữ từ xa đó.

Bạn sẽ thấy đôi khi có refs/tags/v0.1.6 và refs/tags/v0.1.6^{}: ký hiệu ^{} để liệt kê tag đã giải thích (tức là cam kết mà tag đang trỏ tới).

Kể từ git 2.8 (tháng 3 năm 2016), bạn có thể tránh mục nhập đôi cho một tag và liệt kê trực tiếp các tag đã giải thích với:

bash

Sao chép mã

git ls-remote --ref

Nó cũng có thể giúp giải quyết URL thực tế được sử dụng bởi một kho lưu trữ từ xa khi bạn có thiết lập cấu hình url.<base>.insteadOf.

Nếu git remote --get-url <tên\_remote> trả về https://server.com/user/repo, và bạn đã thiết lập git config url.ssh://git@server.com:.insteadOf https://server.com/:

bash

Sao chép mã

git ls-remote --get-url <tên\_remote>

ssh://git@server.com:user/repo

#### Mục 3.7: Thêm một kho lưu trữ từ xa mới

bash

Sao chép mã

git remote add upstream [url-kho-luu-tru-git]

Thêm kho lưu trữ git từ xa được biểu thị bởi url-kho-luu-tru-git làm remote mới có tên upstream vào kho lưu trữ git.

#### Mục 3.8: Thiết lập Upstream cho một nhánh mới

Bạn có thể tạo một nhánh mới và chuyển sang nó bằng:

bash

Sao chép mã

git checkout -b AP-57

Sau khi sử dụng git checkout để tạo một nhánh mới, bạn cần thiết lập upstream cho nhánh đó để có thể đẩy lên:

bash

Sao chép mã

git push --set-upstream origin AP-57

Sau đó, bạn có thể sử dụng git push khi đang ở trên nhánh đó.

#### Mục 3.9: Bắt đầu

Cú pháp để đẩy lên một nhánh từ xa:

bash

Sao chép mã

git push <tên\_remote> <tên\_nhánh>

Ví dụ:

bash

Sao chép mã

git push origin master

#### Mục 3.10: Đổi tên một Remote

Để đổi tên remote, sử dụng lệnh git remote rename. Lệnh này nhận hai đối số:

* Tên remote hiện tại, ví dụ: origin
* Tên mới cho remote, ví dụ: destination

Lấy tên remote hiện có:

bash

Sao chép mã

git remote

# origin

Kiểm tra remote hiện có với URL:

bash

Sao chép mã

git remote -v

# origin https://github.com/username/repo.git (fetch)

# origin https://github.com/username/repo.git (push)

Đổi tên remote:

bash

Sao chép mã

git remote rename origin destination

# Đổi tên remote từ 'origin' thành 'destination'

Xác minh tên mới:

bash

Sao chép mã

git remote -v

# destination https://github.com/username/repo.git (fetch)

# destination https://github.com/username/repo.git (push)

#### Các Lỗi Có Thể Xảy Ra

1. Không thể đổi tên phần cấu hình 'remote.[tên\_cũ]' thành 'remote.[tên\_mới]'
   * Lỗi này có nghĩa là remote bạn thử đổi tên không tồn tại.
2. Remote [tên\_mới] đã tồn tại.
   * Thông báo lỗi này tự giải thích.

#### Mục 3.11: Hiển thị thông tin về một Remote cụ thể

Xuất một số thông tin về một remote đã biết: origin

bash

Sao chép mã

git remote show origin

Chỉ in URL của remote:

bash

Sao chép mã

git config --get remote.origin.url

Với phiên bản 2.7+, bạn cũng có thể làm điều này, có thể tốt hơn so với lệnh trên sử dụng lệnh config:

bash

Sao chép mã

git remote get-url origin

#### Mục 3.12: Thiết lập URL cho một Remote cụ thể

Bạn có thể thay đổi URL của một remote hiện có bằng lệnh:

bash

Sao chép mã

git remote set-url <tên\_remote> <url>

#### Mục 3.13: Lấy URL cho một Remote cụ thể

Bạn có thể lấy URL của một remote hiện có bằng cách sử dụng lệnh:

bash

Sao chép mã

git remote get-url <tên>

Mặc định, điều này sẽ là:

bash

Sao chép mã

git remote get-url origin

#### Mục 3.14: Thay đổi một kho lưu trữ từ xa

Để thay đổi URL của kho lưu trữ mà bạn muốn remote của bạn trỏ tới, bạn có thể sử dụng tùy chọn set-url, như sau:

bash

Sao chép mã

git remote set-url <tên\_remote> <url\_kho\_luu\_tru\_từ\_xa>

Ví dụ:

bash

Sao chép mã

git remote set-url heroku https://git.heroku.com/fictional-remote-repository.git

## Chương 4: Staging (Tạm thời)

#### Mục 4.1: Staging tất cả các thay đổi trong tệp

bash

Sao chép mã

git add -A

Phiên bản ≥ 2.0

bash

Sao chép mã

git add .

Trong phiên bản 2.x, git add . sẽ staging tất cả các thay đổi trong thư mục hiện tại và tất cả các thư mục con của nó. Tuy nhiên, trong phiên bản 1.x, nó chỉ staging các tệp mới và tệp đã chỉnh sửa, không bao gồm các tệp bị xóa. Sử dụng git add -A hoặc lệnh tương đương git add --all để staging tất cả các thay đổi trong bất kỳ phiên bản nào của git.

#### Mục 4.2: Bỏ staging một tệp chứa thay đổi

bash

Sao chép mã

git reset <filePath>

#### Mục 4.3: Thêm thay đổi theo từng phần

Bạn có thể xem những "hunks" (phần thay đổi) nào sẽ được staging để commit bằng cách sử dụng cờ patch:

bash

Sao chép mã

git add -p

hoặc

bash

Sao chép mã

git add --patch

Điều này mở ra một giao diện tương tác cho phép bạn xem các khác biệt và quyết định có bao gồm chúng hay không.

Tùy chọn khi staging một hunk:

* y staging hunk này cho lần commit tiếp theo
* n không staging hunk này cho lần commit tiếp theo
* q thoát; không staging hunk này hoặc bất kỳ hunk nào còn lại
* a staging hunk này và tất cả các hunk sau đó trong tệp
* d không staging hunk này hoặc bất kỳ hunk nào sau đó trong tệp
* g chọn một hunk để chuyển đến
* / tìm kiếm một hunk khớp với regex đã cho
* j để hunk này không quyết định, xem hunk chưa quyết định tiếp theo
* J để hunk này không quyết định, xem hunk tiếp theo
* k để hunk này không quyết định, xem hunk chưa quyết định trước đó
* K để hunk này không quyết định, xem hunk trước đó
* s chia hunk hiện tại thành các hunk nhỏ hơn
* e chỉnh sửa thủ công hunk hiện tại
* ? in trợ giúp về hunk

Điều này giúp dễ dàng bắt những thay đổi mà bạn không muốn commit.

Bạn cũng có thể mở thông qua git add --interactive và chọn p.

#### Mục 4.4: Thêm thay đổi tương tác

git add -i (hoặc --interactive) sẽ cung cấp cho bạn một giao diện tương tác để chỉnh sửa index, chuẩn bị những gì bạn muốn có trong lần commit tiếp theo. Bạn có thể thêm và xóa các thay đổi trong toàn bộ tệp, thêm các tệp chưa được theo dõi và xóa các tệp khỏi việc theo dõi, nhưng cũng có thể chọn các phần thay đổi để đưa vào index, bằng cách chọn các phần thay đổi, chia nhỏ các phần đó, hoặc thậm chí chỉnh sửa khác biệt. Nhiều công cụ commit đồ họa cho Git (như git gui) bao gồm tính năng này; điều này có thể dễ sử dụng hơn so với phiên bản dòng lệnh.

Nó rất hữu ích khi:

1. Bạn có các thay đổi lẫn lộn trong thư mục làm việc mà bạn muốn đặt trong các commit riêng biệt, không phải tất cả trong một commit duy nhất.
2. Bạn đang ở giữa một rebase tương tác và muốn chia nhỏ commit quá lớn.

bash

Sao chép mã

$ git add -i

staged unstaged path

1: unchanged +4/-4 index.js

2: +1/-0 nothing package.json

\*\*\* Commands \*\*\*

1: status 2: update 3: revert 4: add untracked

5: patch 6: diff 7: quit 8: help

What now>

Nửa trên của đầu ra này hiển thị trạng thái hiện tại của index được chia thành các cột staged và unstaged:

1. index.js đã thêm 4 dòng và xóa 4 dòng. Hiện tại nó không được staging, vì trạng thái hiện tại báo cáo "unchanged." Khi tệp này được staging, phần +4/-4 sẽ được chuyển đến cột staged và cột unstaged sẽ đọc "nothing."
2. package.json đã thêm một dòng và đã được staging. Không có thay đổi nào thêm kể từ khi nó được staging như đã được chỉ ra bởi dòng "nothing" dưới cột unstaged.

Nửa dưới hiển thị những gì bạn có thể làm. Hoặc nhập một số (1-8) hoặc một chữ cái (s, u, r, a, p, d, q, h).

* status hiển thị đầu ra giống như phần trên của đầu ra trên.
* update cho phép bạn thực hiện các thay đổi bổ sung vào các commit đã staging với cú pháp bổ sung.
* revert sẽ hoàn nguyên thông tin commit đã staging về HEAD.
* add untracked cho phép bạn thêm các đường dẫn tệp chưa được theo dõi trước đó bởi kiểm soát phiên bản.
* patch cho phép chọn một đường dẫn từ đầu ra tương tự như trạng thái để phân tích thêm.
* diff hiển thị những gì sẽ được commit.
* quit thoát lệnh.
* help hiển thị trợ giúp thêm về việc sử dụng lệnh này.

#### Mục 4.5: Hiển thị các thay đổi đã staging

Để hiển thị các phần đã staging cho commit:

bash

Sao chép mã

git diff --cached

#### Mục 4.6: Staging một tệp đơn lẻ

Để staging một tệp để commit, chạy:

bash

Sao chép mã

git add <filename>

#### Mục 4.7: Staging các tệp đã xóa

bash

Sao chép mã

git rm <filename>

Để xóa tệp khỏi git mà không xóa nó khỏi đĩa, sử dụng cờ --cached

bash

Sao chép mã

git rm --cached <filename>

Top of Form

Bottom of Form