

## 1. Benefits of Version Control Systems?

- Những Lợi ích của Version Control - VCS?

- Version control systems là một danh mục các công cụ phần mềm giúp quản lý các thay đổi đối với mã nguồn theo thời gian.
- Version control software theo dõi mọi sửa đổi đối với mã trong một loại cơ sở dữ liệu đặc biệt.
- Nếu khi xảy ra lỗi, lập trình viên có thể quay lại và so sánh các phiên bản mã trước đó để giúp sửa lỗi trong khi giảm thiểu sự gián đoạn cho tất cả các thành viên trong nhóm.
- Đối với hầu hết tất cả các dự án phần mềm, mã nguồn giống như một tài sản quý giá có giá trị phải được bảo vệ. Đối với hầu hết các nhóm dự án, mã nguồn là kho lưu trữ kiến thức và hiểu biết vô giá mà các nhà phát triển đã thu thập và nỗ lực tinh chỉnh cho chuẩn.
- Hệ thống kiểm soát phiên bản - Version Control Systems (VCS) đã có nhiều cải tiến lớn trong vài thập kỷ qua.
- VCS đôi khi được gọi là các công cụ SCM (Quản lý mã nguồn) hoặc RCS (Hệ thống kiểm soát sửa đổi).
- Một trong những công cụ VCS phổ biến nhất được sử dụng hiện nay được gọi là Git. Git là một VCS phân tán, một thể loại được gọi là **Distributed Version Control system - DVCS**.
- Git là nguồn mở và miễn phí.

- Tính năng của VCS - Features of VCS?

- Bảo trì các branches độc lập để các thành viên trong nhóm theo dõi các thay đổi tương ứng của họ.
- Dễ so sánh mã giữa các branches khác nhau.
- Nhập mã từ một số branches của nhiều thành viên trong nhóm.
- Thuận tiện theo dõi các thay đổi trong mã để tìm phiên bản.
- Chú thích thay đổi với tên của tác giả và tin nhắn trong một phiên bản mã cụ thể. Điều này giúp dễ dàng xác định ý định thay đổi.
- So sánh đơn giản giữa các phiên bản để giúp giải quyết xung đột trong mã trong quá trình hợp nhất (merging).
- Hoàn nguyên (Revert) các thay đổi được thực hiện cho bất kỳ trạng thái nào từ lịch sử của nó.

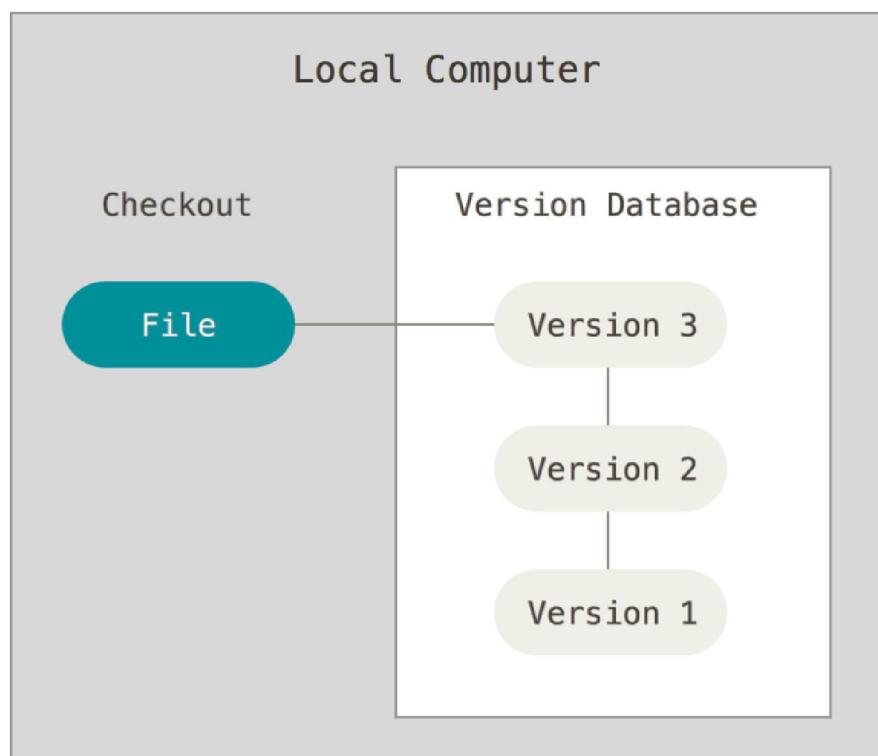
- **Hệ thống kiểm soát phiên bản hàng đầu - Top Version Control Systems?**

- Dưới đây là các hệ thống kiểm soát phiên bản hàng đầu được nhiều người sử dụng:
  - **GIT**
  - CVS
  - SVN
  - Assembla
  - Mercurial
  - Bazaar

## 2. Local, Central and Distributed Version Control Systems?

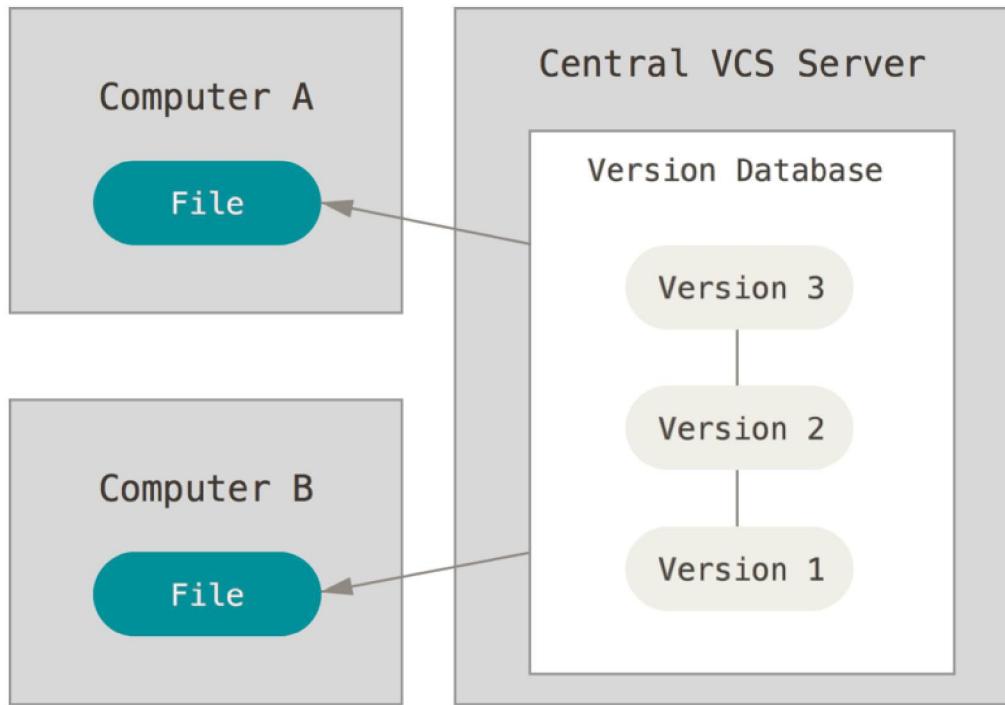
- **Local VCS?**

- Là hệ thống quản lý phiên bản sơ khai nhất.
- Local VCS có một nhược điểm lớn là chỉ hữu ích cho một nhà phát triển duy nhất tại một thời điểm.
- Do đó, một số nhà phát triển làm việc trên các hệ thống khác nhau trong một dự án không thể cộng tác hiệu quả bằng cách sử dụng Local VCS.



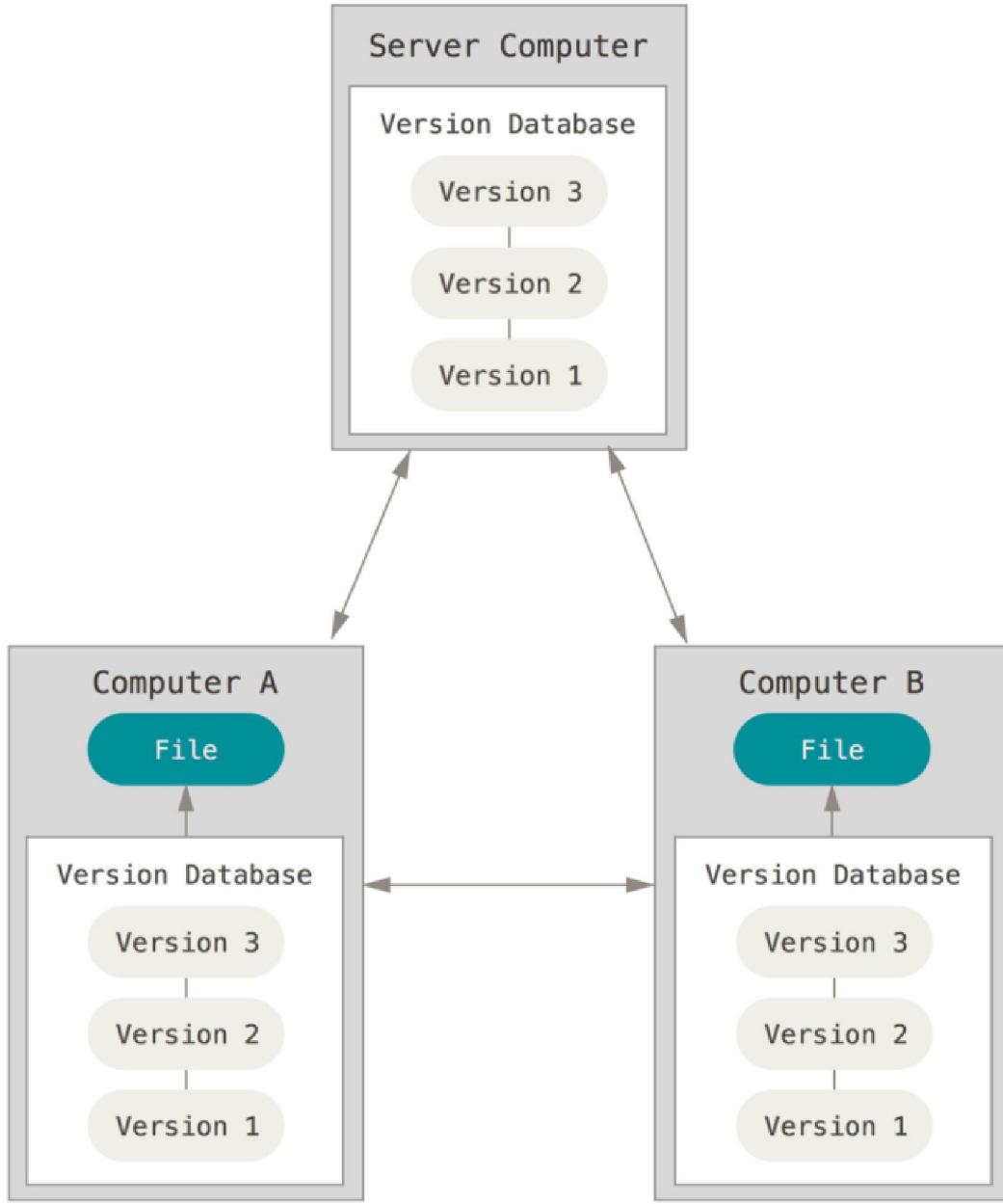
- Central VCS?

- Sử dụng Central VCS, nhà phát triển có thể lưu trữ phiên bản ở một vị trí duy nhất tại máy chủ trung tâm (central server).
- Mặc dù Central VCS đã giúp các nhà phát triển nhưng các nhóm dự án vẫn đối diện với nguy cơ ngừng hoạt động của Central Sever.
- Các thành viên trong nhóm sẽ không thể hợp tác cũng như ghi lại những thay đổi của bất cứ điều gì họ làm việc do sự cố ngừng hoạt động như vậy.
- Những máy chủ như vậy cũng dễ bị hack.
- Trong trường hợp như vậy, nó sẽ mất nhiều thời gian để phục hồi công việc.
- Do đó, nó không nên lưu trữ tất cả các thay đổi đối với mã ở một nơi duy nhất.
- Ví dụ: SVN là tiêu biểu của Central VCS.



- Distributed VCS?

- Trong Distributed VCS, mọi máy cục bộ sẽ có bản sao của toàn bộ cơ sở mã (còn gọi là kho lưu trữ) cùng với lịch sử của nó.
- Git, mà chúng ta sẽ học trong loạt bài tiếp theo là một hệ thống Kiểm soát Phiên bản Phân tán (Distributed Version Control system).



### 3. What is GIT?

- Git là một Hệ thống kiểm soát phiên bản phân tán mã nguồn mở (Open Source Distributed Version Control System).
- Git được phát triển bởi Linus Torvalds, cũng là cha đẻ của Hệ điều hành Linux.
- Năm 2002 Linus có thực hiện các dự án với việc sử dụng 1 DVCs có tên gọi là "BitKeeper" có trả phí.

- Do Linux là nguồn mở và nhiều nhà phát triển muốn đóng góp cho Linux nên từ 05 Linus Torvalds đã cho ra đời 1 loại VCS có tên gọi là Git cho cộng đồng phát triển Linux.
- **Git là một hệ thống điều khiển - Git is a Control System:**
  - Điều này về cơ bản có nghĩa là Git là một trình theo dõi nội dung.
  - Vì vậy, Git có thể được sử dụng để lưu trữ nội dung và nó chủ yếu được sử dụng để lưu trữ mã.
  - Nó có nhiều tính năng khác, do đó nó khá phổ biến trong cộng đồng phát triển.
- **Git là một hệ thống kiểm soát phiên bản - Git is a Version Control System:**
  - Git là một hệ thống kiểm soát phiên bản mã được lưu trữ trong Git tiếp tục thay đổi khi thêm mã.
  - Ngoài ra, nhiều nhà phát triển có thể thêm mã song song.
  - Vì vậy, Hệ thống kiểm soát phiên bản giúp xử lý việc này bằng cách duy trì lịch sử về những thay đổi đã xảy ra.
  - Ngoài ra, Git cung cấp các tính năng như rẽ nhánh (branches) và sáp nhập (merges).
- **Git là một hệ thống kiểm soát phiên bản phân tán - Git is a Distributed Version Control System:**
  - Git có một kho lưu trữ từ xa được lưu trữ trong một máy chủ và một kho lưu trữ cục bộ được lưu trữ trong máy tính của mỗi nhà phát triển.
  - Điều này có nghĩa là mã không chỉ được lưu trữ trong một máy chủ trung tâm, mà là bản sao đầy đủ của mã có trong tất cả các máy tính của nhà phát triển.
  - Git là một Hệ thống kiểm soát phiên bản phân tán do mã có trong mọi máy tính của nhà phát triển.

#### 4. Why to use Git?

- **Kể từ khi phát triển và phát hành Git, nó đã trở nên phổ biến rộng rãi trong số các nhà phát triển và là nguồn mở đã kết hợp nhiều tính năng.**
- **Ngày nay, một số lượng lớn các dự án sử dụng Git để kiểm soát phiên bản, cả thương mại và cá nhân.**
- **Các tiêu chí đặt ra dành cho Git như:**
  - Tốc độ - Speed.
  - Thiết kế đơn giản - Simple design.

- Hỗ trợ mạnh với quá trình phát triển phi tuyến tính non-linear development (hỗ trợ hàng nghìn nhánh khác nhau).
- Bản sao toàn bộ phân tán - Fully distributed.
- Khả năng tương thích với nhiều hệ thống lớn với Linux Kernel.

## 5. Git Basics?

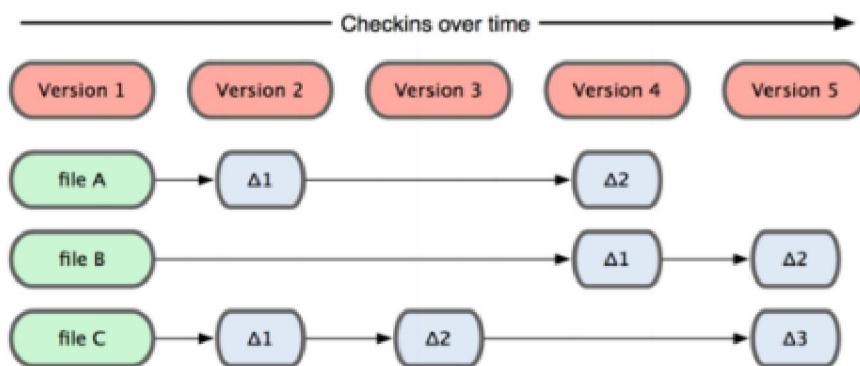
### Về cơ bản Git có 5 tính chất sau:

- Snapshots, Not Differences
- Nearly Every Operation Is Local
- Git Has Integrity
- Git Generally Only Adds Data
- The Three States

### a - Snapshots, Not Differences?

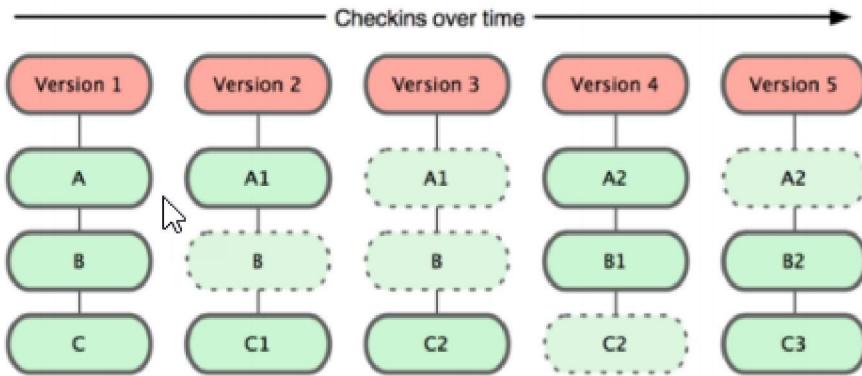
- Phần lớn các VCS khác lưu trữ danh sách các tệp file thay đổi.
- Như vậy thì mỗi phiên bản sẽ chỉ lưu giữ các tệp file có sự thay đổi mà thôi.
- Đây chính là cách quản lý của các VCS khác.

## Subversion



- Còn với Git thì mỗi phiên bản sẽ là 1 SnapShots đầy đủ.
- Các tệp file ko có sự thay đổi sẽ có liên kết trả đến tệp file thay đổi gần nhất.

# Git



## b - Nearly Every Operation Is Local?

- Tất cả các thao tác đều làm việc trên Local là tính chất thứ 2 của Git.
- Ít nhất các thao tác phải làm việc với Server.
- Có thể làm việc bình thường khi không có kết nối mạng với Server.
- Ví dụ Commit file, View History, ... đều hoàn toàn ko cần internet vẫn làm việc bình thường.

## c - Git Has Integrity?

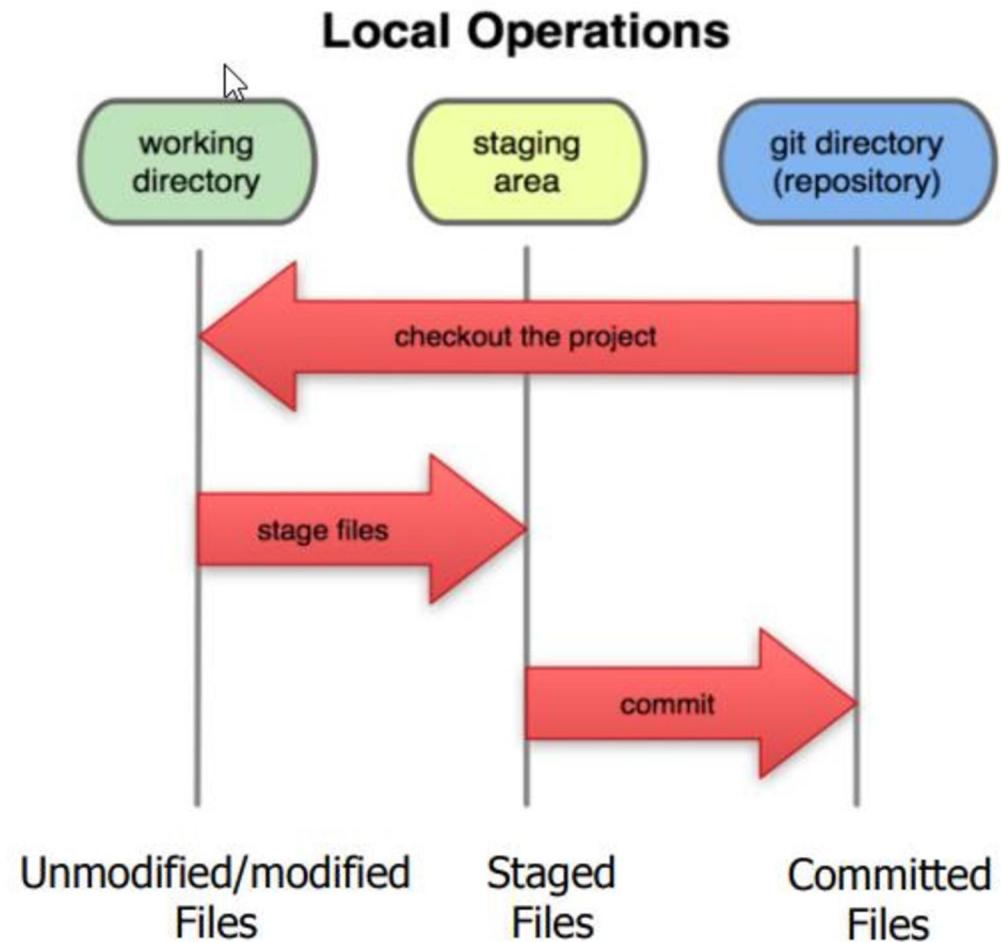
- Tính toàn vẹn trong Git là tính chất cũng rất quan trọng.
- Tất cả các dữ liệu trong Git đều được check-summed trước khi lưu trữ.
- Git sử dụng cơ chế checksumming gọi là SHA-1.
- Đây là 1 chuỗi 40 ký tự kiểu Hexadecimal Characters (0-9 và a-f) và được tính toán dự trên dữ liệu nội dung của tệp hoặc cấu trúc thư mục trên Git.
- Như vậy khi quá trình truyền nhận file bị lỗi "corruption" thì Git sẽ dự vào checksum để detect dữ liệu đó.

## d - Git Generally Only Adds Data?

- Khi thực hiện bất cứ thao tác nào trên Git thì các dữ liệu đó đều được đưa vào Git Database.
- Sau khi commit 1 snapshot vào Git, thì rất khó để mất dữ liệu.
- Đây là thể hiện tính bền vững trong Git.

## e - The Three States?

- Đây là tính chất khá quan trọng trong Git.
- Đó là khi làm việc với dữ liệu luôn có 03 trạng thái: **committed, modified, staged**.



## 6. Installing and learning Git?

### a. Tải các công cụ để làm việc với Git?

- Bạn cần tải và download Git: <https://git-scm.com/downloads>
- Bạn cần tải TortoiseGit: <https://tortoisegit.org/download/>
- Bạn cần tải SourceTree: <https://www.sourcetreeapp.com/>
- Bạn có thể xài thêm Git Kraken: <https://www.gitkraken.com/git-client>
- Nếu bạn cần tải SourceTree phiên bản cũ hơn tại đây:  
<https://www.sourcetreeapp.com/download-archives>

### b. Đăng ký tài khoản trên các Web Apps?

- Đăng ký tài khoản tại trang Bitbucket: <https://bitbucket.org/>

#### **c. Thực hiện tạo SSH Key cho cấu hình tài khoản trên GitLab?**

- Tham khảo tại: <https://help.github.com/> (Generating a new SSH key and adding it to the ssh-agent)
- Các bước tạo SSH Key như sau:
  - Open **Git Bash**.
  - Paste the text below, substituting in your GitHub email address:
    - \$ ssh-keygen -t rsa -b 4096 -C "anv@gmail.com"
  - When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.
    - Enter a file in which to save the key (/c/Users/you/.ssh/id\_rsa):[Press enter]
  - At the prompt, type a secure passphrase. For more information, see "Working with SSH key passphrases".
    - Enter passphrase (empty for no passphrase): [Type a passphrase]
    - Enter same passphrase again: [Type passphrase again]
  - Ensure the ssh-agent is running:
    - \$ eval \$(ssh-agent -s)
  - Add your SSH private key to the ssh-agent. If you created your key with a different name, or if you are adding an existing key that has a different name, replace id\_rsa in the command with the name of your private key file.
    - \$ ssh-add ~/.ssh/id\_rsa
  - Add the SSH key to your GitHub/GitLab account.

#### **d. Cấu hình Notepad++ Editor vào Git?**

- Kiểm tra xem máy tính cấu hình cho Git chuẩn chưa:
  - git config --list <enter>** (xem xong ấn "q" để thoát cửa sổ đó)
- Có thể cấu hình editor để làm việc với Git. Nếu không cấu hình thì Git sử dụng hệ thống editor mặc định.
- Ví dụ cấu hình sử dụng Notepad++:

- x86: git config --global core.editor "'C:\npp.7.6.6.bin.x64\notepad++.exe' -multiInst -nosession"
- x64: git config --global core.editor "'C:\npp.7.6.6.bin.x64\notepad++.exe' -multiInst -nosession"
- multiInst: Có nghĩa là có thể bật nhiều cửa sổ 1 lúc.
- nosession: Không lưu các phiên làm việc trước đó.
- Sau đó sử dụng lệnh: git config core.editor <enter>

#### **e. Qui trình sử dụng Merge Tool với TortoiseGit?**

- Khi xảy ra xung đột conflicts.
- Sử dụng lệnh để gọi công cụ merge tool: **\$ git mergetool --tool=tortoisemerge**
- Choose:
  - Use text block from 'mine' before 'theirs'
  - Use text block from 'theirs' before 'mine'
- Complete of fix:
  - Delete a file \*.orig
  - git status
  - git commit -a -m "fix ok"
  - git push origin <master|...>
- Note:
  - You need 'pull' before the update a file and 'push' into Git Server (Remote Repositories).
  - You need instal TortoiseGit tools for conflit error.

## **7. Getting Started with Git?**

- Trước khi làm việc với Git cần thực hiện cấu hình. Gồm:

- **Your Identity:**
  - Cần cấu hình thông tin về người sử dụng với User Name và Email.
  - \$ git config --global user.name "doannv"
  - \$ git config --global user.email "doannv@imic.edu.vn"

- **Your Editor:**
  - Có thể cấu hình editor để làm việc với Git. Nếu không cấu hình thì Git sử dụng hệ thống editor mặc định.
  - Ví dụ cấu hình sử dụng Notepad++:
    - x86: **git config --global core.editor** "'C:\npp.7.6.6.bin.x64\Notepad++.exe' -multiInst -nosession"
    - x64: **git config --global core.editor** "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst -nosession"
  - multiInst: Có nghĩa là có thể bật nhiều cửa sổ 1 lúc.
  - nosession: Không lưu các phiên làm việc trước đó.
- **Checking your settings:**
  - Sử dụng lệnh git config --list để kiểm tra lại toàn bộ các thiết lập.
  - Sử dụng lệnh git config với 1 key value bất kỳ để hiển thị ra từng thông tin có trong danh sách.
  - Ví dụ:
    - \$ git config user.name <enter>
    - \$ git config core.editor <enter>
- **Getting Helps:**
  - Khi cần tìm hiểu các cú pháp hoặc các lệnh làm việc với git chúng ta có thể sử dụng git help như sau:
    - \$ git help <verb>
    - \$ git <verb> --help (ví dụ: git clone --help)

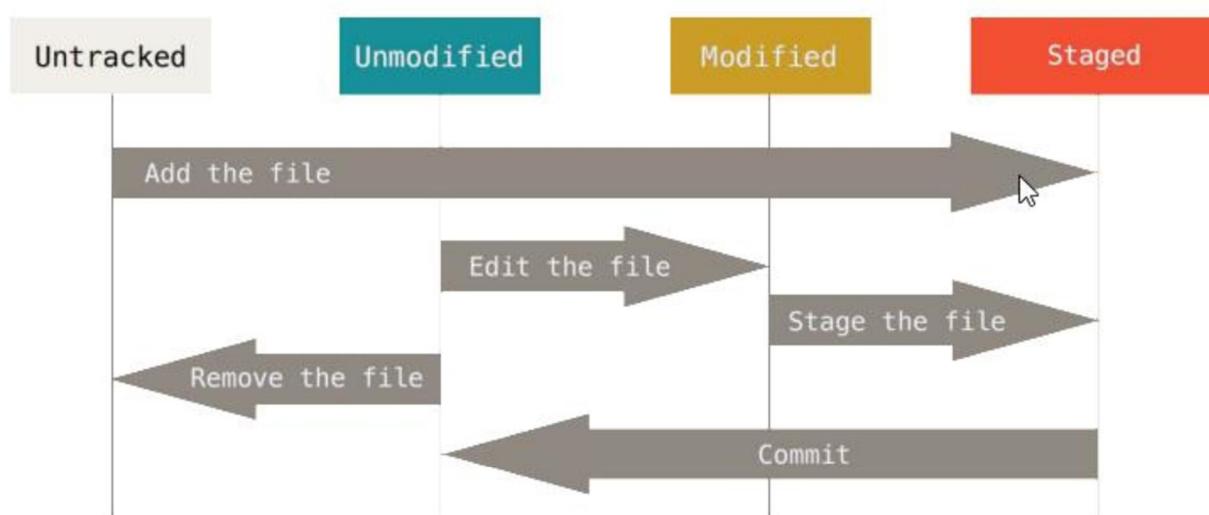
## 8. Recording Changes to the Repository?

- Phần này chúng ta sẽ cần nắm được các nội dung như sau:

- The lifecycle of the status of your files
- Checking the status of your files
- Tracking new files
- Staging Modified Files
- Short Status

- Ignoring Files
- Viewing Your Staged and Unstaged Changes
- Committing your changes
- Skipping the staging area
- Removing files
- Moving files

### a. The lifecycle of the status of your files?



### b. Checking the status of your files?

- **\$ git status:** Kiểm tra thư mục đang làm việc hiện thời.
- **\$ git status -s hoặc \$ git status --short**
- **\$ git log**
- **\$ git add readme.txt** để chuyển trạng thái tệp về Staged.

### c. Ignoring Files

- Git ignore file có tên là **.gitignore**
- Nó được sử dụng để loại trừ (excluded) tất cả các tệp file and thư mục trong dự án.
- Có thể sử dụng định nghĩa qui tắc trên file nội dung.
- Cần định nghĩa các qui tắc đó trước khi commit.

- Github cung cấp tất cả các gitignore cho các công cụ tại link: <https://github.com/github/gitignore>
- hoặc trang: <https://www.gitignore.io/>
- Để tạo ra 1 tệp gitignore có thể dùng lệnh: **\$ touch .gitignore**

#### **d. Viewing Your Staged and Unstaged Changes**

- Lệnh **\$ git diff --staged** để xem thông tin trạng thái các tệp file đã staged.
- Hoặc sử dụng lệnh **\$ git rm --cached <file>** để chuyển trạng thái tệp về Unmodified.

#### **e. Committing your changes?**

- Khi bạn muốn viết 1 nội dung ghi chú dài thì có thể dùng lệnh **\$ git config --global core.editor** để thiết lập cho 1 editor được sử dụng.
- Khi bạn muốn commit các tệp file thì sử dụng lệnh **\$ git commit -m "chú thích"**.

#### **f. Removing files?**

- **\$ git rm:** Sử dụng để remove 1 tệp file từ Git, có thể sử dụng để xóa nó từ tracked files.
- **\$ git rm --cached readme.txt** (--cached là xóa nhưng nó vẫn còn lưu lại trên ổ cứng)
- **\$ git rm bin/\*.\*txt:** Khi muốn xóa toàn bộ các tệp \*.txt trong thư mục bin.

#### **g. Moving files?**

- Có thể đổi tên file trong Git sử dụng lệnh **\$ git mv <source file> <destination file>**

### **9. Viewing the commit history?**

- Lệnh được sử dụng khá nhiều là **\$ git log**.
- Có thể thêm tùy chọn **-p** khi muốn hiển thị ra sự khác nhau trong mỗi lần commit đó.
- Có thể thêm tùy chọn số **-2** là giới hạn số lần commit gần nhất.
- Lệnh khi đó sẽ là **\$ git log -p -2**
- Ấn **q** (quit) khi muốn thoát cửa sổ hiển thị log.
- Có thể sử dụng tùy chọn **-stat** để xem thông tin chi tiết đầy đủ.
- Lệnh khi đó là: **\$ git log -stat**
- Có thể sử dụng tùy chọn **--pretty** để thay đổi log output formats.

- Lệnh sẽ là: `$ git log --pretty=oneline`
- `$ git log --pretty=format:"%h - %an, %ar:%s"`

*Table 1. Useful options for `git log --pretty=format`*

| Option           | Description of Output   |
|------------------|---|
| <code>%H</code>  | Commit hash   |
| <code>%h</code>  | Abbreviated commit hash                                       |
| <code>%T</code>  | Tree hash   |
| <code>%t</code>  | Abbreviated tree hash   |
| <code>%P</code>  | Parent hashes   |
| <code>%p</code>  | Abbreviated parent hashes                                     |
| <code>%an</code> | Author name   |
| <code>%ae</code> | Author email  |
| <code>%ad</code> | Author date (format respects the <code>--date=option</code> ) |
| <code>%ar</code> | Author date, relative   |
| <code>%cn</code> | Committer name  |
| <code>%ce</code> | Committer email   |
| <code>%cd</code> | Committer date  |
| <code>%cr</code> | Committer date, relative                                      |
| <code>%s</code>  | Subject   |

- `$ git log --since=2.weeks`
- `$ git log --pretty="%h - %s" --author=gitster --since="2018-06-08"\`

*Table 3. Options to limit the output of `git log`*

| Option                         | Description  |
|--------------------------------|--|
| <code>-&lt;n&gt;</code>        | Show only the last n commits   |
| <code>--since, --after</code>  | Limit the commits to those made after the specified date.                    |
| <code>--until, --before</code> | Limit the commits to those made before the specified date.                   |
| <code>--author</code>          | Only show commits in which the author entry matches the specified string.    |
| <code>--committer</code>       | Only show commits in which the committer entry matches the specified string. |
| <code>--grep</code>            | Only show commits with a commit message containing the string                |
| <code>-S</code>                | Only show commits adding or removing code matching the string                |

## 10. Undo things - Unstaging a staged file - Unmodifying a Modified file?

- [Undo things?](#)

- Khi muốn thực hiện commit lại mà không muốn git ghi log commit nhiều lần có thể thêm tùy chọn -amend
- Lệnh sẽ là: \$ git commit --amend
- Ví dụ:
- \$ git commit -m "Initial committed"
- \$ git add forgotten\_file
- \$ git commit --amend

- [Unstaging a staged file?](#)

- Khi muốn chuyển 1 tệp file từ Staged thành Unstaging thì lệnh sẽ là **\$ git reset HEAD <file\_name>**
- Ví dụ: **\$ git reset HEAD abc.txt**
- Như vậy tệp abc.txt đã được chuyển trạng thái thành Unstaging (unmodified).

- [Unmodifying a Modified file?](#)

- Khi muốn chuyển 1 tệp file từ Modified thành Unmodifying thì sử dụng lệnh \$ git checkout -- <tên file>.
- Ví dụ: **\$ git checkout -- abcd.txt**

## 11. Working with Remotes - Github/GitLab?

- Phần này chúng ta cần nắm được 1 số nội dung như sau:
  - Showing your remotes
  - Adding remote repositories
  - Fetching and pulling from your remotes
  - Pushing to your remotes
  - Inspecting a Remote
  - Removing and renaming remotes
- [Showing your remotes?](#)

- Sử dụng lệnh **\$ git clone <url>** để lấy về 1 snapshot từ trên remote về máy (local repositories).
- Sử dụng lệnh **\$ git remote -v** khi muốn hiển thị URLs git lưu trữ với shortname để sử dụng khi đọc và ghi dữ liệu lên remote.
- Mặc định shortname đó là origin, chúng ta có thể đổi tên nếu muốn.
- **Adding remote repositories?**
  - Khi muốn gắn 1 tên cho 1 liên kết nào đó của Remote thì sử dụng lệnh dưới đây:
  - Sử dụng lệnh **\$ git remote add <shortname> <url>**
  - Ví dụ: `$ git remote add imic https://github.com/imic.technology/git.git`

## 12. Tagging in Git?

- Phần này cần nắm được một số nội dung sau:
  - Listing your tags
  - Creating tags
  - Annotated tags
  - Lightweight tags
  - Tagging later
  - Sharing tags
  - Checking out tags
  - Delete tag

### a. Listing your tags?

- Hầu hết các VCS đều hỗ trợ tính năng này.
- Câu lệnh **\$ git tag** để liệt kê ra danh sách các tag trong history.
- Có thể tìm kiếm tag theo 1 pattern. Ví dụ: Muốn tìm kiếm ra các phiên bản với mẫu bắt đầu từ 1.8.5...
  - (dấu \* sẽ là các mã chưa biết sẽ giúp tìm kiếm rộng hơn)
  - Lệnh sẽ là: **\$ git tag -l "v1.8.5\*"**

### b. Creating tags?

- Có 2 loại tag chính là:
- LightWeight tags (nó là 1 pointer đến 1 specific commit, chứa ít thông tin)
- Annotated tags (lưu đầy đủ thông tin full objects trong Git Database, thông thường lựa chọn kiểu này)

#### c. Annotated tags?

- Câu lệnh để tạo mới 1 Annotated tag trong Git cũng khá đơn giản.
- **\$ git tag -a v1.0 -m "my version 1.0"**
- **\$ git tag**
- Khi muốn xem chi tiết 1 phiên bản nào đó thì sử dụng lệnh **\$ git show v1.0**

#### d. Lightweight tags?

- Sử dụng lệnh sau để tạo ra 1 Lightweight tags.
- **\$ git tag v1.0-lw**
- **\$ git tag**

#### e. Tagging later?

- Để tạo ra tags cho lần commit nào đó dựa vào commit id thông qua các bước sau:
- - Bước 01: Sử dụng lệnh git log để hiển thị thông tin
- **\$ git log --pretty=oneline**
- - Bước 02: Copy mã commit id để sử dụng cho việc tạo ra tags cho lần commit đó.
- **\$ git tag -a v1.2 <commit id>**

#### f. Sharing tags?

- Mặc định tags chỉ thể hiện dưới Local Repositories.
- Do vậy muốn chia sẻ tags thì sử dụng các lệnh dưới đây:
- **\$ git push origin [tagname]**
- Ví dụ: **\$ git push origin v1.2** (có nghĩa là chỉ đưa tags 1.2 lên Remote)
- Khi muốn đưa tất cả các tags được định nghĩa dưới Local lên Remote thì sử dụng lệnh:

- **\$ git push origin --tags**

#### g. Checking out tags?

- Khi muốn lấy về 1 phiên bản nào đó, hoặc muốn tách từ 1 phiên bản nào đó ra thành 1 nhánh mới thì sử dụng lệnh sau:
- **\$ git checkout -b <tên nhánh mới> <từ phiên bản nào>**
- Ví dụ: **\$ git checkout -b newversion v1.0**
- Trong đó:
- newversion: là nhánh mới tạo ra.
- v1.0: là tag đã tạo ra trước đó.

#### h. Delete tag?

- Để xóa tags ở Local: **\$ git tag -d {tag\_name}**
- Để xóa tags ở Remote: **\$ git push {remote\_name} --delete {tag\_name}**
- Có thể sử dụng lệnh **\$ git ls-remote --tags** để hiển thị ra danh sách các tags trên Remote.
- Ví dụ: **\$ git push origin --delete v1.0**

### 13. Git Alias

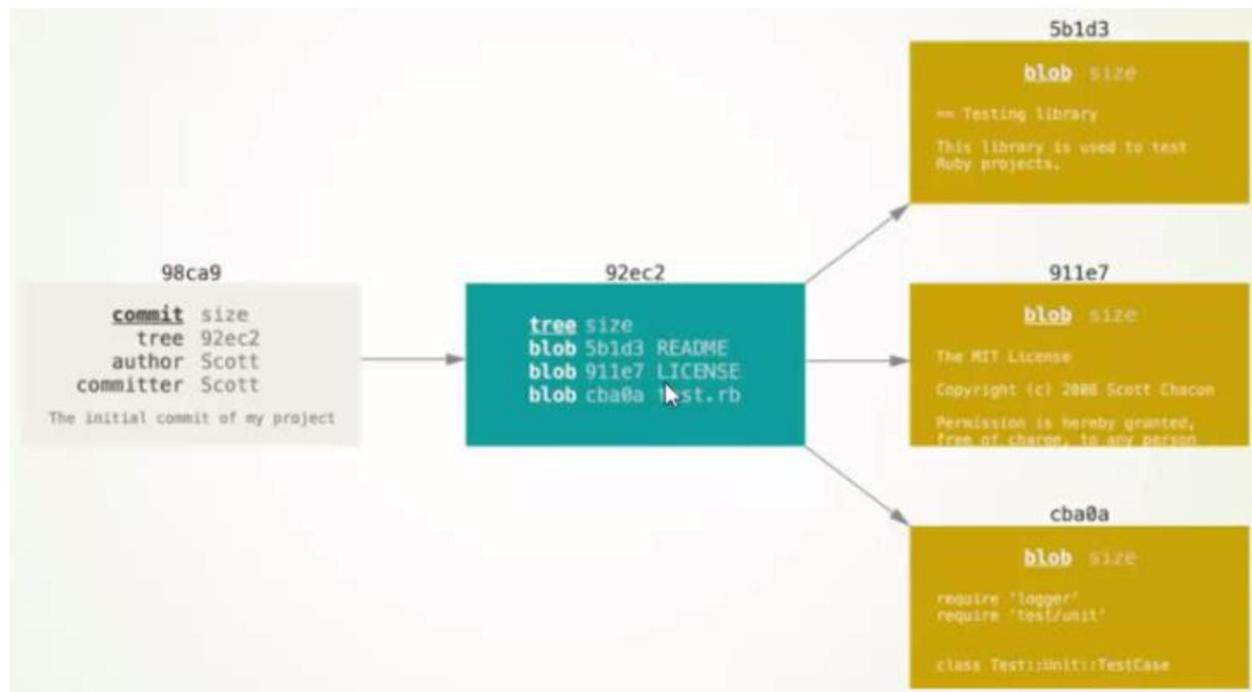
- Khi chúng ta muốn viết câu lệnh nhanh hơn thì ta nên xài tính năng này.
- Cú pháp thực hiện:
  - **\$ git config --global alias.co checkout** (*co: là bí danh cho checkout*)
  - **\$ git config --global alias.br branch** (*br: là bí danh cho branch*)
  - **\$ git config --global alias.ci commit** (*ci: là bí danh cho commit*)
  - **\$ git config --global alias.st status** (*st: là bí danh cho status*)
- Khi sử dụng chúng ta chỉ cần gõ: **\$ git st <enter>** sẽ tương đương lệnh **\$ git status <enter>**
- Có thể nối chuỗi lệnh vào alias bằng cách như sau:
  - Ví dụ khi chúng ta muốn chuyển 1 file về trạng thái unstaged sử dụng thêm reset HEAD --
    - Sử dụng gán bí danh như sau: **\$ git config --global alias unstage 'reset HEAD --'**

- Thay vì phải gõ:
- \$ git unstage fileA
- \$ git reset HEAD --fileA
- Để test thử thì chúng ta cần thử thay đổi 1 file (test.txt) để trạng thái sẽ tự động là stage.
- Giờ chúng ta muốn hủy bỏ sự thay đổi chỉ cần gõ: **\$ git unstage test.txt**
- Khi chúng ta muốn xem log lần commit cuối cùng: **\$ git log -1 HEAD**
- Chúng ta sẽ đặt bí danh: **\$ git config --global alias.last 'log -1 HEAD'**
- Khi muốn xài chỉ cần: **\$ git last**

## 14. Git Branching

- Lợi ích của Git là cho phép chúng ta dễ dàng có thể tạo ra các nhánh để tránh xung đột tài nguyên hoặc ảnh hưởng đến tài nguyên gốc.
- Ví dụ các tính năng phát triển thêm chúng ta có thể tách ra thành các nhánh để không ảnh hưởng đến phiên bản hiện tại.

### a - A commit and its tree?



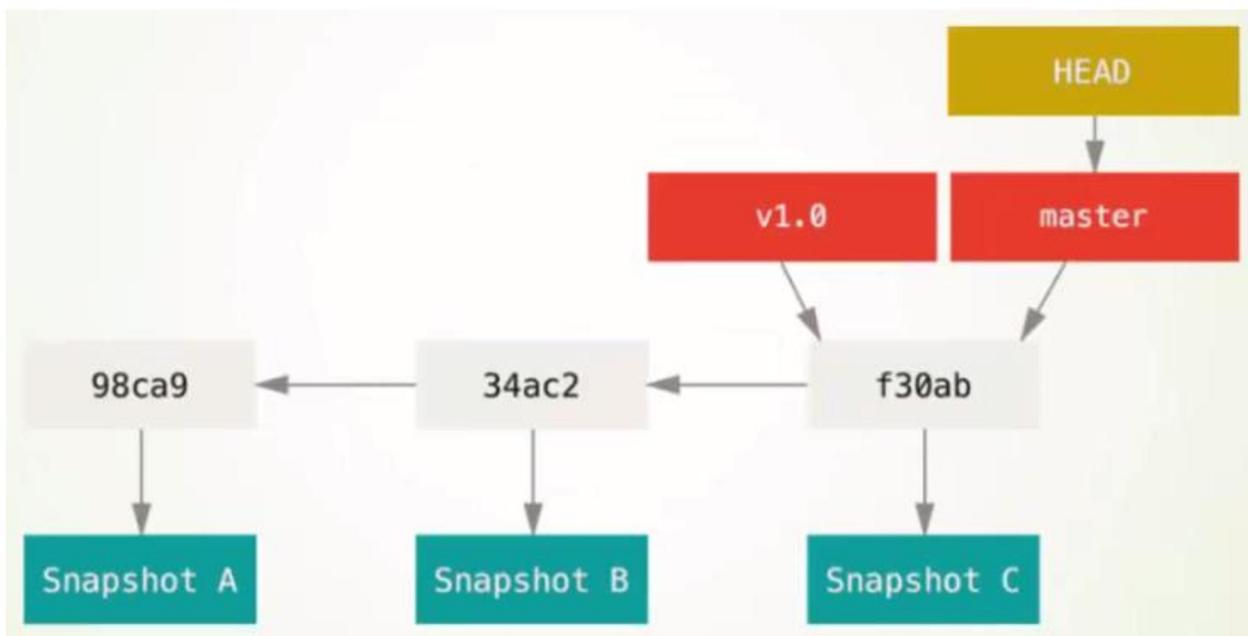
- Trong 1 tree có thể lưu nhiều mã băm Hash, mỗi mã băm đại diện cho 1 file có sự thay đổi (đối với file

- Không có sự thay đổi thì sẽ reference đến mã băm trước đó).

### b - Commits and their parents?



### c - A branch and its commit history?



- HEAD:** là 1 con trỏ (pointer) để trỏ đến nhánh hiện tại đang làm việc (nhờ có con trỏ này mà sẽ giúp)
- Chúng ta nhận biết nhánh đang làm việc 1 cách dễ dàng).

### d - Creating a new branch?

- Để tạo mới nhánh bằng lệnh: **\$ git branch testing**