Mục lục

[1. init: Khởi tạo một kho chứa từ thư mục cũ: 1](#_Toc501097645)

[2. clone: Sao chép một kho chứa đã tồn tại: 1](#_Toc501097646)

[3. status: Kiểm tra trạng thái của tập tin: 2](#_Toc501097647)

[4. add: Theo dõi các tập tin mới: 2](#_Toc501097648)

[5. Quản lý các tập tin đã thay đổi: 2](#_Toc501097649)

[6. Bỏ qua các tập tin: 2](#_Toc501097650)

[7. diff: Xem các thay đổi staged và unstaged: 3](#_Toc501097651)

[8. commit thay đổi: 4](#_Toc501097652)

[9. Bỏ qua khu vực tổ chức: 4](#_Toc501097653)

[10. rm: Xoá tập tin: 4](#_Toc501097654)

[11. mv: Di chuyển tập tin: 5](#_Toc501097655)

[12. log: Xem lịch sử commit: 5](#_Toc501097656)

[13. Giới Hạn Thông Tin Đầu Ra: 6](#_Toc501097657)

[14. gitk: Hiển thị lịch sử trên giao diện: 7](#_Toc501097658)

[15. Thay đổi commit cuối cùng 7](#_Toc501097659)

[16. git reset HEAD: Loại bỏ tập tin đã tổ chức: 7](#_Toc501097660)

[17. Phục hồi tập tin đã thay đổi: 7](#_Toc501097661)

Các lệnh git cơ bản:

# 1. init: Khởi tạo một kho chứa từ thư mục cũ:

- Nếu như bạn muốn theo dõi một dự án cũ trong Git, bạn cần ở trong thư mục của dự án đó và gõ lệnh sau:

+ "$ git init"

+ Lệnh này sẽ tạo một thư mục mới có tên .git, thư mục này chứa tất cả các tập tin cần thiết cho kho chứa - đó chính là bộ khung/xương của kho chứa Git.

- $ git init --bare <directory>: tạo ra một repository rỗng, nhưng không có working directory nên không thể chỉnh sửa các tệp tin hay commit những thay đổi trong repository này. Lệnh này thường được dùng để tạo remote central repository.

- $ git init <directory> --template=<template\_directory>: tạo ra một repository mới và copy các file từ <template\_directory> vào repository này.

- Các option khác:

+ -q, --quiet: chỉ in ra các error, warning message và bỏ qua các output khác.

+ --separate-git-dir=<git dir>: tạo ra một file text chứa đường dẫn (path) tới các thư mục của git.

+ --shared[=(false|true|umask|group|all|world|everybody|0xxx)]: chỉ định việc chia sẻ repository đến nhiều người dùng

\* umask (hoặc false): dùng phân quyền được report bởi umask. (Mặc định).

\* group (hoặc true): khiến repository có thể được chỉnh sửa bởi group.

\* all (hoặc world, everybody): giống với group nhưng ở đây là khiến cho tất cả mọi người dùng đều đọc được repository này.

\* 0xxx: 0640 - tạo ra repository có thể đọc bởi group nhưng không thể sửa hay công khai cho người khác. 0660 - tạo ra repository có thể đọc và sửa bởi người dùng hiện tại và group nhưng không thể công khai với người khác.

- Sau câu lệnh init là câu lệnh add, commit rồi đến câu lệnh remote: $ git remote add origin <URL> với URL là đường dẫn của một repository online (được tạo trên trang github). Sau đó có thể dùng $ git push -u origin master để push lên repository này (Tham số -u để cấu hình mặc định remote của mình sẽ là origin và nhánh mặc định sẽ là nhánh hiện tại. Từ nay về sau chỉ việc gõ git push là đủ.).

- Có thể xem URL của remote repo đó bằng câu lệnh git remote -v.

- git remote set-url origin <url>: được sử dụng khi bạn cần đổi remote repo. Giả dụ bạn copy một cái repo của người khác và bạn muốn chuyển owner thành tài khoản GitHub của bạn. Thì bạn sử dụng câu lệnh đó.

# 2. clone: Sao chép một kho chứa đã tồn tại:

- Sử dụng lệnh git clone [url] để sao chép một kho chứa. Ví dụ, nếu bạn muốn tạo một bản sao của thư viện Ruby Git có tên Grit, bạn có thể thực hiện như sau:

+ "$ git clone git://github.com/schacon/grit.git"

+ Một thư mục mới có tên grit sẽ được tạo, kèm theo thư mục .git và bản sao mới nhất của tất cả dữ liệu của kho chứa đó bên trong. Nếu bạn xem bên trong thư mục grit, bạn sẽ thấy các tập tin của dự án bên trong, và đã sẵn sàng cho bạn làm việc hoặc sử dụng.

- Nếu bạn muốn sao chép kho chứa này vào một thư mục có tên khác không phải là grit, bạn có thể chỉ định tên thư mục đó như là một tuỳ chọn tiếp theo khi chạy dòng lệnh:

+ "$ git clone git://github.com/schacon/grit.git mygrit"

+ Lệnh này thực thi tương tự như lệnh trước, nhưng thư mục của kho chứa lúc này sẽ có tên là mygrit.

- Các option:

+ --local, -l: bỏ qua cơ chế "Git aware" thông thường mà chỉ thực hiện clone repository bằng cách copy HEAD và các đối tượng dưới nó, thư mục refs. Các tệp trong thư mục .git/objects sẽ được hardlink để tiết kiệm không gian. Nếu repository là một local path thì --local là mặc định. Nếu repository là URL thì --local bị bỏ qua. Sử dụng --no--local để ghi đè lên trong trường hợp repository là local path, lúc này ta sẽ sử dụng được phương thức vận chuyển của Git theo các thông thường.

+ --no-hardlinks: bắt buộc clone cả những file trong thư mục .git/objects thay vì dùng hardlinks. Điều này có ý nghĩa khi tạo dữ liệu back-up.

# 3. status: Kiểm tra trạng thái của tập tin:

- Hiển thị những tập tin chưa được add, chưa được commit, nhánh (branch) hiện tại...

+ On branch master...: branch hiện tại.

+nothing to commit...: thư mục làm việc "sạch", không có tập tin nào bị thay đổi.

+ untracked files... có tập tin mới được tạo ra và chưa được theo dõi (untrack).

# 4. add: Theo dõi các tập tin mới:

- Để có thể theo dõi các tập tin mới tạo, bạn sử dụng lệnh git add. Và để bắt đầu theo dõi tập tin README bạn có thể chạy lệnh sau:

+ "$ git add README"

+ sau đó chạy lệnh status để kiểm tra, ta thấy thông báo:

- "...Changes to be committed:... new file: README..."

- ta thấy README đã được staged và đang nằm trong danh sách các thay đổi chuẩn bị commit.

# 5. Quản lý các tập tin đã thay đổi:

- mỗi lần sửa đổi một tập tin đã được theo dõi từ trước phải chạy lại lệnh git add.

# 6. Bỏ qua các tập tin:

- Một số tập tin mà bạn không muốn git tự động thêm vào hay không theo dõi (các tập tin nhật ký, các tập tin sinh ra khi biên dịch chương trình...). Trong trường hợp này bạn có thể tạo một tập tin tên là ".gitignore" liệt kê các mẫu (patterns) để hiển thị những tập tin đó, ví dụ như:

+ "$ cat .gitignore

\*.[oa]

\*~"

- dòng đầu tiên yêu cầu git bỏ qua các tập tin có đuôi .o hoặc .a (các tập tin object và archiev được tạo ra khi dịch mã nguồn)

- dòng thứ hai yêu cầu git bỏ qua các tập tin có đuôi ~ (các tập tin được dùng để lưu các giá trị tạm thời bởi các chương trình soạn thảo như Emacs)

- bạn có thể thêm các thư mục: log, tmp, pid...

+ Quy tắc tạo các mẫu:

- dòng trống hoặc bắt đầu bằng # sẽ được bỏ qua.

- các mẫu chuẩn toàn cầu hoạt động tốt (giống biểu thức chính quy trong shell).

- các mẫu có thể kết thúc bằng / để chỉ định một thư mục.

- có thể có các mẫu phủ định bằng các thêm ! vào đầu của mẫu.

- mẫu \*\*/ có trong git từ phiên bản 1.8.2.

# 7. diff: Xem các thay đổi staged và unstaged:

- nếu muốn biết chính xác cái đã thay đổi là gì chứ không chỉ tập tin nào bị thay đổi, bạn có thể dùng lệnh git diff.

+ Để xem chính xác bạn đã thay đổi nhưng chưa stage những gì, hãy dùng git diff không sử dụng tham số nào khác.

- Câu lệnh này so sánh cái ở trong thư mục làm việc của bạn với cái ở trong khu vực tổ chức (staging).

+ Nếu bạn muốn xem những gì bạn đã staged mà chuẩn bị được commit, bạn có thể sử dụng git diff --cached. (từ git 1.6.1 có thể sử dụng git diff --staged).

- Lệnh này so sánh những thay đổi đã được tổ chức với lần commit trước đó.

+ Nếu chỉ chạy git diff không thôi thì nó sẽ không hiển thị cho bạn tất cả thay đổi từ lần comiit trước - mà chỉ có các thay đổi chưa được tổ chức. Điều này có thể gây khó hiểu một chút, bởi vì nếu như bạn đã tổ chức tất cả các thay đổi, git diff sẽ không hiện gì cả.

# 8. commit thay đổi:

- Bây giờ, sau khi đã tổ chức các tập tin theo ý muốn, bạn có thể commit chúng. Những gì chưa được tổ chức ( những tập tin được tạo ra hoặc sửa đổi sau khi chạy lệnh git add) sẽ không được commit. Cách đơn giản nhất để commit là gõ vào git commit.

+ hiện ra thông báo, có thể xoá những dòng chú thích (#) và nhập vào nội dung cho commit đó, hoặc giữ nguyên chú thích để dễ ghi nhớ (để rõ ràng hơn có thể truyền tham số -v cho git commit để đưa tất cả các thay đổi khi thực hiện lệnh diff vào).

+ Khi thoát khỏi trình soạn thảo, git sẽ tạo commit của bạn với các thông báo đó.

+ có thể gõ trực tiếp thông điệp với lệnh commit:

- "$ git commit -m "gõ vào nội dung thông điệp" ".

# 9. Bỏ qua khu vực tổ chức:

- Đôi khi khu vực tổ chức khiến quy trình làm việc của bạn trở nên phức tạp. Nếu bạn muốn bỏ qua bước này, chỉ cần thêm vào lựa chọn -a khi thực hiện git commit, Git sẽ tự động thêm tất cả các tập tin đã được theo dõi trước khi thực hiện lệnh commit, cho phép bạn bỏ qua bước git add, ví dụ:

+ "$ git commit -a -m 'gõ vào nội dung commit' ".

# 10. rm: Xoá tập tin:

- Để xoá một tập tin khỏi Git, bạn phải xoá nó khỏi danh sách được theo dõi (chính xác hơn, xoá nó khỏi khu vực tổ chức) và sau đó commit. Lệnh git rm thực hiện điều đó và cũng xoá tập tin khỏi thư mục làm việc vì thế bạn sẽ không thấy nó như là tập tin không được theo dõi trong những lần tiếp theo.

- Nếu bạn muốn giữ tập tin trong thư mục làm việc nhưng không thêm chúng vào khu vực tổ chức (lưu tập tin trên ổ cứng nhưng không muốn Git theo dõi chúng). Điều này đặc biệt hữu ích nếu như bạn quên thêm nó vào tập .gitignore và vô tình tổ chức (stage) chúng, ví dụ như một tập tin nhật ký lớn hoặc rất nhiều tập tin .a. Để làm được điều này, hãy sử dụng lựa chọn --cached, ví dụ:

+ "$ git rm --cached readme.txt"

# 11. mv: Di chuyển tập tin:

- Nếu muốn di chuyển tập tin, có thể dùng lệnh:

+ "$ git mv file\_from file\_to"

# 12. log: Xem lịch sử commit:

- Mặc định, không sử dụng tham số nào, git log liệt kê các commit được thực hiện trong kho chứa đó theo thứ tự thời gian.

- Các tuỳ chọn: Một trong các tuỳ chọn hữu ích nhất là -p, nó hiện thị diff của từng commit. Bạn cũng có thể dùng -2 để giới hạn chỉ hiển thị hai commit gần nhất.

- "$ git log -U1 --word-diff": để xem diff một cách tổng quát thay vì xem từng dòng theo cách thông thường.

+ Thay đổi được hiển thị ngay trên một dòng. Bạn có thể thấy phần thêm mới được bao quanh trong {+ +} còn phần xoá đi thì trong [- -].

- "$ git log --stat": xem một số thống kê tóm tắt cho mỗi commit.

- "--pretty": thay đổi phần hiển thị ra theo các cách khác nhau. Có một số lựa chọn được cung cấp sẵn cho bạn sử dụng. Lựa chọn oneline in mỗi commit trên một dòng, có ích khi bạn xem nhiều commit cùng lúc. Ngoài ra các lựa chọn short, full, và fuller hiện thị gần như tương tự nhau với ít hoặc nhiều thông tin hơn theo cùng thứ tự.

- "format": cho phép bạn chỉ định định dạng riêng của phần hiện thị. Nó đặc biệt hữu ích khi bạn đang xuất ra cho các máy phân tích thông tin (machine parsing) - vì bạn là người chỉ rõ định dạng, nên bạn sẽ biết được nó không bị thay đổi cùng với các cập nhật sau này của Git, ví dụ:

+ "$ git log --pretty=format:"%h - %an, %ar : %s" ".

+ Bảng 2-1 liệt kê một vài lựa chọn mà format sử dụng.

|  |  |
| --- | --- |
| Lựa chọn | Mô tả thông tin đầu ra |
| %H | Mã băm của commit |
| %h | Mã băm của commit ngắn gọn hơn |
| %T | Băm hiển thị dạng cây |
| %t | Băm hiển thị dạng cây ngắn gọn hơn |
| %P | Các mã băm gốc |
| %p | Mã băm gốc ngắn gọn |
| %an | Tên tác giả |
| %ae | E-mail tác giả |
| %ad | Ngày "tác giả" (định dạng tương tự như lựa chọn --date= ) |
| %ar | Ngày tác giả, tương đối |
| %cn | Tên người commit |
| %ce | Email người commit |
| %cd | Ngày commit |
| %cr | Ngày commit, tương đối |
| %s | Chủ để |

+ Tác giả là người đầu tiên viết bản vá (patch), trong khi đó người commit là người cuối cùng áp dụng miếng vá đó.

+ Lựa chọn oneline và format đặc biệt hữu ích khi sử dụng với một tham số khác của log là --graph. Khi sử dụng, tham số này sẽ thêm một biểu đồ sử dụng dựa trên các ký tự ASCII hiển thị nhánh và lịch sử tích hợp các tập tin của bạn.

# 13. Giới Hạn Thông Tin Đầu Ra:

- git log nhận vào một số lựa chọn cho phép giới hạn hiển thị một phần các commit.

- Dùng -n với n là số nguyên dương bất kỳ để hiển thị n commit mới nhất.

- tham số giới hạn theo thời gian như --since, --until, ví dụ lệnh để hiển thị các commit được thực hiện trong vòng hai tuần gần đây là:

+ "$ git log --since=2.weeks".

+ bạn cũng có thể chỉ định một ngày cụ thể ("2008-01-15") hoặc tương đối như "2 years 1 day 3 minutes ago".

- Có thể lọc các commit theo một số tiêu chí nhất định như:

|  |  |
| --- | --- |
| Lựa chọn | Mô tả |
| -(n) | Chỉ hiển thị n commit mới nhất |
| --since, --after | Giới hạn các commit được thực hiện sau ngày nhất định. |
| --until, --before | Giới hạn các commit được thực hiện trước ngày nhất định. |
| --author | Chỉ hiện thị các commit mà tên tác giả thoả mãn điều kiện nhất định. |
| --committer | Chỉ hiện thị các commit mà tên người commit thoả mãn điều kiện nhất định. |

# 14. gitk: Hiển thị lịch sử trên giao diện:

- Nếu muốn sử dụng một công cụ đồ hoạ để trực quan hoá lịch sử commit, bạn có thể dùng chương trình Tcl/Tk có tên gitk đi kèm với git. Gitk cơ bản là một công cụ git log trực quan, chấp nhận hầu hết các tuỳ chọn lọc của git log.

# 15. Thay đổi commit cuối cùng

- Trường hợp bạn quên thêm vào một số tập tin hoặc thông điệp commit không như ý muốn. Nếu bạn muốn thực hiện lại lệnh commit đó, bạn có thể chạy lệnh commit:

+ "$ git commit --amend".

# 16. git reset HEAD: Loại bỏ tập tin đã tổ chức:

- Khi muốn unstage một tập tin đang ở trạng thái staged, ta gõ lệnh:

+ "$ git reset HEAD tên\_tập\_tin".

# 17. Phục hồi tập tin đã thay đổi:

- "$ git checkout -- tên\_tập\_tin".

+ Đây là một câu lệnh nguy hiểm, bất kỳ thay đổi nào được thực hiện trên tập tin đó không còn nữa.

# \*\* $ git remote show origin

Để kiểm tra tình trạng các nhánh ở local với master.s