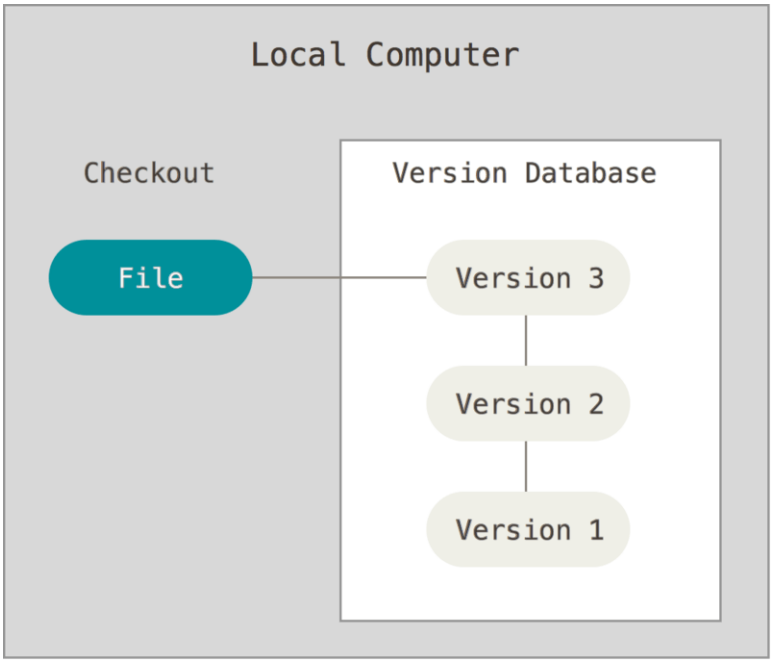
Báo cáo nghiên cứu giữa kỳ: Quản lý phiên bản với Git

Nguyễn Thị Ngọc Mai - Viettel Digital Talents 2024

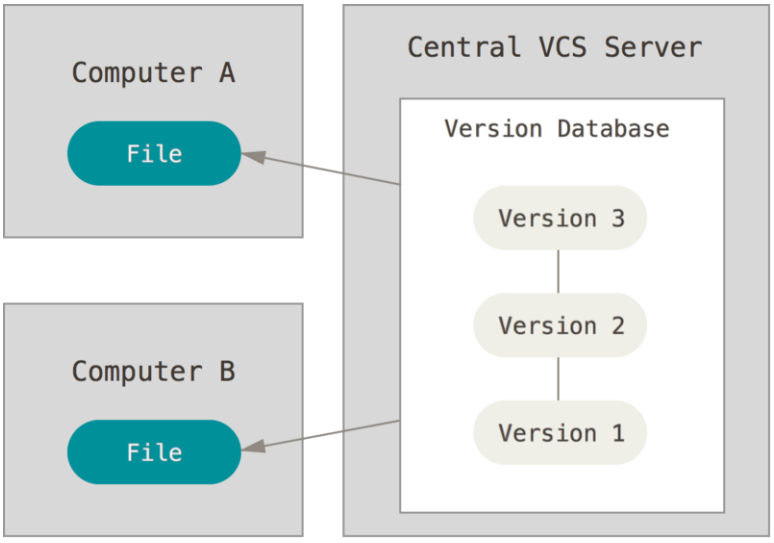
Quản lý phiên bản

1. Quản lý phiên bản

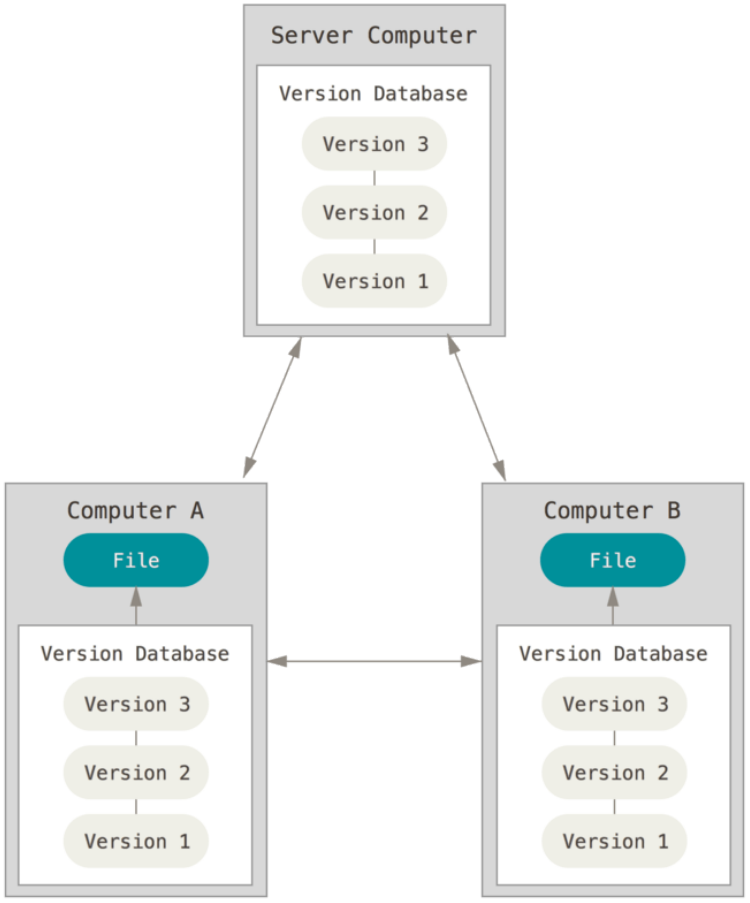
- **Khái niệm**: Quản lý phiên bản hay Version management, là một hệ thống theo dõi và kiểm soát các thay đổi trong các tệp hoặc tập hợp các tệp theo thời gian để bạn có thể quay lại một phiên bản cụ thể nếu cần. Điều này đặc biệt hữu ích trong các ngữ cảnh như phát triển phần mềm, nơi một đội ngũ nhà phát triển cần làm việc cùng nhau trên cùng một dự án và theo dõi và quản lý tất cả các thay đổi và điều chỉnh.
- **Các loại hệ thống quản lý phiên bản (VCS):**
- **Local Version Control Systems**



- **Centralized Version Control Systems**

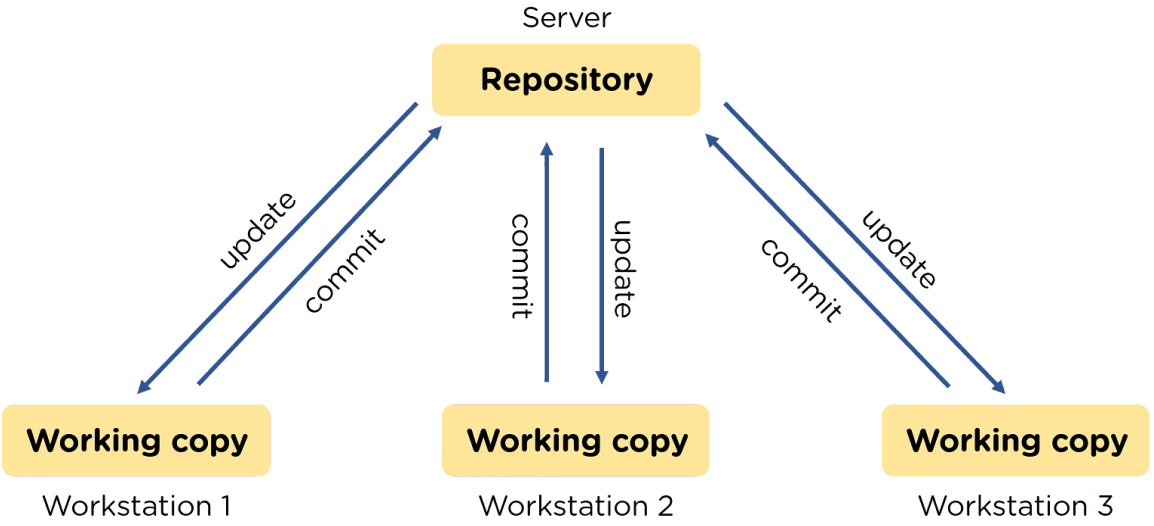


- **Distributed Version Control Systems**



- **Mô tả quá trình quản lý phiên bản**

- Giả sử trong một dự án có 3 người (3 máy) cùng làm việc tại 3 địa điểm khác nhau và có 1 server phụ trách lưu trữ mã nguồn của dự án (repository). 3 máy sẽ dùng chung 1 repository, hay đúng hơn là mỗi máy sẽ có 1 bản copy của repo đó để làm vệc và một hoạt động thêm, sửa, xóa sẽ được lưu ở bản repo copy đó.
- Server sẽ phụ trách việc quản lý các thay đổi mà các máy đã commit, chịu trách nhiệm update lại mã nguồn dự án để các máy luôn được cập nhật mã nguồn mới nhất của dự án
- Cơ chế này sẽ giúp cho việc làm việc với nhiều người trở nên dễ dàng hơn, nếu như mã nguồn của 1 trong 3 máy không hoạt động được, thì repo trên server sẽ không bị ảnh hưởng và các máy khác vẫn có thể tiếp tục phát triển dự án.



2. Lợi ích của việc quản lý phiên bản

- **Quản lý và bảo vệ mã nguồn**

Hệ thống quản lý phiên bản sẽ giúp lưu trữ, quản lý mã nguồn đội ngũ phát triển bằng cách lưu trữ tất cả các thay đổi liên quan đến mã nguồn. Nó cũng giúp bảo vệ mã nguồn khỏi các lỗi mà người phát triển tạo ra.

- **Keep track tất cả các thay đổi về mã nguồn**

Nhóm làm việc trong dự án liên tục tạo ra các mã nguồn mới và liên tục sửa đổi mã hiện có. Những thay đổi này được ghi lại để tham khảo trong tương lai và có

thể được sử dụng nếu cần trong tương lai để phát hiện nguyên nhân của bất kỳ vấn đề hay lỗi cụ thể nào.

- **Xem lại các phiên bản trước của mã nguồn**

Vì tất cả các phiên bản của mã đều được lưu nên điều này giúp người phát triển có thể quay lại bất kỳ lúc nào và so sánh các phiên bản mã trước đó để giúp sửa lỗi đồng thời giảm sự gián đoạn cho tất cả các thành viên trong nhóm.

- **Dễ dàng làm việc nhóm**

Việc sử dụng một hệ thống quản lý phiên bản sẽ cho phép nhiều người có thể cùng làm việc trên một mã nguồn hơn.

3. Tools và frameworks thường sử dụng

- **Version Control:**
  - Version Control Systems VCS: **Git**, Subversion,...
  - Repository Manager: **Gitlab**, **Github**, Bitbucket,...
- **Package & Release:** SemVer (Sematic-release)
- **Configuration Management**: **Git**, Helm (nếu deploy trên k8s-only)
- **Workflows & Frameworks**
  - Workflows: Reature Branch, Gitflow, Trunk-based
  - Frameworks: **Gitops**,...

Hệ thống quản lý phiên bản Git

1. Git là gì?

Git là **phần mềm quản lý mã nguồn phân tán** được phát triển bởi Linus Torvalds vào năm 2005, ban đầu dành cho việc phát triển nhân Linux. Hiện nay, Git trở thành một trong các phần mềm quản lý mã nguồn phổ biến nhất. Git là phần mềm mã nguồn mở được phân phối theo giấy phép công cộng GPL2 và có khả năng chạy trên nhiều hệ điều hành khác nhau như Linux, Windows, Mac OSX v.v..

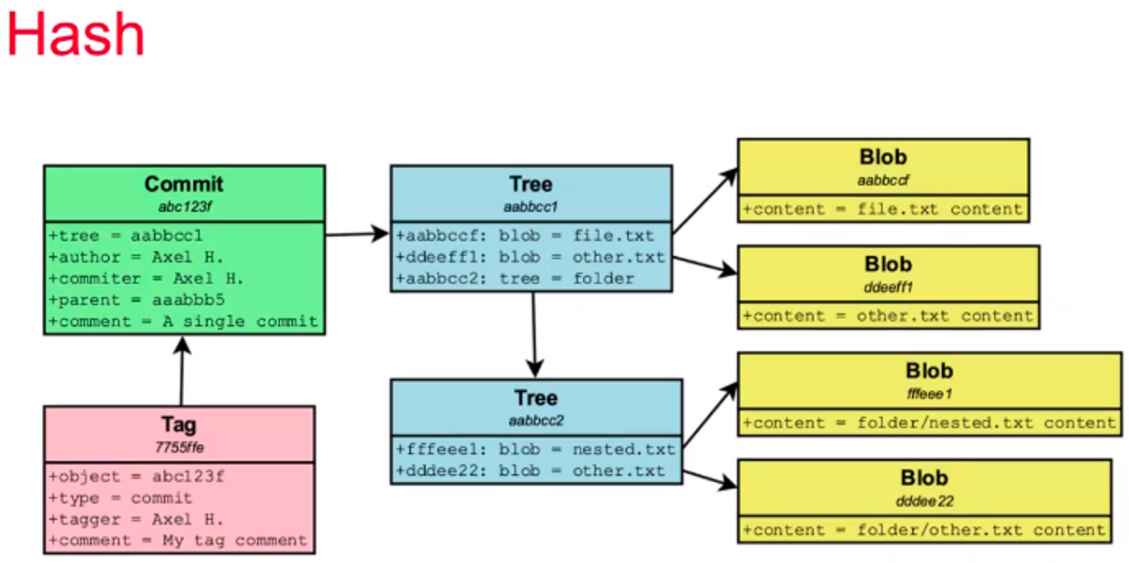
2. Một số khái niệm cơ bản trong Git

- **Repository**
  - Repository (nhà kho) hay được gọi tắt là Repo đơn giản là nơi chứa/cơ sở dữ liệu (database) tất cả những thông tin cần thiết để duy trì và quản lý các sửa đổi và lịch sử của dự án.
  - Trong Repo có 2 cấu trúc dự liệu chính là Object Store và Index. Tất cả dữ liệu của Repo đều được chứa trong thư mục bạn đang làm việc dưới dạng folder ẩn có tên là.git (không có phần tên trước dấu chấm).
- **Object store**
  - Object store là trái tim của Git, nó chứa dữ liệu nguyên gốc (original data files), các file log ghi chép quá trình sửa đổi, tên người tạo file, ngày tháng và các thông tin khác. Git có bốn loại object là: *Blobs, Trees, Commits, Tags*
- **Blobs:**
  - là file nhị phân có thể chứa được mọi loại dữ liệu bất kể là dữ liệu của chương trình gì.
- **Trees:**
  - lớp đại diện cho thông tin thư mục như thông tin định danh của blob, đường dẫn, chứa một ít metadata chứa thông tin cấu trúc và các thư mục nhỏ có trong thư mục đó.
- **Commits:**
  - Chứa metadata có thông tin về mọi thứ như tên tác giả, người tải lên (commiter), ngày tải lên, thông tin log...
- **Tags:**
  - Đánh dấu cho dễ đọc. Thay vì một cái tên dài như là 9da581d910c9c4ac93557ca4859e767f5caf5169, chúng ta có thể tên tag là Ver-1.0- Alpha. Dễ nhớ và dễ sử dụng hơn.
- **Index**
  - Index là file nhị phân động và tạm thời miêu tả cấu trúc thư mục của toàn bộ Repo và trạng thái của dự án được thể hiện thông qua commit và tree tại một thời điểm nào đó trong lịch sử phát triển. Git là một hệ thống truy tìm nội dung (content tracking system).

- Index không chứa nội dung file mà chỉ dùng để truy tìm (track) những thứ mà bạn muốn commit.

4. Cơ chế hoạt động của Git

- **Snapshot, not differences**: Git lưu cái gì để keep track of changes
  - Git lưu data thành các snapshot. Cái nào thay đổi thì trỏ vào snapshot mới nhất
  - VD: A→ B → C. B thay đổi thành B1, B giữ nguyên, A trỏ vào B1 mới. A→B1→C
- **Everything is a hash:**
  - Dùng bảng băm SHA-1 để định danh các object, changes
  - Commit: meta data
  - Blob object: object tại thời điểm đang sử dụng



5. Các lệnh git hay thường được sử dụng

Git init	Tạo ra 1 nhánh master. Có thể đổi tên nhánh này thành main hoặc stable
Git clone [link repo]	Clone 1 repo vào địa chỉ mới
Git add	Thêm các thay đổi vào index trước khi commit
Git branch [Tên nhánh]	Tạo nhánh mới
Git checkout [Tên nhánh]	Quay lại nhánh nào đó

Git commit	Tạo một commit mới, thường kèm một message (-m)
Git stash	Tạm thời lưu trữ các thay đổi chưa commit
Git cherry-pick	Lấy sự thay đổi của nhánh này để dùng cho nhánh khác
Git merge	Nhập các nhánh vào với nhau
Git fetch	Nạp các commit từ remote repository
Git pull	= fetch + merge, nạp commit từ kho chứa từ xa và hợp nhất chúng
Git push [nhánh đích] [nhánh gửi]	Tải lên các thay đổi vào nhánh từ xa được chỉ định và cập nhật nhánh đó
Git rebase [Tên nhánh được gộp]	Gộp nhánh hiện tại sang nhánh được chỉ định, thường conflict nên dùng trên local là chủ yếu
Git revert	Quay lại commit trước đó
Git log	Trả về thông tin của các commit đã được tạo (giờ tạo, thông tin nhánh,...)
Git status	Hiển thị tình trạng của directory hiện tại, thay đổi nào đã được keep track,...

6. Git conventions

Đặt tên nhánh

Code Flow Branches

Những nhánh cố định dùng để theo dõi sự thay đổi của mã nguồn từ lúc phát triển đến lúc đi vào hoạt động, triển khai

- **Development** (`dev`)

  Tất cả các nhánh feature và bug fixes sẽ nằm ở nhánh này.

- **QA/Test** (`test`)

  Quản lý tất cả mã nguồn cho việc kiểm thử

- **Staging** (`staging`, Optional)

  Chứa các demo/test feature, chưa được đi vào phát triển và cần được xem xét có nên phát triển không.

- **Master** (`master`)

  The production branch, if the repository is published, this is the default branch being presented.

Trừ Hotfixes, mã nguồn sẽ được merge theo thứ tự development > test > staging > production.

Temporary Branches

Các nhánh được tạo ra và xóa đi tùy theo nhu cầu người phát triển. Nên viết thường và các từ cách nhau bằng dấu gạch ngang (-). ID và mô tả thì nên cách nhau bằng dấu (_)

- **Feature**

  Chứa các thay đổi của mã nguồn khi thêm một tính năng, use case, module.

  Examples:

  - `feature/integrate-swagger`

- **Bug Fix**

  Nếu trong quá trình phát triển, nhánh feature sau khi được push lên github nhưng bị từ chối, thì sẽ lưu lại các sửa chữa cần thiết trên nhánh bugfix.

  Examples:

  - `bugfix/more-gray-shades`
  - `bugfix/JIRA-1444_gray-on-blur-fix`

- **Hot Fix**

  Sửa chữa nhanh, gấp. Có thể được nhập ngay vào nhánh production.

  Examples:

  - `hotfix/increase-scaling-threshold`

- **Experimental**

  Những nhánh thử nghiệm hoặc thêm các idea không có trong sprint.

  Examples:

  - `experimental/dark-theme-support`

- **Build**

  Một nhánh đặc biệt để tạo build artifacts cụ thể hoặc để thực hiện các lần chạy mã.

  Examples:

  - `build/jacoco-metric`

- **Release**

  Nhánh để đánh dấu version của hệ thống

  Examples:

  - `release/myapp-1.01.123`
- **Merging**

  Nhánh để giải quyết conflict khi merge nhánh

  Examples:

  - `merge/combined-device-support`

Commit messages

- Cấu trúc

`<type>(<optional scope>):<description>`

`<optional body>`

`<optional footer>`

- Initial Commit: `init`

Types

- `feat` thêm hoặc xóa một feature
- `fix` sửa lỗi
- `refactor` tái cấu trúc mã nguồn nhưng không làm thay đổi tính năng của hệ thống
- `style` những commit không làm thay đổi mục tiêu của hệ thống (xóa bớt khoảng trắng, format, comment, thiếu dấu chấm phẩy)
- `test` thay đổi liên quan đến test, thêm test, xóa test
- `docs` chỉnh sửa văn bản (README)
- `build` thay đổi đến các build component: tool, ci pipeline, dependencies, project version, ...
- `ops` thay đổi đến các operational components như infrastructure, deployment, backup, recovery,...

- `chore` các commit khác. ví dụ như chỉnh sửa `.gitignore`

Scopes

`scope` cung cấp thêm ngữ cảnh của commit. **Không bắt buộc**

Breaking Changes Indicator

Các thay đổi lớn nên được đánh dấu bởi dấu `!` trước `:`. **Không bắt buộc** có

VD: `feat(api)!: remove status endpoint`

Description

`description`: tóm tắt mô tả thay đổi trong commit, giả sử viết bằng Tiếng Anh, là phần **bắt buộc** có

- Dùng thì hiện tại: "change"
  - `This commit will...` , `This commit should...`
- Không viết hoa chữ cái đầu dòng
- Không dùng dấu chấm (.) để kết thúc dòng

Body

The `body`: Có thể nêu một số lí do commit, lí do phải thay đổi mã nguồn (lỗi,...), thay đổi so với trước, **không bắt buộc**

Footer

The `footer`: Chứa **Breaking Changes** và các reference liên quan đến vấn đề mà commit gặp phải. **Không bắt buộc**

- **Breaking Changes** bắt đầu bằng `BREAKING CHANGES: [...]`

Ví dụ:

- `fix(api): handle empty message in request body`
- `fix(api): fix wrong calculation of request body checksum`

- `fix: add missing parameter to service call`
  `The error occurred because of <reasons>.`
- `feat!: remove ticket list endpoint`
  `refers to JIRA-1337`
  `BREAKING CHANGES: ticket enpoints no longer supports list all entites.`

Nguồn tham khảo:

https://www.simplilearn.com/tutorials/devops-tutorial/version-control

https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

https://vi.wikipedia.org/wiki/Git_(phần_mềm)

https://dev.to/couchcamote/git-branching-name-convention-cch

https://gist.github.com/qoomon/5dfcdf8eec66a051ecd85625518cfd13