Khái niệm cơ bản về Git

Nếu bạn chỉ có thể đọc một chương để bắt đầu với Git thì đây chính là chương đó. Chương này trình bày mọi lệnh cơ bản mà bạn cần để thực hiện phần lớn những việc mà cuối cùng bạn sẽ dành thời gian làm với Git. Đến cuối chương này, bạn sẽ có thể định cấu hình và khởi tạo kho lưu trữ, bắt đầu và dừng theo dõi các tệp cũng như thực hiện và thực hiện các thay đổi. Chúng tôi cũng sẽ chỉ cho bạn cách thiết lập Git để bỏ qua một số tệp và mẫu tệp nhất định, cách hoàn tác các lỗi nhanh chóng và dễ dàng, cách duyệt lịch sử dự án của bạn và xem các thay đổi giữa các lần xác nhận cũng như cách đẩy và kéo từ kho lưu trữ từ xa .

Lấy kho lưu trữ Git

Bạn thường có được kho lưu trữ Git theo một trong hai cách:

1. Bạn có thể lấy một thư mục cục bộ hiện không được kiểm soát phiên bản và biến nó thành kho lưu trữ Git hoặc
2. Bạn có thể sao chép kho lưu trữ Git hiện có từ nơi khác.

Trong cả hai trường hợp, bạn đều có kho lưu trữ Git trên máy cục bộ của mình, sẵn sàng hoạt động.

Khởi tạo một kho lưu trữ trong một thư mục hiện có

Nếu bạn có một thư mục dự án hiện không được kiểm soát phiên bản và bạn muốn bắt đầu kiểm soát nó bằng Git, trước tiên bạn cần vào thư mục của dự án đó. Nếu bạn chưa bao giờ thực hiện việc này thì nó sẽ hơi khác một chút tùy thuộc vào hệ thống bạn đang chạy:

Điều này tạo ra một thư mục con mới có tên .git chứa tất cả các tệp kho lưu trữ cần thiết của bạn - khung kho lưu trữ Git. Tại thời điểm này, chưa có gì trong dự án của bạn được theo dõi. Xem Nội bộ Git để biết thêm thông tin về chính xác những tệp nào có trong thư mục .git bạn vừa tạo.

Nếu bạn muốn bắt đầu kiểm soát phiên bản các tệp hiện có (ngược lại với một thư mục trống), có lẽ bạn nên bắt đầu theo dõi các tệp đó và thực hiện cam kết ban đầu. Bạn có thể thực hiện điều đó bằng lệnh git add chỉ định các tệp bạn muốn theo dõi, sau đó là lenh git commit:

Nhân bản một kho lưu trữ hiện có

Nếu bạn muốn có bản sao của kho lưu trữ Git hiện có - ví dụ: một dự án bạn muốn đóng góp - lệnh bạn cần là git clone. Nếu bạn quen thuộc với các VCS khác như Subversion, bạn sẽ nhận thấy rằng lệnh này là "sao chép" chứ không phải "kiểm tra". Đây là một điểm khác biệt quan trọng — thay vì chỉ nhận một bản sao đang hoạt động, Git nhận được một bản sao đầy đủ của gần như tất cả dữ liệu mà máy chủ có. Theo mặc định, mọi phiên bản của mọi tệp trong lịch sử dự án đều được kéo xuống khi bạn chạy git clone. Trên thực tế, nếu đĩa máy chủ của bạn bị hỏng, bạn thường có thể sử dụng gần như bất kỳ bản sao nào trên bất kỳ máy khách nào để đặt máy chủ trở lại trạng thái như khi nó được sao chép (bạn có thể mất một số hook phía máy chủ, v.v., nhưng tất cả dữ liệu được phiên bản sẽ ở đó - xem Lấy Git trên Máy chủ để biết thêm chi tiết).

Bạn sao chép một kho lưu trữ bằng git clone <url>. Ví dụ: nếu bạn muốn sao chép thư viện có thể liên kết Git có tên libgit2, bạn có thể làm như sau:

git clone https://github.com/libgit2/libgit2

Thao tác đó tạo ra một thư mục có tên libgit2, khởi tạo thư mục .git bên trong nó, lấy tất cả dữ liệu cho kho lưu trữ đó xuống và kiểm tra bản sao hoạt động của phiên bản mới nhất. Nếu bạn đi vào thư mục libgit2 mới vừa được tạo, bạn sẽ thấy các tệp dự án trong đó, sẵn sàng để làm việc hoặc sử dụng.

Nếu bạn muốn sao chép kho lưu trữ vào một thư mục có tên khác libgit2, bạn có thể chỉ định tên thư mục mới làm đối số bổ sung:

git clone https://github.com/libgit2/libgit2 mylibgit

Lệnh đó thực hiện tương tự như lệnh trước, nhưng thư mục đích được gọi là mylibgit.

Git có một số giao thức truyền tải khác nhau mà bạn có thể sử dụng. Ví dụ trước sử dụng giao thức https://, nhưng bạn cũng có thể thấy git:// hoặc

user@server:path/to/repo.git, sử dụng giao thức truyền SSH. Tải Git trên Máy chủ sẽ giới thiệu tất cả các tùy chọn có sẵn mà máy chủ có thể thiết lập để truy cập kho lưu trữ Git của bạn cũng như những ưu và nhược điểm của từng tùy chọn.

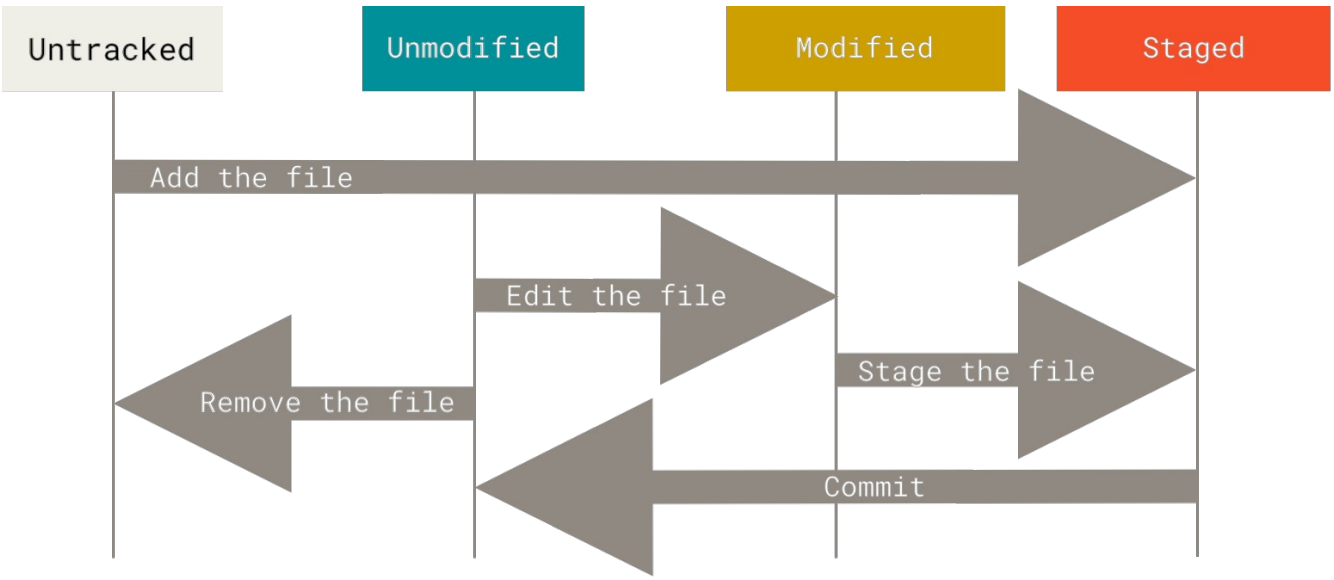
Ghi lại các thay đổi đối với kho lưu trữ

Tại thời điểm này, bạn nên có một kho lưu trữ Git thực sự trên máy cục bộ của mình và bản kiểm tra hoặc bản sao hoạt động của tất cả các tệp trong đó ngay trước mặt bạn. Thông thường, bạn sẽ muốn bắt đầu thực hiện các thay đổi và đưa ảnh chụp nhanh của những thay đổi đó vào kho lưu trữ của mình mỗi khi dự án đạt đến trạng thái bạn muốn ghi lại.

Hãy nhớ rằng mỗi tệp trong thư mục làm việc của bạn có thể ở một trong hai trạng thái: được theo dõi hoặc không được theo dõi. Tệp được theo dõi là các tệp nằm trong ảnh chụp nhanh cuối cùng, cũng như mọi tệp mới được sắp xếp; chúng có thể không được sửa đổi, sửa đổi hoặc dàn dựng. Nói tóm lại, các tệp được theo dõi là các tệp mà Git biết.

Các tệp không bị theo dõi là mọi thứ khác — bất kỳ tệp nào trong thư mục làm việc của bạn không có trong ảnh chụp nhanh cuối cùng và không nằm trong khu vực tổ chức của bạn. Khi bạn sao chép kho lưu trữ lần đầu tiên, tất cả các tệp của bạn sẽ được theo dõi và không được sửa đổi vì Git chỉ kiểm tra chúng và bạn chưa chỉnh sửa bất kỳ thứ gì.

Khi bạn chỉnh sửa các tệp, Git sẽ xem chúng như đã được sửa đổi vì bạn đã thay đổi chúng kể từ lần chuyển giao cuối cùng của bạn. Khi làm việc, bạn sắp xếp có chọn lọc các tệp đã sửa đổi này rồi thực hiện tất cả các thay đổi theo giai đoạn đó và chu trình lặp lại.



Kiểm tra trạng thái tệp của bạn

Công cụ chính bạn sử dụng để xác định tập tin nào đang ở trạng thái nào là lệnh git status. Nếu bạn chạy lệnh này trực tiếp sau khi sao chép, bạn sẽ thấy nội dung như sau:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

Điều này có nghĩa là bạn có một thư mục làm việc sạch sẽ; nói cách khác, không có tệp nào được theo dõi của bạn được sửa đổi. Git cũng không thấy bất kỳ tệp nào chưa được theo dõi, nếu không chúng sẽ được liệt kê ở đây. Cuối cùng, lệnh cho bạn biết bạn đang ở nhánh nào và thông báo cho bạn rằng nhánh đó chưa được chuyển hướng khỏi cùng một nhánh trên máy chủ. Hiện tại, nhánh đó luôn là nhánh chính, đây là nhánh mặc định; bạn sẽ không lo lắng về nó ở đây. Phân nhánh Git sẽ đi qua các nhánh và tài liệu tham khảo một cách chi tiết.

GitHub đã thay đổi tên nhánh mặc định từ master thành main vào giữa năm 2020 và các máy chủ Git khác cũng làm theo. Vì vậy, bạn có thể thấy rằng tên nhánh mặc định trong một số kho mới được tạo là main chứ không phải master. Ngoài ra, tên nhánh mặc định có thể được thay đổi (như bạn đã thấy trong Tên nhánh mặc định của bạn), do đó bạn có thể thấy một tên khác cho nhánh mặc định.

Tuy nhiên, bản thân Git vẫn sử dụng master làm mặc định nên chúng ta sẽ sử dụng nó xuyên suốt cuốn sách.

Giả sử bạn thêm một tệp mới vào dự án của mình, một tệp README đơn giản. Nếu tệp không tồn tại trước đó và bạn chạy git status, bạn sẽ thấy tệp không bị theo dõi như sau:

$ echo 'My Project' > README
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
(use "git add <file>..." to include in what will be committed)
README
nothing added to commit but untracked files present (use "git add" to track)

Bạn có thể thấy rằng tệp README mới của mình không bị theo dõi vì nó nằm trong tiêu đề “Tệp không bị theo dõi” trong đầu ra trạng thái của bạn. Về cơ bản, không bị theo dõi có nghĩa là Git nhìn thấy một tệp mà bạn không có trong ảnh chụp nhanh trước đó (cam kết) và tệp này chưa được sắp xếp; Git sẽ không bắt đầu đưa nó vào ảnh chụp nhanh cam kết của bạn cho đến khi bạn yêu cầu nó làm như vậy một cách rõ ràng. Nó thực hiện điều này để bạn không vô tình bắt đầu bao gồm các tệp nhị phân được tạo hoặc các tệp khác mà bạn không có ý định đưa vào. Bạn muốn bắt đầu bao gồm README, vì vậy hãy bắt đầu theo dõi tệp.

Theo dõi tập tin mới

Để bắt đầu theo dõi một file mới, bạn sử dụng lệnh git add. Để bắt đầu theo dõi tệp README, bạn có thể chạy lệnh này:

git add README

Nếu bạn chạy lại lệnh git status, bạn có thể thấy rằng tệp README của bạn hiện đã được theo dõi và sắp xếp để cam kết:

$ git status
On branch main
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file: README
Bạn có thể biết rằng nó được dàn dựng vì nó nằm trong tiêu đề “Những thay đổi cần cam kết”. Nếu bạn cam kết tại thời điểm này, phiên bản của tệp tại thời điểm bạn chạy git add sẽ có trong ảnh chụp

nhanh lịch sử tiếp theo. Bạn có thể nhớ lại rằng khi chạy git init trước đó, sau đó bạn chạy git add <files> — để bắt đầu theo dõi các tập tin trong thư mục của bạn. Lệnh git add lấy tên đường dẫn cho tệp hoặc thư mục; nếu đó là một thư mục, lệnh sẽ thêm tất cả các tệp trong thư mục đó theo cách đệ quy.

Sắp xếp các tập tin đã sửa đổi

Hãy thay đổi một tập tin đã được theo dõi. Nếu bạn thay đổi tệp được theo dõi trước đó có tên CONTRIBUTING.md rồi chạy lại lệnh git status, bạn sẽ nhận được kết quả như thế này:

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:    CONTRIBUTING.md
```

Tệp CONTRIBUTING.md xuất hiện trong phần có tên “Changes not staged for commit” - có nghĩa là tệp được theo dõi đã được sửa đổi trong thư mục làm việc nhưng chưa được sắp xếp. Để thực hiện nó, bạn chạy lệnh git add. git add là một lệnh đa năng — bạn sử dụng nó để bắt đầu theo dõi các tệp mới, sắp xếp các tệp và thực hiện những việc khác như đánh dấu các tệp xung đột khi hợp nhất là đã được giải quyết. Có thể hữu ích nếu bạn nghĩ về nó giống như “thêm chính xác nội dung này vào cam kết tiếp theo” thay vì “thêm tệp này vào dự án”. Hãy chạy git add ngay bây giờ để tạo tệp CONTRIBUTING.md, sau đó chạy lại git status:

```
$ git add CONTRIBUTING.md
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README
        modified:   CONTRIBUTING.md
```

Cả hai tệp đều được dàn dựng và sẽ đi vào lần xác nhận tiếp theo của bạn. Tại thời điểm này, giả sử bạn nhớ một thay đổi nhỏ mà bạn muốn thực hiện trong

CONSTRIBUTING.md trước khi thực hiện nó. Bạn mở lại và thực hiện thay đổi đó và bạn đã sẵn sàng cam kết. Tuy nhiên, hãy chạy git status một lần nữa:

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README
        modified:   CONTRIBUTING.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   CONTRIBUTING.md
```

Cái quái gì vậy? Bây giờ CONTRIBUTING.md được liệt kê ở cả “Changes to be commited” và “Changes not staged for commit”. Làm sao điều đó có thể được? Hóa ra Git xử lý một tệp chính xác như khi bạn chạy lệnh git add. Nếu bạn cam kết bây giờ, phiên bản CONTRIBUTING.md giống như lần cuối bạn chạy lệnh git add là cách nó sẽ đi vào cam kết chứ không phải phiên bản của tệp như trong thư mục làm việc của bạn khi bạn chạy git commit. Nếu bạn sửa đổi một tệp sau khi chạy git add, bạn phải chạy lại git add để cập nhật phiên bản mới nhất của tệp:

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README
        modified:   CONTRIBUTING.md
```

Trạng thái ngắn

Mặc dù đầu ra trạng thái git khá đầy đủ nhưng nó cũng khá dài dòng. Git cũng có một cờ trạng thái ngắn để bạn có thể xem các thay đổi của mình một cách gọn gàng hơn. Nếu bạn chạy git status -s hoặc git status --short bạn sẽ nhận được kết quả đầu ra đơn giản hơn nhiều từ lệnh:

Bỏ qua tập tin

Thông thường, bạn sẽ có một loại tệp mà bạn không muốn Git tự động thêm hoặc thậm chí hiển thị cho bạn là không bị theo dõi. Đây thường là các tệp được tạo tự động như tệp nhật ký hoặc tệp do hệ thống xây dựng của bạn tạo ra. Trong những trường hợp như vậy, bạn có thể tạo mẫu danh sách tệp để khớp với chúng có tên .gitignore. Đây là một tệp .gitignore ví dụ:

Dòng đầu tiên yêu cầu Git bỏ qua mọi tệp kết thúc bằng “.o” hoặc “.a” - tệp đối tượng và tệp lưu trữ có thể là sản phẩm của quá trình xây dựng mã của bạn. Dòng thứ hai yêu cầu Git bỏ qua tất cả các tệp có tên kết thúc bằng dấu ngã (~), được nhiều trình soạn thảo văn bản như Emacs sử dụng để đánh dấu các tệp tạm thời. Bạn cũng có thể bao gồm thư mục log, tmp hoặc pid; tài liệu được tạo tự động; và như thế. Thiết lập tệp .gitignore cho kho lưu trữ mới của bạn trước khi bắt đầu thường là một ý tưởng hay để bạn không vô tình đưa các tệp mà bạn thực sự không muốn vào kho Git của mình.

Các quy tắc cho các mẫu bạn có thể đặt trong tệp .gitignore như sau:

• Các dòng trống hoặc các dòng bắt đầu bằng # sẽ bị bỏ qua.

• Các mẫu hình cầu tiêu chuẩn hoạt động và sẽ được áp dụng đệ quy trên toàn bộ cây làm việc.

• Bạn có thể bắt đầu mẫu bằng dấu gạch chéo lên (/) để tránh đệ quy.

• Bạn có thể kết thúc mẫu bằng dấu gạch chéo lên (/) để chỉ định thư mục.

• Bạn có thể phủ định một mẫu bằng cách bắt đầu nó bằng dấu chấm than (!).

Các mẫu toàn cầu giống như các biểu thức chính quy được đơn giản hóa mà shell sử dụng. Dấu hoa thị (*) khớp với 0 hoặc nhiều ký tự; [abc] khớp với bất kỳ ký tự nào bên trong dấu ngoặc (trong trường hợp này là a, b hoặc c); dấu chấm hỏi (?) khớp với một ký tự; và dấu ngoặc bao quanh các ký tự được phân tách bằng dấu gạch nối ([0-9]) khớp với bất kỳ ký tự nào giữa chúng (trong trường hợp này là 0 đến 9).

Bạn cũng có thể sử dụng hai dấu hoa thị để khớp với các thư mục lồng nhau; a/**/z sẽ khớp với a/z, a/b/z, a/b/c/z, v.v.

Đây là một tệp .gitignore ví dụ khác:

```
# ignore all .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in any directory named build
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

GitHub duy trì một danh sách khá đầy đủ các ví dụ về tệp .gitignore tốt cho hàng tá dự án và ngôn ngữ tại https://github.com/github/gitignore nếu bạn muốn có điểm khởi đầu cho dự án của mình.

Trong trường hợp đơn giản, một kho lưu trữ có thể có một tệp .gitignore duy nhất trong thư mục gốc của nó, tệp này áp dụng đệ quy cho toàn bộ kho lưu trữ. Tuy nhiên, cũng có thể có thêm các tệp .gitignore trong thư mục con. Các quy tắc trong các tệp .gitignore lồng nhau này chỉ áp dụng cho các tệp trong thư mục chứa chúng. Kho lưu trữ nguồn nhân Linux có 206 tệp .gitignore.

Việc đi sâu vào chi tiết của nhiều tệp .gitignore nằm ngoài phạm vi của cuốn sách này; sử dụng lệnh man gitignore để biết chi tiết.

Xem Staged and Unstaged

Nếu lệnh git status quá mơ hồ đối với bạn — bạn muốn biết chính xác những gì bạn đã thay đổi — bạn có thể sử dụng lệnh git diff. Chúng tôi sẽ đề cập chi tiết hơn về git diff sau, nhưng có thể bạn sẽ sử dụng nó thường xuyên nhất để trả lời hai câu hỏi sau: Bạn đã thay đổi điều gì nhưng chưa được staged? Và bạn đã staged những gì mà bạn sắp commit? Git status trả lời những câu hỏi đó một cách tổng quát bằng cách liệt kê tên tệp, git diff hiển thị cho bạn các dòng chính xác được thêm và xóa - bản vá, như trước đây.

Giả sử bạn chỉnh sửa và sắp xếp lại tệp README, sau đó chỉnh sửa tệp CONTRIBUTING.md mà không sắp xếp nó. Nếu bạn chạy lệnh git status, một lần nữa bạn sẽ thấy thứ gì đó như thế này:

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:    CONTRIBUTING.md
```

Để xem những gì bạn đã thay đổi nhưng chưa được dàn dựng, hãy nhập git diff mà không có đối số nào khác:

```
$ git diff --staged
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
new file mode 100644
index 0000000..ad80c42
--- /dev/null
+++ b/CONTRIBUTING.md
@@ -0,0 +1 @@
+create file
```

Điều quan trọng cần lưu ý là bản thân git diff không hiển thị tất cả các thay đổi được thực hiện kể từ lần cam kết cuối cùng của bạn - chỉ những thay đổi vẫn chưa được thực hiện. Nếu bạn đã dàn dựng tất cả các thay đổi của mình, git diff sẽ không cung cấp cho bạn kết quả nào.

Đối với một ví dụ khác, nếu bạn tạo giai đoạn cho tệp CONTRIBUTING.md rồi chỉnh sửa nó, bạn có thể sử dụng git diff để xem những thay đổi trong tệp được phân giai đoạn và những thay đổi không được phân giai đoạn. Nếu môi trường của chúng ta trông như thế này:

Chúng ta sẽ tiếp tục sử dụng lệnh git diff theo nhiều cách khác nhau trong suốt phần còn lại của cuốn sách. Có một cách khác để xem xét những khác biệt này nếu bạn thích chương trình xem khác biệt bằng đồ họa hoặc bên ngoài. Nếu bạn chạy git Difftool thay vì git diff, bạn có thể xem bất kỳ khác biệt nào trong số này trong phần mềm như nổi, vimdiff và nhiều phần mềm khác (bao gồm cả các sản phẩm thương mại). Chạy git Difftool --tool-help để xem những gì có sẵn trên hệ thống của bạn.

Cam kết thay đổi của bạn

Bây giờ khu vực tổ chức của bạn đã được thiết lập theo cách bạn muốn, bạn có thể thực hiện các thay đổi của mình. Hãy nhớ rằng mọi thứ vẫn chưa được phân loại - bất kỳ tệp nào bạn đã tạo hoặc sửa đổi mà bạn chưa chạy git add kể từ khi bạn chỉnh sửa chúng - sẽ không được đưa vào cam kết này. Chúng sẽ ở dạng tệp đã sửa đổi trên đĩa của bạn. Trong trường hợp này, giả sử rằng lần cuối cùng bạn chạy git status, bạn thấy rằng mọi thứ đã được dàn dựng nên bạn đã sẵn sàng thực hiện các thay đổi của mình. Cách đơn giản nhất để cam kết là gõ git commit:

Làm như vậy sẽ khởi chạy trình soạn thảo bạn chọn.

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#
new file:
README
#
modified:
CONTRIBUTING.md
#
~
~
~
".git/COMMIT_EDITMSG" 9L, 283C
```

Bạn có thể thấy rằng thông báo cam kết mặc định chứa kết quả đầu ra mới nhất của lệnh git status đã được nhận xét và một dòng trống ở trên cùng. Bạn có thể xóa những nhận xét này và nhập thông báo cam kết của mình hoặc bạn có thể để chúng ở đó để giúp bạn ghi nhớ những gì bạn đang cam kết.

Để có lời nhắc rõ ràng hơn về những gì bạn đã sửa đổi, bạn có thể sử dụng lệnh git commit -v. Làm như vậy cũng tạo ra sự khác biệt về thay đổi của bạn trong trình chỉnh sửa để bạn có thể biết chính xác những thay đổi mà bạn đang thực hiện.

Banj hayx thu su dung lenh git commit va git commit -v, sau do so sanh

Khi bạn thoát khỏi trình chỉnh sửa, Git sẽ tạo cam kết của bạn với thông báo cam kết đó (đã loại bỏ các nhận xét và khác biệt).

Ngoài ra, bạn có thể nhập nội dung thông báo cam kết của mình bằng lệnh cam kết bằng cách chỉ định nó sau cờ -m, như sau:

$ git commit -m "Story 182: fix benchmarks for speed"

[master 463dc4f] Story 182: fix benchmarks for speed

2 files changed, 2 insertions(+)

create mode 100644 README

Bạn có thể thấy rằng cam kết đã cung cấp cho bạn một số kết quả về chính nó: bạn đã cam kết với nhánh nào (master), tổng kiểm tra SHA-1 mà cam kết có (463dc4f), số lượng tệp đã được thay đổi và số liệu thống kê về các dòng được thêm và xóa trong làm.

Hãy nhớ rằng cam kết ghi lại ảnh chụp nhanh mà bạn thiết lập trong khu vực tổ chức của mình. Bất cứ điều gì bạn chưa thực hiện vẫn đang được sửa đổi ở đó; bạn có thể thực hiện một cam kết khác để thêm nó vào lịch sử của mình. Mỗi khi bạn thực hiện một cam kết, bạn đang ghi lại ảnh chụp nhanh của dự án mà bạn có thể hoàn nguyên hoặc so sánh với sau này.

Bỏ qua khu vực Staging

Mặc dù nó có thể cực kỳ hữu ích trong việc tạo ra các cam kết chính xác theo cách bạn muốn, nhưng khu vực tổ chức đôi khi phức tạp hơn một chút so với mức bạn cần trong quy trình làm việc của mình. Nếu bạn muốn bỏ qua khu vực tổ chức, Git cung cấp một phím tắt đơn giản. Việc thêm tùy chọn -a vào lệnh git commit làm cho Git tự động sắp xếp mọi tệp đã được theo dõi trước khi thực hiện cam kết, cho phép bạn bỏ qua phần git add:

Xoa file

Để xóa một tệp khỏi Git, bạn phải xóa tệp đó khỏi các tệp được theo dõi của mình (chính xác hơn là xóa tệp đó khỏi khu vực tổ chức của bạn) rồi cam kết. Lệnh git rm thực hiện điều đó và cũng xóa tệp khỏi thư mục làm việc của bạn để bạn không thấy nó là tệp không bị theo dõi vào lần tiếp theo.

Nếu bạn chỉ cần xóa tệp khỏi thư mục làm việc của mình, nó sẽ hiển thị trong vùng "Thay đổi không được tổ chức cho cam kết" (nghĩa là chưa được phân loại) của đầu ra trạng thái git của bạn:

$ rm aone.oa	$ git rm aone.oa
$ git status On branch main Changes not staged for commit: (use "git add/rm <file>..." to update what will be committed) (use "git restore <file>..." to discard changes in working directory) deleted: aone.oa no changes added to commit (use "git add" and/or "git commit -a")	git status On branch main Changes to be committed: (use "git restore --staged <file>..." to unstage) deleted: aone.oa

Một điều hữu ích khác mà bạn có thể muốn làm là giữ tệp trong cây đang làm việc nhưng xóa nó khỏi khu vực tổ chức của bạn. Nói cách khác, bạn có thể muốn giữ tệp trên ổ cứng của mình nhưng không để Git theo dõi nó nữa. Điều này đặc biệt hữu ích nếu bạn quên thêm nội dung nào đó vào tệp .gitignore 38 của mình và vô tình sắp xếp nó, chẳng hạn như một tệp nhật ký lớn hoặc một loạt tệp .a được biên dịch. Để thực hiện việc này, hãy sử dụng tùy chọn –cached:

Di chuyển tập tin

Không giống như nhiều VCS khác, Git không theo dõi chuyển động của tệp một cách rõ ràng. Nếu bạn đổi tên một tệp trong Git thì không có siêu dữ liệu nào được lưu trữ

trong Git cho biết bạn đã đổi tên tệp. Tuy nhiên, Git khá thông minh khi tìm ra điều đó sau thực tế - chúng ta sẽ xử lý việc phát hiện chuyển động của tệp sau đó.

$ git mv file_from file_to

Vì vậy, hơi khó hiểu khi Git có lệnh mv. Nếu bạn muốn đổi tên một tệp trong Git, bạn có thể chạy một cái gì đó như:

$ git mv aone.oa one.oa

$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed: aone.oa -> one.oa

Hoàn tác mọi thứ

Ở bất kỳ giai đoạn nào, bạn có thể muốn hoàn tác điều gì đó. Tại đây, chúng tôi sẽ xem xét một số công cụ cơ bản để hoàn tác các thay đổi mà bạn đã thực hiện. Hãy cẩn thận vì không phải lúc nào bạn cũng có thể hoàn tác một số thao tác hoàn tác này. Đây là một trong số ít khu vực trong Git mà bạn có thể mất một số công việc nếu làm sai.

Một trong những trường hợp hoàn tác phổ biến xảy ra khi bạn cam kết quá sớm và có thể quên thêm một số tệp hoặc bạn làm sai thông điệp cam kết của mình. Nếu bạn muốn làm lại cam kết đó, hãy thực hiện các thay đổi bổ sung mà bạn đã quên, sắp xếp chúng và cam kết lại bằng tùy chọn –amend:

Làm việc với điều khiển từ xa

Để có thể cộng tác trên bất kỳ dự án Git nào, bạn cần biết cách quản lý kho lưu trữ từ xa của mình. Kho lưu trữ từ xa là phiên bản dự án của bạn được lưu trữ trên Internet hoặc mạng ở đâu đó. Bạn có thể có một vài trong số chúng, mỗi cái thường ở dạng chỉ đọc hoặc đọc/ghi cho bạn. Cộng tác với những người khác bao gồm việc quản lý các kho lưu trữ từ xa này cũng như đẩy và kéo dữ liệu đến và đi từ chúng khi bạn cần

chia sẻ công việc. Quản lý kho lưu trữ từ xa bao gồm biết cách thêm kho lưu trữ từ xa, xóa các điều khiển từ xa không còn hợp lệ, quản lý các nhánh từ xa khác nhau và xác định chúng có được theo dõi hay không, v.v. Trong phần này, chúng tôi sẽ đề cập đến một số kỹ năng quản lý từ xa này.

Kho lưu trữ từ xa có thể nằm trên máy cục bộ của bạn. Hoàn toàn có khả năng là bạn có thể làm việc với một kho lưu trữ "từ xa", trong thực tế là bạn đang ở trên cùng một máy chủ. Từ "từ xa" không nhất thiết ngụ ý rằng kho lưu trữ ở một nơi nào khác trên mạng hoặc Internet, chỉ có nghĩa là nó ở nơi khác. Làm việc với một kho lưu trữ từ xa như vậy vẫn sẽ bao gồm tất cả các hoạt động đẩy, kéo và tìm nạp tiêu chuẩn như với bất kỳ điều khiển từ xa nào khác.

Hiển thị điều khiển từ xa của bạn

Để xem bạn đã định cấu hình máy chủ từ xa nào, bạn có thể chạy lệnh git remote. Nó liệt kê tên viết tắt của từng điều khiển từ xa mà bạn đã chỉ định. Nếu bạn đã sao chép kho lưu trữ của mình, ít nhất bạn sẽ thấy nguồn gốc - đó là tên mặc định mà Git đặt cho máy chủ mà bạn đã sao chép từ đó:

Bạn có thể sử dụng lệnh git remote -v, lệnh này hiển thị cho bạn các URL mà Git đã lưu trữ cho tên viết tắt được sử dụng khi đọc và ghi vào điều khiển từ xa đó:

```
$ git remote -v
origin https://github.com/schacon/ticgit (fetch)
origin https://github.com/schacon/ticgit (push)
```

Lưu ý rằng những điều khiển từ xa này sử dụng nhiều giao thức khác nhau; chúng tôi sẽ đề cập nhiều hơn về vấn đề này trong Tải Git trên Máy chủ.

Thêm kho lưu trữ từ xa

Chúng tôi đã đề cập và đưa ra một số minh họa về cách lệnh git clone ngầm thêm điều khiển từ xa gốc cho bạn. Đây là cách thêm một điều khiển từ xa mới một cách rõ ràng. Để thêm kho lưu trữ Git từ xa mới làm tên viết tắt mà bạn có thể tham khảo dễ dàng, hãy chạy git remote add <shortname> <url>:

```
$ git remote
origin
$ git remote add pb https://github.com/paulboone/ticgit
$ git remote -v
origin https://github.com/schacon/ticgit (fetch)
```

origin https://github.com/schacon/ticgit (push)
pb https://github.com/paulboone/ticgit (fetch)
pb https://github.com/paulboone/ticgit (push)

Bây giờ bạn có thể chạy lệnh git fetch pb để lấy kho lưu trữ

Tìm nạp và kéo từ điều khiển từ xa của bạn

Như bạn vừa thấy, để lấy dữ liệu từ các dự án từ xa, bạn có thể chạy:
git fetch <remote>
exp: git fetch origin

Lệnh đi đến dự án từ xa đó và lấy xuống tất cả dữ liệu từ dự án từ xa mà bạn chưa có. Sau khi thực hiện việc này, bạn sẽ có các tham chiếu đến tất cả các nhánh từ điều khiển từ xa đó mà bạn có thể hợp nhất hoặc kiểm tra bất kỳ lúc nào.

Nếu bạn sao chép một kho lưu trữ, lệnh sẽ tự động thêm kho lưu trữ từ xa đó dưới tên "origin". Vì vậy, git fetch Origin tìm nạp bất kỳ tác phẩm mới nào đã được đẩy tới máy chủ đó kể từ khi bạn sao chép (hoặc tìm nạp lần cuối từ) nó. Điều quan trọng cần lưu ý là lệnh git tìm nạp chỉ tải dữ liệu xuống kho lưu trữ cục bộ của bạn - nó không tự động hợp nhất dữ liệu đó với bất kỳ công việc nào của bạn hoặc sửa đổi những gì bạn hiện đang làm. Bạn phải hợp nhất nó theo cách thủ công vào tác phẩm của mình khi bạn đã sẵn sàng.

Nếu nhánh hiện tại của bạn được thiết lập để theo dõi một nhánh từ xa (xem phần tiếp theo và Phân nhánh Git để biết thêm thông tin), bạn có thể sử dụng lệnh git pull để tự động tìm nạp và sau đó hợp nhất nhánh từ xa đó vào nhánh hiện tại của bạn. Đây có thể là quy trình làm việc dễ dàng hoặc thoải mái hơn cho bạn; và theo mặc định, lệnh git clone sẽ tự động thiết lập nhánh chính cục bộ của bạn để theo dõi nhánh chính từ xa (hoặc bất kỳ nhánh mặc định nào được gọi) trên máy chủ mà bạn đã sao chép từ đó. Việc chạy git pull thường tìm nạp dữ liệu từ máy chủ mà bạn đã sao chép ban đầu và tự động cố gắng hợp nhất dữ liệu đó vào mã mà bạn hiện đang làm việc.