THE EXPERT'S VOICE®

SECOND EDITION

Pro Git

Tiếng Việt

Chương 1: Bắt đầu với Git

EVERYTHING YOU NEED TO KNOW ABOUT GIT

Scott Chacon and Ben Straub (Dịch bởi elinux.vn)

Apress®

Pro Git

Scott Chacon, Ben Straub, Dịch bởi elinux.vn

Version 2.1.86-15-gcc265d1, 2019-05-02

Table of Contents

Bắt đầu

Chương này trình bày kiến thức nền tảng về quản lí phiên bản, sau đó là làm thế nào để Git chạy trên hệ thống của bạn và cuối cùng là cài đặt nó để bắt đầu làm việc với nó. Cuối chương bạn sẽ hiểu sơ bộ về Git, tại sao bạn sử dụng nó.

Về quản lý phiên bản (version control)

“Quản lý phiên bản” là gì, và tại sao bạn nên quan tâm tới nó? Quản lý phiên bản là quá trình ghi lại những thay đổi của một tệp tin (file) hoặc nhiều tệp tin theo thời gian, từ đó bạn có thể quay lại phiên bản tại một thời điểm nhất định nào đó. Trên thực tế, có thể quản lí phiên bản cho bất kì loại tệp tin nào trên máy tính. Trong cuốn sách này lấy ví dụ là mã nguồn (source code) của phần mềm.

Nếu bạn là một nhà thiết kế Web hoặc nhà thiết kế đồ họa và muốn lưu giữ lại mọi phiên bản của bạn đã làm, Hệ thống quản lí phiên bản (Version Control System - gọi tắt là VCS) là một lựa chọn thông minh để giúp bạn làm việc đó. Nó cho phép bạn lấy lại bất kì một tệp tin hoặc cả dự án (project) bạn đã thực hiện, so sánh các phiên bản với nhau, dễ dàng tìm được chỗ gây ra lỗi và ai là người đã gây ra việc đó. Sử dụng VCS cũng giúp bạn khôi phục lại tệp tin dễ dàng trong trường hợp bị mất.

Hệ thống Quản lý phiên bản cục bộ (LVCS - Local Version Control System)

Nhiều người lựa chọn phương án Quản lý phiên bản bằng cách sao chép các tệp tin vào thư mục khác (với người khôn khéo hơn thì nó là thư mục có ghi lại các sự kiện xảy ra theo thời gian). Cách này rất phổ biến bởi vì nó đơn giản, nhưng nó cũng rất hay gây ra lỗi. Rất dễ dàng quên thư mục nào bạn đang sử dụng và dễ ghi tới tệp tin lỗi hoặc ghi đè tệp tin bạn không mong muốn.

Để giải quyết vấn đề này, các lập trình viên rất xa về trước đã phát triển Hệ thống quản lí phiên bản cục bộ (gọi tắt là LVCS), nó có cơ sở dữ liệu (database) đơn giản, cơ sở dữ liệu này giữ tất cả thay đổi của các tệp tin.

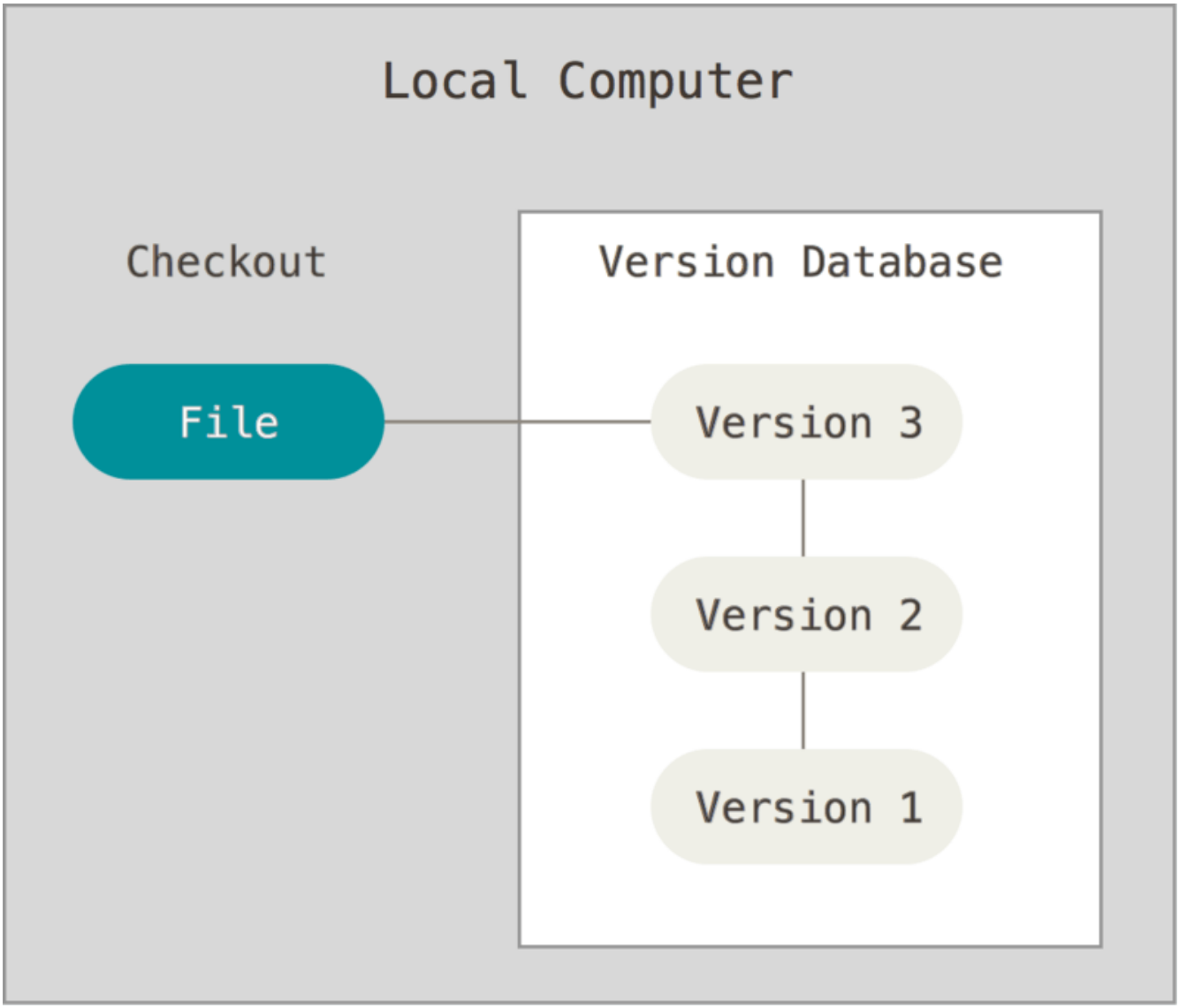

Figure 1. Sơ đồ hệ thống quản lí cục bộ.

Một trong những LVCS phổ biến được gọi là RCS, ngày nay nó vẫn có ở nhiều máy tính. RCS làm việc bằng cách giữ các bản vá (patch) (sự khác nhau giữa các phiên bản của tệp tin) trong định dạng đặc biệt trên đĩa cứng; Sau đó nó dùng các bản vá này để tạo lại tệp tin tại bất kì thời điểm nào.

Hệ thống quản lí phiên bản tập trung (gọi tắt là CVCS - Centralized Version Control System)

Vấn đề tiếp theo các nhà phát triển gặp phải là hợp tác với nhau trên các hệ thống khác nhau. Để giải quyết vấn đề này thì Hệ thống quản lý phiên bản tập trung (CVCSs) được phát triển. các CVSC này (như CVS, Subversion, và Perforce) có một máy chủ (server) chứa tất cả các tệp đã được quản lí phiên bản, và các máy khách (client) khác nhau lấy tệp tin từ máy chủ này. Trong nhiều năm, đây là chuẩn của các hệ thống quản lí phiên bản.

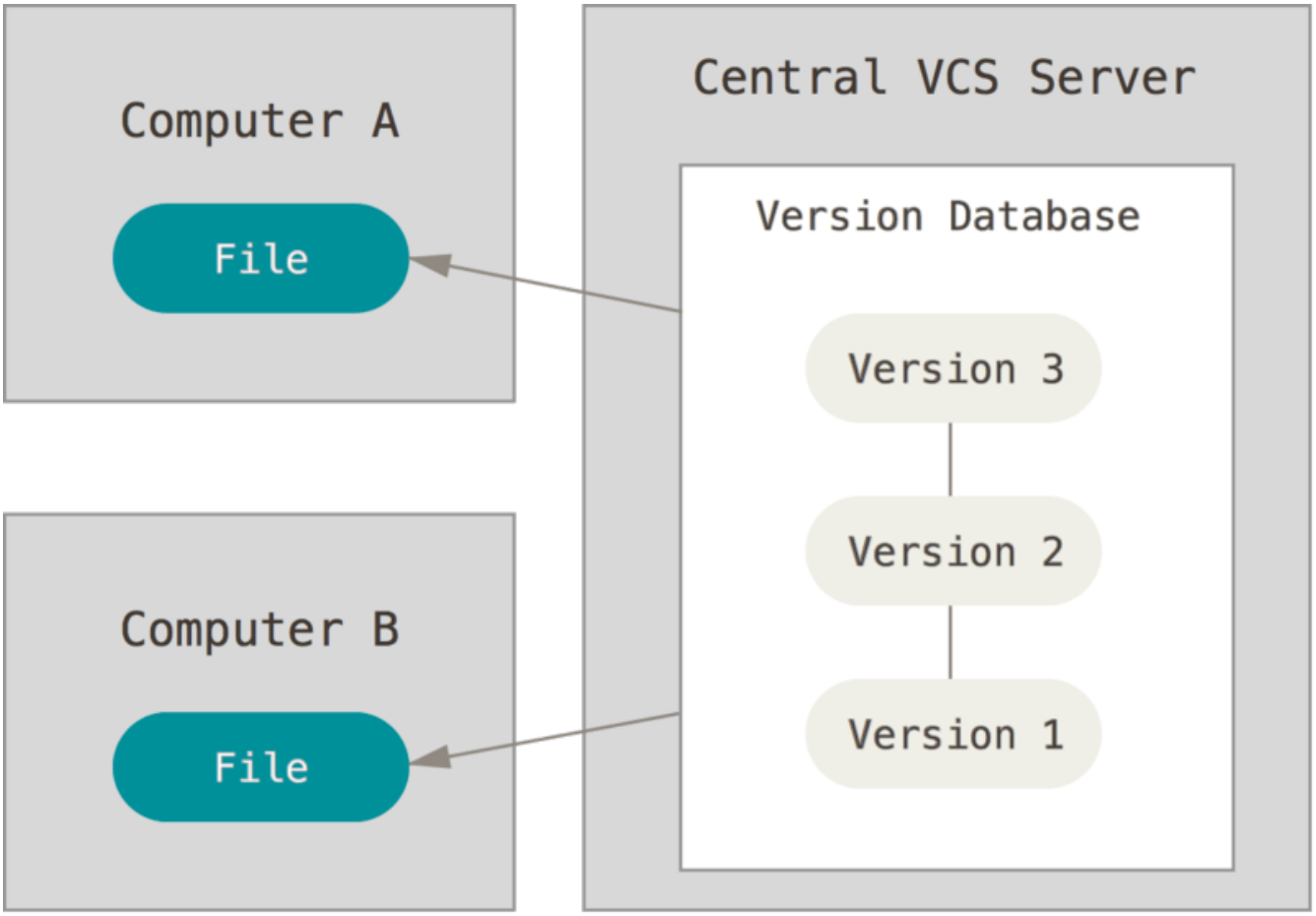

Figure 2. Sơ đồ hệ thống quản lí tập trung.

CVSC có nhiều ưu điểm, đặc biệt là so với Quản lí phiên bản kiểu cục bộ. Ví dụ, mọi người đều biết người khác đang làm cái gì trên Dự án mình đang tham gia. Các quản trị viên (Administrator) dễ dàng hơn việc phân quyền, và dễ dàng hơn so với việc phải quản lí riêng lẻ cơ sở dữ liệu ở từng máy khách.

Mặc dù vậy, CVSC cũng có những nhược điểm nghiêm trọng. Nếu trong thời gian máy chủ không hoạt động thì không ai có thể hợp tác hoặc lưu những thay đổi mà họ đang thực hiện. Nếu ổ đĩa cứng của máy chủ bị hỏng, thì toàn bộ dữ liệu sẽ mất mà chỉ còn lại dữ liệu của riêng các máy khách. Các hệ thống quản lí phiên bản cục bộ cũng có vấn đề này — Bởi vì lưu toàn bộ cái gì ở một nơi thì nguy cơ mất toàn bộ dữ liệu luôn thường trực.

Hệ thống quản lí phiên bản phân tán (gọi tắt là DVCS - Distributed Version Control System)

Hệ thống quản lí phân tán (DVCS) xuất hiện để giải quyết nhược điểm trên. Trong DVCS (như Git, Mercurial, Bazaar hoặc Darcs), các máy khách không chỉ lấy dữ liệu về phiên bản mới nhất của các tệp tin; mà còn sao chép toàn bộ kho chứa (repository) bao gồm toàn bộ lịch sử của Dự án. Bởi vậy, nếu máy chủ có hỏng, thì có thể dùng kho chứa của bất kì máy khách nào để khôi phục lại máy chủ. Mỗi bản sao ở trên máy khách chính là một bản dự phòng đầy đủ.

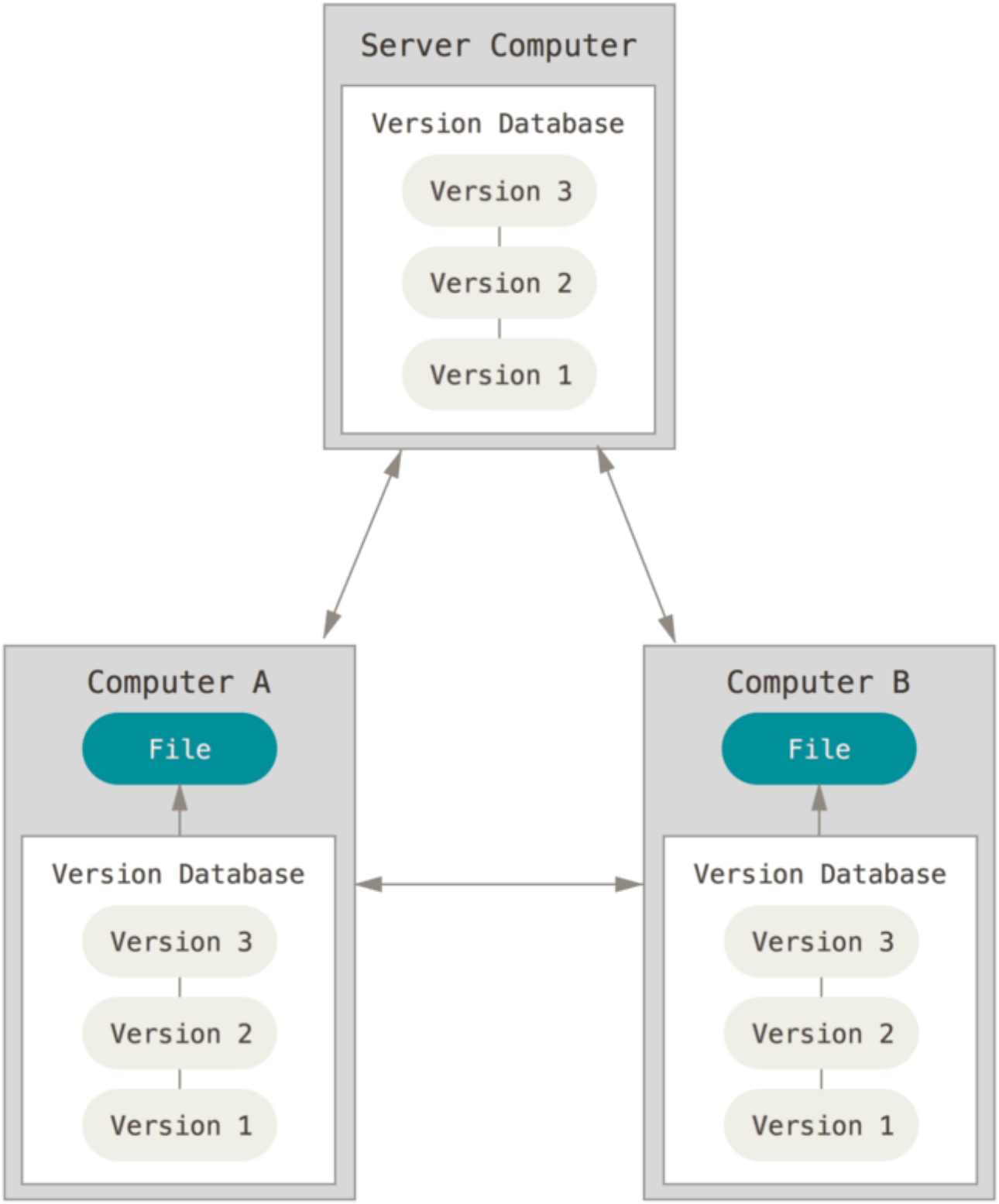

Figure 3. Sơ đồ hệ thống quản lí phân tán.

Hơn nữa, các DVCS làm việc khá tốt với nhiều kho chứa từ xa (remote repository), điều đó cho phép có thể hợp tác với nhiều nhóm khác nhau theo các cách khác nhau đồng thời trên cùng một dự án. Điều này cho phép bạn dùng nhiều loại Quy trình làm việc, mà điều này không thể thực hiện được với các hệ thống quản lí tập trung, như là các mô hình phân cấp.

Tóm tắt lịch sử của Git

Giống như nhiều thứ khác trong cuộc sống, Git bắt đầu với một chút phá hoại sáng tạo và tranh cãi nảy lửa.

Linux kernel là một dự án nguồn mở khá lớn. Trong thời gian duy trì dự án này quãng thời gian 1991 tới 2002, những thay đổi được thực hiện thông qua bản vá (patch) và tệp lưu trữ (archived files). Vào năm 2002, dự án này bắt đầu sử dụng một DVCS bản quyền tên là BitKeeper.

Vào năm 2005, mối quan hệ giữa cộng đồng phát triển Linux kernel và công ty thương mại phát triển BitKeeper bị đổ vỡ, và công cụ này bị thu hồi. Chính điều này đặt ra cho cộng đồng phát triển Linux (và cụ thể là Linux Torvalds, cha đẻ của Linux) phải phát triển một công cụ của riêng nó dựa trên những bài học mà họ học được trong quá trình sử dụng BitKeeper.

Một vài mục tiêu của hệ thống mới này là:

- Nhanh
- Đơn giản về thiết kế
- Hỗ trợ tốt cho quá trình phát triển phi tuyến (tức là, hàng nghìn nhánh song song)
- Hoàn toàn phân tán
- Có thể làm việc hiệu quả với những Dự án lớn như Linux kernel

Kể từ khi ra đời của nó vào năm 2005, Git đã tiến hóa và hoàn thiện để dễ dàng sử dụng mà vẫn giữ được các tiêu chí ban đầu trên. Nó nhanh kinh ngạc, nó rất hiệu quả với những Dự án lớn, và nó có một đặc trưng vượt trối là phân nhánh, đặc trưng này tốt không thể tin được cho phát triển phi tuyến(Xem ch03-git-branching.pdf).

Cơ bản về Git

Vậy, Git là gì? Đây là phần quan trọng để tiếp thu, bởi vì nếu bạn biết Git là gì và những nguyên lí làm việc của nó, thì khi sử dụng Git sẽ dễ dàng hơn nhiều. Khi bạn học Git, đầu tiên cố gắng xóa bỏ hết trong đầu bạn những thứ bạn có thể biết về VCS khác như CVS, Subversion hoặc Perforce — làm thế sẽ giúp bạn loại bỏ được sự nhầm lẫn khi sử dụng công cụ. Mặc dù, giao diện của Git khá giống với những VCS khác, nhưng Git lưu trữ và nghĩ về thông tin theo một cách rất khác, và chính việc hiểu rõ sự khác nhau giữa Git và các VCS sẽ giúp loại trừ nhầm lẫn trong khi sử dụng nó.

Snapshots, chứ không phải là Thay đổi

Sự khác biệt chính giữa Git và bất kì một VCS nào khác là cái cách mà Git nghĩ về dữ liệu của nó. Về mặt nguyên lí, hầu hết các hệ thống khác Git (như CVS, Subversion, Perforce, Bazar,...) lưu trữ thay đổi của tệp tin theo thời gian.

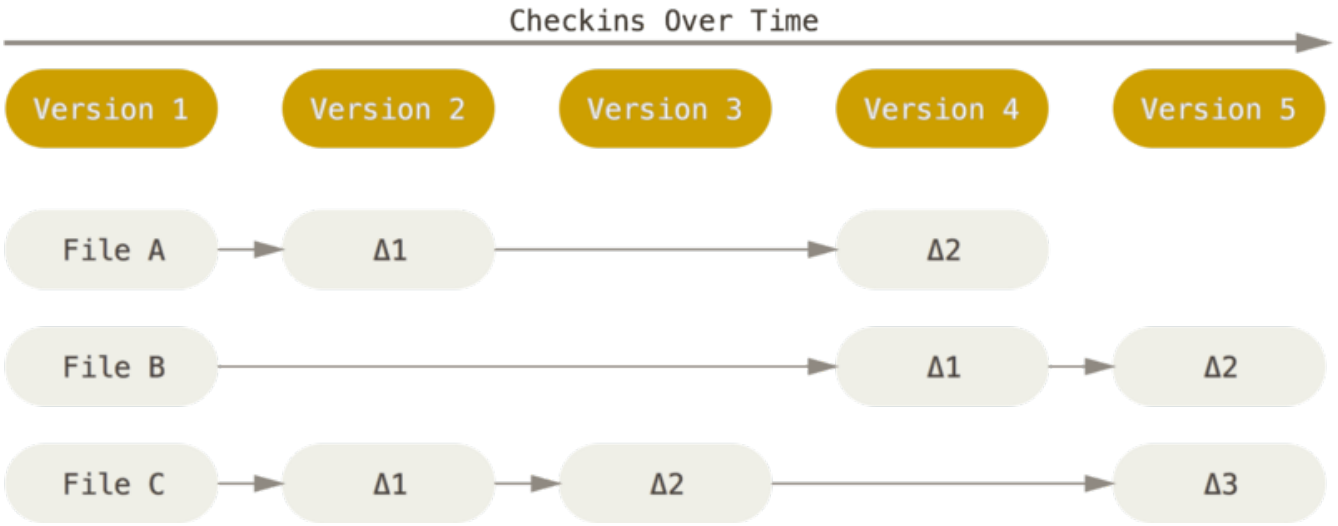

Figure 4. Lưu trữ dữ liệu kiểu chỉ lưu những thay đổi.

Git không nghĩ hoặc lưu trữ dữ liệu theo cách đó. Thay vào đó, Git nghĩ về dữ liệu là một chuỗi snapshot (nghĩa là, toàn bộ dữ liệu). Với Git, mỗi lần bạn commit, hoặc lưu trạng thái của dự án, Git sẽ tạo một snapshot và lưu tham chiếu tới snapshot đó. Để hiệu quả, nếu tệp tin không thay đổi, Git sẽ không lưu tệp tin đó lại, chỉ liên kết với nó ở snapshot trước. Git nghĩ về dữ liệu của nó giống hơn là **một chuỗi snapshot**.

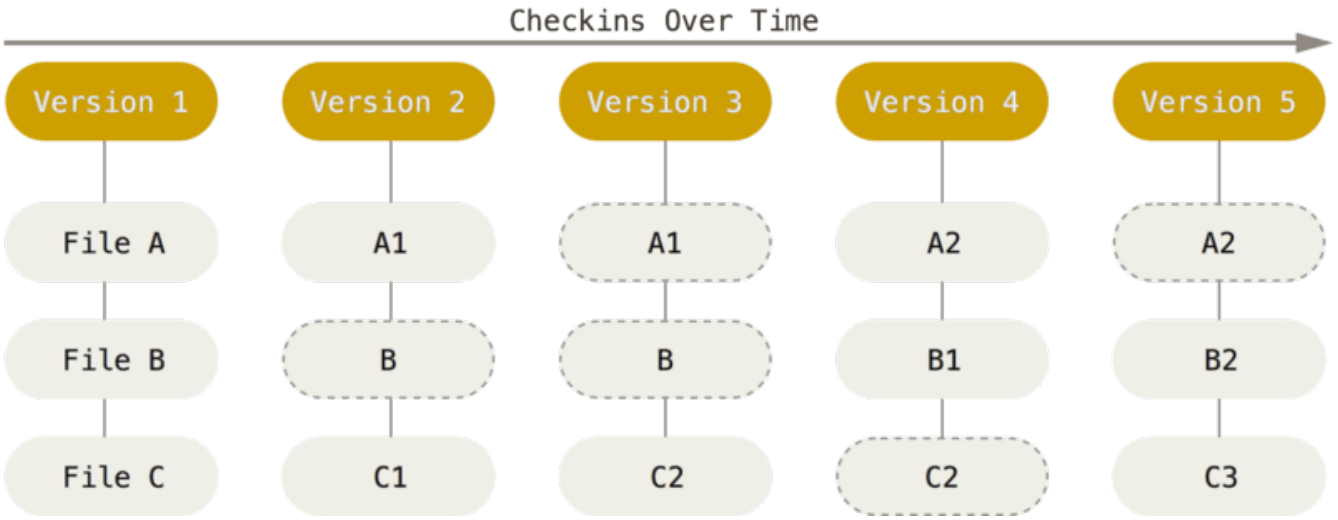

Figure 5. Lưu trữ dữ liệu theo kiểu snapshot theo thời gian.

Đây chính là sự khác nhau quan trọng giữa Git và các VCS khác. Sự khác biệt này làm cho Git cân nhắc lại mọi khí cạnh của quản lí phiên bản mà hầu hết các VCS khác sao chép từ thế hệ trước. Chình vì thế Git giống một hệ thống quản lí tập tin thu nhỏ (mini filesystem) được trang bị một vài tính năng rất mạnh, hơn là việc đơn thuần chỉ là một VCS. Chúng ta sẽ khám phá lợi ích có được của việc nghĩ dữ liệu theo cách này trong ch03-git-branching.pdf.

Gần như mọi hoạt động diễn ra tại chỗ

Hầu hết các hoạt động của Git chỉ cần các tệp và tài nguyên tại chỗ — nhìn chung không có thông tin cần từ máy tính khác. Nếu bạn sử dụng một CVCS, hầu hết các hoạt động phải sử dụng thông tin từ máy chủ, còn với Git toàn bộ thông tin đã ở trên máy tính của bạn, vì thế hầu như mọi hoạt động diễn ra tức thì, điều đó làm cho bạn có cảm giác là vị thần tốc độ đang phù hộ cho Git.

Ví dụ, để duyệt lịch sử của dự án, git không cần tới máy chủ để lấy lịch sử và sau đó hiển thị lên cho

bạn — rất đơn giản, nó chỉ đọc trực tiếp từ cơ sở dữ liệu ngay trên máy tính của bạn. Điều đó nghĩa rằng bạn sẽ nhìn thấy lịch sử của dự án hầu như ngay tức thì. Nếu bạn muốn xem những thay đổi của một tệp tin ở thời điểm hiện tại với thời điểm một tháng trước, git có thể tìm tệp tin đó ở thời điểm 1 tháng trước và tính toán sự khác nhau cho bạn, chứ không cần phải yêu cầu máy chủ làm việc đó hoặc kéo phiên bản tệp tin đó ở thời điểm 1 tháng trước từ máy chủ và sau đó mới thực hiện tính toán sự khác nhau.

Điều này cũng có nghĩa là rất ít việc không thể làm được nếu bạn đang offline hoặc ngắt vpn. Nếu bạn ở trên máy bay hoặc tàu hỏa và muốn làm việc, bạn vẫn có thể commit (tới cơ sở dữ liệu trên máy của bạn) cho tới khi bạn có thể kết nối internet để đẩy công việc của mình lên. Nếu bạn ở nhà và mạng internet không làm việc, bạn vẫn có thể làm việc. Đối với nhiều hệ thống khác, làm vậy một là không thể hoặc là khó khăn. Ví dụ, với perforce, bạn không thể làm nhiều thứ khi bạn không kết nối tới máy chủ; và với subversion và cvs, bạn có thể thay đổi các tệp, nhưng không thể commit những thay đổi tới cơ sở dữ liệu của bạn. Đây dường như không phải là điều gì lớn lao, nhưng bạn có thể ngạc nhiên về sự khác biệt lớn nó có thể làm.

Git có tính toàn vẹn

Mọi thứ trong Git được checksum (tạo mã băm) trước khi được lưu trữ và sau đó dùng chúng mã checksum để tham chiếu tới dữ liệu đó. Điều đó có nghĩa là không thể thay đổi bất kì nội dung của bất kì tệp tin hoặc thư mục nào mà qua mặt được Git. Chứ năng này là chức năng sẵn có của Git ở mức thấp nhất và đã thành triết lí của nó. Bạn không thể mất thông tin trong khi truyền hoặc lấy về một tệp hỏng mà Git không phát hiện ra nó.

Kỹ thuật Git sử dụng để checksum có tên là SHA-1 hash (gọi là mã SHA-1). Đây là một chuỗi gồm 40 kí tự hexa (cơ số 16) và được tính toán dựa trên nội dung của một tệp hoặc cấu trúc thư mục trong Git. Một mã SHA-1 trông giống như sau:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Bạn sẽ nhìn thấy chúng rất nhiều trong Git vì Git sử dụng chúng rất nhiều. Trong thực tế, Git lưu mọi thứ trong cơ sở dữ liệu không phải bằng tên tệp (filename) mà sử dụng mã SHA-1 nội dung của tệp đó.

Nói chung Git chỉ thêm dữ liệu

Khi bạn làm việc với Git, các hành động của nó hầu như chỉ thêm dữ liệu tới cơ sở dữ liệu. Rất khó để làm cho Git làm việc gì mà không khôi phục lại được hoặc xóa dữ liệu đi bằng mọi cách. Giống như với các VCS khác, bạn có thể mất hoặc làm rối tung những thay đổi mà bạn chưa commit, nhưng sau khi đã commit thì nó rất khó bị mất, đặc biệt nếu bạn đã đẩy cơ sở dữ liệu của mình tới một kho chứa khác.

Điều này làm cho việc sử dụng Git trở nên thích thú bởi vì chúng ta có thể thử nghiệm mà không có mối nguy hại về phá hỏng cái gì. Để tìm hiểu sâu hơn về việc Git lưu trữ dữ liệu của nó như thế nào và làm thế nào để khôi phục dữ liệu dường như đã mất, xem ch02-git-basics-chapter.pdf.

Ba trạng thái

Bây giờ, chú ý — đây là thứ chính cần nhớ về Git nếu bạn muốn học phần sau trôi chảy. Git có 3 trạng thái mà tệp tin trải qua: *committed*, *modified*, and *staged*:

- *Committed* nghĩa là dữ liệu đã được lưu trữ an toàn vào cơ sở dữ liệu của bạn.
- *Modified* nghĩa là bạn đã thay đổi tệp tin nhưng chưa commit tới cơ sở dữ liệu.
- *Staged* nghĩa là bạn đã đánh dấu tệp đó sẵn sàng được commit.

Điều này làm cho một dự án sử dụng Git để quản lí phiên bản có 3 phần chính: thư mục Git, thư mục làm việc (working tree), và vùng staging.

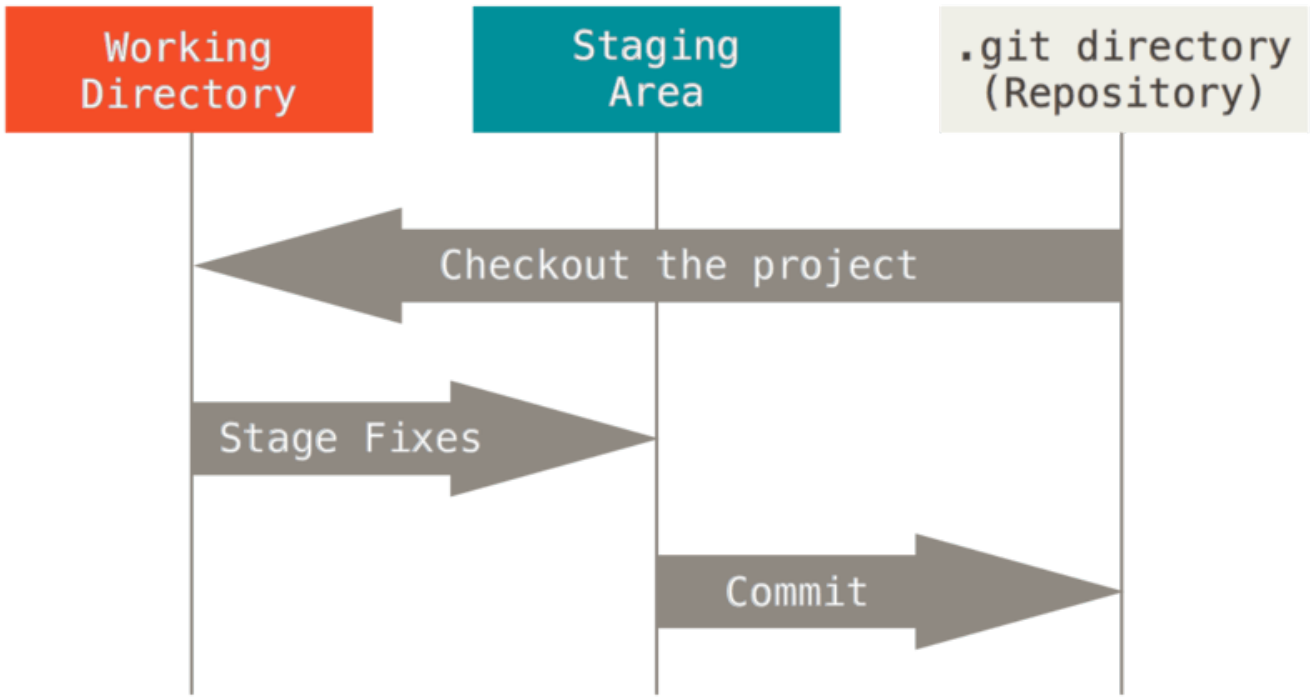

Figure 6. Thư mục làm việc, vùng staging, và thư mục Git.

Thư mục Git là nơi Git lưu trữ metadata và cơ sở dữ liệu cho dự án của bạn. Đây là phần quan trọng nhất của Git, và nó là cái được sao chép khi bạn clone một kho chứa từ một máy tính khác.

Thư mục làm việc là một checkout của bất kì một phiên bản nào đó của dự án. Các tệp này được lấy từ cơ sở dữ liệu trong thư mục Git (vừa nói ở trên) và được đặt trên đĩa cứng để bạn có thể sử dụng hoặc sửa.

Vùng staging là một tệp, nhìn chung được chứa trong thư mục Git, nó lưu trữ thông tin về cái sẽ ở trong commit tiếp theo. Tên kỹ thuật của nó theo lối Git là “index”, nhưng cụm từ “vùng staging” cũng ổn.

Vậy, Quy trình làm việc với Git giờ sẽ là:

1. Bạn sửa các tệp trong thư mục làm việc của bạn.
2. Bạn tùy ý stage những thay đổi (nhưng thay đổi mà bạn muốn nó ở trong commit tiếp theo) đó để nó ở trong vùng staging.
3. Bạn thực hiện commit, những thứ ở trong vùng staging sẽ được lưu vĩnh viễn vào trong thư mục

Git.

Nếu một phiên bản nào đó của một tệp ở trong thư mục Git, nó được xem như là commited. Nếu nó được sửa và được đưa tới vùng staging, nó ở trạng thái staged. Và nếu nó được thay đổi so với thời điểm nó được checkout nhưng chưa được staged, nó ở trạng thái modified. Trong ch02-git-basics-chapter.pdf, bạn sẽ học sâu hơn về các trạng thái này và cách làm thế nào bạn có thể tận dụng điểm mạnh của chúng hoặc hoàn toàn bỏ qua bước stage.

Dòng lệnh

Có nhiều cách khác nhau để sử dụng Git: sử dụng bằng dòng lệnh hoặc sử dụng các phiên bản có giao diện đồ họa (GUI). Trong cuốn sách này, chúng ta sẽ sử dụng Git bằng dòng lệnh. Dòng lệnh là nơi duy nhất bạn có thể chạy tất cả các lệnh của Git — hầu hết GUI chỉ thực thi một phần chức năng của Git. Và nếu bạn biết sử dụng Git bằng dòng lệnh, bạn cũng có thể hiểu làm thế nào để sử dụng các phiên bản giao diện đồ họa khác. Ngược lại chưa chắc đã đúng. Ngoài ra, trong khi công cụ dòng lệnh được cài và sẵn có mặc định, thì giao diện đồ họa là lựa chọn của riêng mỗi cá nhân.

Bởi vậy, chúng tôi mong rằng bạn biết làm thế nào để mở Terminal trong hệ điều hành Mac hoặc Command Prompt hoặc Powershell trong Windows. Nếu bạn không biết cái chúng tôi đang nói ở đây, bạn cần dừng lại và nghiên cứu nhanh để có thể theo kịp các ví dụ và các mô tả trong cuốn sách này.

Cài Git

Trước khi bắt đầu sử dụng Git, bạn phải làm cho nó có ở trên máy tính của mình. Thậm chí nếu nó đã được cài, tốt hơn là cập nhật phiên bản mới nhất. Bạn có thể cài nó bằng nhiều cách: cài bằng gói đã được đóng sẵn (package), hoặc qua bộ cài đặt khác, hoặc tải mã nguồn rồi tự biên dịch.

> Cuốn sách này được viết sử dụng Git phiên bản **2.0.0**. Mặc dù, hầu hết các lệnh chúng tôi sử dụng làm việc tốt với các phiên bản trước, nhưng một vài có thể không hoặc hành vi của chúng có khác đôi chút nếu bạn sử dụng phiên bản Git cũ hơn. Vì Git rất tuyệt hảo trong việc duy trì tương thích cho nên bất kì phiên bản nào mới hơn 2.0 đều ổn.

Cài Git trên Linux

Nếu bạn muốn cài các công cụ cơ bản của Git trên Linux thông qua binary installer, nhìn chung bạn có thể thực hiện thông qua công cụ quản lí gói (package management tool) có sẵn trong phân phối Linux bạn đang dùng. Nếu bạn sử dụng Fedora (hoặc bất kì phân phối nào dựa trên RPM như RHEL hoặc CentOS), bạn có thể sử dụng `dnf`:

```
$ sudo dnf install git-all
```

Nếu bạn sử dụng phân phối nào dựa trên Debian, như Ubuntu, thử `apt`:

```
$ sudo apt install git-all
```

Với một số hệ điều hành Linux, Unix khác, lệnh cài đặt có tại địa chỉ http://git-scm.com/download/linux.

Cài trên Mac

Có vài cách để cài Git trên Mac. Cách dễ dàng nhất có lẽ là cài Xcode Command Line Tools. Trên Mavericks (10.9) hoặc cao hơn bạn có thể làm điều này đơn giản bằng thử chạy lệnh *git* từ Terminal.

```
$ git --version
```

Nếu bạn chưa cài nó, nó sẽ nhắc bạn cài nó.

Nếu bạn muốn nâng cấp phiên bản, bạn có thể cài nó sử dụng binary installer tại địa chỉ http://git-scm.com/download/mac.

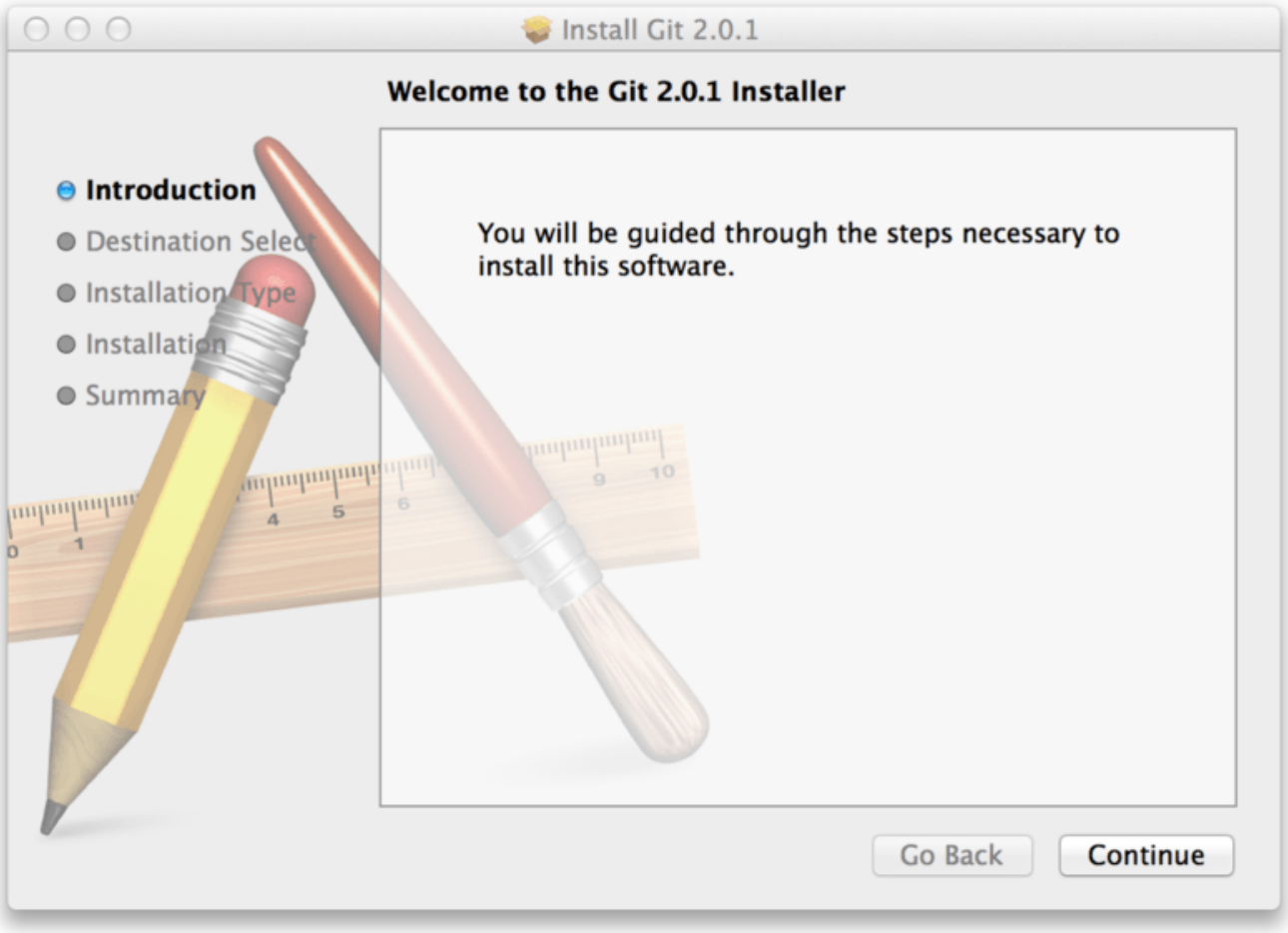

Figure 7. Git macOS Installer.

Ngoài ra, khi cài GitHub cho Mac thì Git cũng được cài kèm theo. Công cụ GUI cho Git cũng có lựa chọn cho phép cài các công cụ dòng lệnh. Bạn có thể tải công cụ đó từ Github cho Mac tại http://mac.github.com.

Cài đặt trên Windows

Có một vài cách để cài Git trên Windows. Bản chính thức mới nhất sẵn có trên trang web của Git tại địa chỉ http://git-scm.com/download/win, click vào quá trình tải xuống sẽ tự động bắt đầu. Lưu ý, đây là một Dự án tên là `Git cho Windows`, nó khác với Dự án Git; để biết thêm thông tin về dự án này, xem tại địa chỉ https://git-for-windows.github.io/.

Để có được quá trình cài tự động, bạn có thể sử dụng Git Chocolatey package. Lưu ý rằng Chocolatey package được duy trì bởi cộng đồng.

Cách khác để cài Git là cài GitHub Desktop. Bộ cài này chứa cả phiên bản dòng lệnh và GUI. Nó cũng làm việc tốt với Powershell, và thiết lập solid credential caching and sane CRLF settings. Chúng ta sẽ tìm hiểu nhiều hơn về những điều này phía sau, nhưng chỉ cần nói rằng chúng là những thứ bạn muốn. Bạn có thể tải GitHub Desktop tại địa chỉ GitHub Desktop website.

Cài đặt từ mã nguồn (source code)

Một vài người có thể thấy hữu ích khi cài Git từ mã nguồn, bởi vì bạn sẽ có được phiên bản mới nhất. Bộ cài sẵn có thường cũ hơn một chút, trong vài năm trở lại đây khi Git trở nên tốt hơn, sự khác biệt phiên bản thường là không lớn lắm.

Nếu bạn muốn cài Git từ mã nguồn, bạn cần có các thư viện mà Git cần như sau: autotools, curl, zlib, openssl, expat, and libiconv. Ví dụ, nếu bạn đang dùng hệ thống có `dnf` (như Fedora) hoặc `apt-get` (như các hệ thống gốc từ Debian), bạn có thể sử dụng một trong các lệnh sau để cài các thư viện dành cho Git:

```
$ sudo dnf install dh-autoreconf curl-devel expat-devel gettext-devel \
  openssl-devel perl-devel zlib-devel
$ sudo apt-get install dh-autoreconf libcurl4-gnutls-dev libexpat1-dev \
  gettext libz-dev libssl-dev
```

Để có thể thêm những tài liệu trong nhiều định dạng (doc, html, info), thì cần thêm các thư viện (Chú ý: đối với RHEL và các phân phối gốc từ RHEL như CentOS and Scientific Linux sẽ phải cho phép kho chứa EPEL để tải gói `docbook2X`):

```
$ sudo dnf install asciidoc xmlto docbook2X
$ sudo apt-get install asciidoc xmlto docbook2x
```

Nếu bạn đang sử dụng các phân phối gốc từ Debian (như Debian, Ubuntu, các loại Ubuntu), bạn có thể cần gói `install-info`:

```
$ sudo apt-get install install-info
```

Nếu bạn đang sử dụng các phân phối dùng RPM để quản lí gói (như Fedora/RHEL/các loại hệ điều hành RHEL), bạn cần gói `getopt` (gói này được cài sẵn ở các phân phối gốc từ Debian):

```
$ sudo dnf install getopt
$ sudo apt-get install getopt
```

Ngoài ra, nếu bạn đang sử dụng Fedora/RHEL/các loại hệ điều hành RHEL, bạn cần thực hiện điều này:

```
$ sudo ln -s /usr/bin/db2x_docbook2texi /usr/bin/docbook2x-texi
```

Vì sự khác nhau của tên binary.

Khi bạn đã có tất cả các thư viện cần này, bạn lấy mã nguồn (dạng tarball) mới nhất từ một số nơi sau: https://www.kernel.org/pub/software/scm/git hoặc https://github.com/git/git/releases.

Sau đó, biên dịch vài cài đặt:

```
$ tar -zxf git-2.0.0.tar.gz
$ cd git-2.0.0
$ make configure
$ ./configure --prefix=/usr
$ make all doc info
$ sudo make install install-doc install-html install-info
```

Sau khi làm xong, bạn có thể thử sử dụng Git:

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

Thiết lập Git cho sử dụng lần đầu

Bây giờ, bạn đã có Git trên hệ thống của bạn, bạn sẽ muốn làm một vài thứ để điều chỉnh môi trường Git của bạn. Bạn chỉ phải làm công việc này duy nhất một lần trên bất kì máy tính nào. Bạn có thể thay đổi chúng bất kì thời điểm nào bằng chạy lại các lệnh.

Git có một công cụ được gọi là `git config` cho phép bạn get và set các biến cấu hình để kiểm soát việc Git trông và làm việc như thế nào. Các biến này có thể được lưu ở ba nơi khác nhau:

1. Tệp `/etc/gitconfig`: chứa các biến được áp dụng cho tất cả user trên hệ thống và tất cả các repository. Nếu bạn truyền `--system` tới lệnh `git config`, nó đọc và ghi từ tệp này. (Bởi vì đây là một tệp cấu hình hệ thống, bạn sẽ cần phải là quản trị viên hoặc user có quyền superuser để thay đổi tệp đó.)
2. Tệp `~/.gitconfig` hoặc `~/.config/git/config`: chứa các giá trị riêng cho mỗi user. Bạn có thể làm Git đọc và ghi tới file này bằng truyền `--global`, và điều đó sẽ làm ảnh hưởng tới *toàn bộ* kho chứa bạn làm việc trên hệ thống.
3. Tệp `config` trong thư mục Git (là thư mục `.git/config`) của kho chứa bạn đang làm việc. Bạn có thể điều khiển Git đọc và ghi tới tệp này bằng `--local`. Trong thực tế, đây là lựa chọn mặc định,

tức là nếu bạn dùng lệnh `git config` không có cờ thì nó sẽ hiểu là `--local`. (Tuy nhiên, để chế độ mặc định này hoạt động đúng, bạn cần đang ở một nơi nào đó bên trong kho chứa Git)

Giá trị biến ở tệp mức sau sẽ ghi đè giá trị biến ở tệp mức trước. Ví dụ, cùng một biến cùng xuất hiện ở `.git/config` và `/etc/gitconfig` thì Git sẽ sử dụng giá trị ở tệp `.git/config`.

Trên các hệ thống Windows, Git tìm kiếm tệp `.gitconfig` trong thư mục `$HOME` (thông thường là thư mục `C:\Users\$USER`). Nó cũng vẫn tìm kiếm `/etc/gitconfig`, mặc dù, thư mục này là tương đối so với thư mục MSys - là bất kì thư mục mà bạn chọn khi cài Git. Nếu bạn đang sử dụng Git phiên bản 2.x hoặc cao hơn cho Windows, cũng có một tệp cấu hình mức hệ thống tại `C:\Documents and Settings\All Users\Application Data\Git\config` trên Windows XP, và `C:\ProgramData\Git\config` từ Windows Vista trở đi. Tệp cấu hình này chỉ có thể thay đổi khi chạy lệnh `git config -f <file>` bằng quyền quản trị (admin).

Thông tin nhận dạng

Thứ đầu tiên bạn nên làm khi cài Git là thiết lập username và địa chỉ email. Điều này quan trọng bởi vì mỗi Git commit sử dụng thông tin này, và thông tin này sẽ được đưa vào commit mỗi khi bạn thực hiện commit:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Nhắc lại, bạn chỉ cần làm điều này duy nhất một lần bằng sử dụng `--global`, bởi vì Git luôn luôn sử dụng thông tin đó cho bất kì việc gì bạn làm trên hệ thống đó. Nếu bạn muốn ghi đè thông tin này bằng user name hoặc địa chỉ email khác cho riêng một Dự án nào, bạn có thể chạy lệnh với không có `--global` khi bạn ở trong Dự án đó.

Nhiều công cụ GUI giúp bạn làm công việc này khi lần đầu tiên bạn chạy nó.

Bộ soạn thảo của bạn

Bây giờ, thông tin nhận dạng của bạn đã được thiết lập, bạn có thể cấu hình Bộ soạn thảo mặc định được sử dụng khi Git cần bạn nhập thông tin. Nếu bạn không cấu hình, Git sử dụng bộ soạn thảo mặc định của hệ thống

Nếu bạn muốn sử dụng một Bộ soạn thảo khác, ví dụ Emacs, bạn sử dụng lệnh sau:

```
$ git config --global core.editor emacs
```

Trên hệ thống Windows, nếu bạn sử dụng Bộ soạn khác, bạn phải chỉ rõ đường dẫn tới Bộ soạn thảo đó.

Trong trường hợp của Notepad++, một Bộ soạn thảo cho lập trình phổ biến, bạn muốn sử dụng phiên bản 32-bit, vì tại thời điểm viết cuốn sách này phiên bản 64-bit không hỗ trợ tất cả plug-in. Nếu bạn ở trên hệ thống Windows 32-bit, hoặc bạn có một bộ soạn thảo 64-bit trên Windows 64-bit, bạn sử dụng lệnh kiểu như sau:

```
$ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe'
-multiInst -nosession"
```

Nếu bạn có một bộ soạn thảo 32-bit trên hệ thống 64-bit, chương trình sẽ được cài trong `C:\Program Files (x86)`:

```
$ git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe'
-multiInst -nosession"
```

Vim, Emacs và Notepad++ là các Bộ soạn thảo phổ biến thường được sử dụng bởi các nhà phát triển trên các hệ thống kiểu Unix như Linux, macOS hoặc hệ thống Windows. Nếu bạn không quen thuộc với các Bộ soạn thảo này, bạn có thể tìm kiếm các hướng dẫn để cài đặt sử dụng Bộ soạn thảo mà bạn ưa thích với Git.

Nếu bạn không thiết lập Bộ soạn thảo thì bạn có thể vướng vào trạng thái bối rối (tức là, không biết nguyên nhân tại sao lại vậy) khi Git cố gắng chạy nó. Ví dụ, trên hệ thống Windows, Git có thể không thực hiện được công việc tiếp theo khi không khởi chạy thành công Bộ soạn thảo.

Kiểm tra các thiết lập của bạn

Nếu bạn muốn kiểm tra các thiết lập cấu hình của bạn, bạn sử dụng lệnh `git config --list`, lệnh này sẽ liệt kê tất cả các thiết lập tại thời điểm chạy lệnh:

```
$ git config --list
user.name=John Doe
user.email=johndoe@example.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

Có trường hợp bạn nhìn thấy biến nào đó xuất hiện nhiều lần, điều này là bởi vì Git đọc giá trị đó từ nhiều tệp khác nhau (Ví dụ, tệp ``/etc/gitconfig` và `~/.gitconfig`) Trong trường hợp này, Git sử dụng giá trị cuối cùng trong danh sách để sử dụng.

Hoặc, bạn cũng có thể kiểm tra rằng Git đang dùng giá trị nào của biến đó bằng lệnh `git config <key>`:

```
$ git config user.name
John Doe
```

Vì Git đọc biến cấu hình từ nhiều tệp khác nhau, cho nên có thể bạn sẽ có giá trị không mong muốn và bạn không biết tại sao. Trong trường hợp đó, bạn có thể hỏi Git xem là giá trị đó là từ tệp nào bằng lệnh:

```
$ git config --show-origin rerere.autoUpdate
file:/home/johndoe/.gitconfig   false
```

Tìm trợ giúp

Nếu bạn muốn tìm sự trợ giúp khi sử dụng Git, có hai cách tương đương nhau để đọc thông tin từ manpage:

```
$ git help <verb>
$ man git-<verb>
```

Ví dụ, bạn có thể đọc manpage của lệnh `git config` bằng chạy

```
$ git help config
```

Lệnh này thuận tiện bởi vì bạn có thể truy cập chúng mọi nơi, thậm chí là offline. Nếu manpage và cuốn sách này là chưa đủ và bạn cần tìm hiểu sâu hơn, bạn có thể thử Freenode IRC server (irc.freenode.net) với kênh `#git` hoặc `#github`. Các kênh này thường có hàng trăm người, họ có kiến thức về Git và thường sẵn sàng trợ giúp.

Ngoài ra, nếu bạn không cần đọc manpage đầy đủ mà chỉ cần lướt qua xem các cờ của lệnh, bạn sử dụng lựa chọn `-h` hoặc `--help` như sau:

```
$ git add -h
usage: git add [<options>] [--] <pathspec>...

    -n, --dry-run         dry run
    -v, --verbose         be verbose

    -i, --interactive     interactive picking
    -p, --patch           select hunks interactively
    -e, --edit            edit current diff and apply
    -f, --force           allow adding otherwise ignored files
    -u, --update          update tracked files
    -N, --intent-to-add   record only the fact that the path will be added later
    -A, --all             add changes from all tracked and untracked files
    --ignore-removal      ignore paths removed in the working tree (same as --no-all)
    --refresh             don't add, only refresh the index
    --ignore-errors       just skip files which cannot be added because of errors
    --ignore-missing      check if - even missing - files are ignored in dry run
    --chmod <(+/-)x>      override the executable bit of the listed files
```

Tóm tắt chương

Kết thúc chương bạn phải hiểu cơ bản Git là gì và nó khác với các Hệ quản lí phiên bản tập chung CVSC (Centralized Version Control Systems) khác như thế nào. Trên hệ thống của bạn có một phiên bản Git đang chạy với các thiết lập nhận dạng cá nhân đã được thực hiện. Bây giờ là thời điểm học những thứ cơ bản về Git.

Index