PHENIKAA UNIVERSITY

Faculty of Computer Science

Nhập Môn Công Nghệ Thông Tin

Bài 5: Kiểm soát phiên bản với Git

Nội dung

PHENIKAA UNIVERSITY
Faculty of Computer Science

- Mục tiêu
- Kiểm soát phiên bản-các vấn đề
- Git-GitHub
- Thực hành

Mục tiêu

PHENIKAA UNIVERSITY
Faculty of Computer Science

- Hoàn thành bài học này sinh viên sẽ có khả năng
  - Hiểu về quản lý phiên bản
  - Hiểu, sử dụng Git, GitHub
    - Sử dụng git locally
    - Sử dụng git with remote

Kiểm soát phiên bản khi làm việc một mình

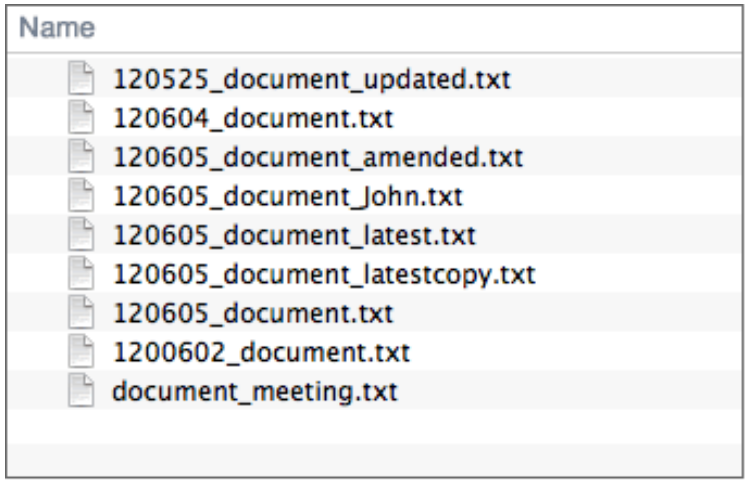
PHENIKAA UNIVERSITY
Faculty of Computer Science

- Vấn đề gặp phải:
  - Đã có mã hoạt động, thực hiện một loạt thay đổi và lưu lại dẫn đến hỏng mã…
  - Vô tình xóa một tập tin quan trọng, ổ cứng hỏng …=> mất dữ liệu
  - Vô tình làm hỏng cấu trúc/nội dung của dự án

  => **Muốn quay trở lại trạng thái trước khi chỉnh sửa của file/dự án**

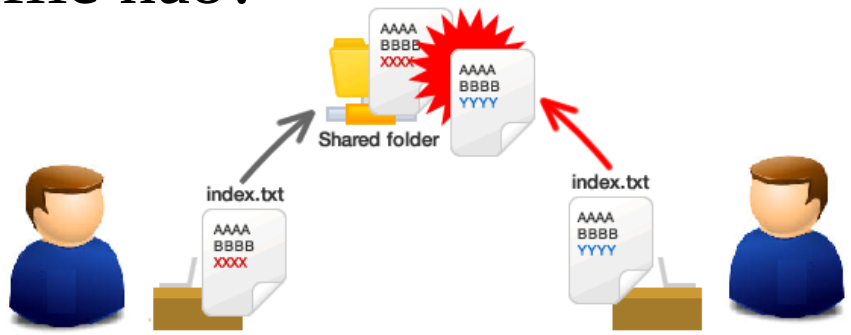
- Giải pháp: Sao chép file trước khi chỉnh sửa
  - Sử dụng ngày thay đổi vào tên thư mục hay file
  - Đặt tên file gợi nhớ: final, final-final, file_chuẩn, ….

  => dễ nhầm lẫn, khó phân biệt được phiển bản mới nhất



Kiểm soát phiên bản khi làm việc nhóm

- Ai là người lưu trữ bản "chính thức" của dự án?
- Mọi người có thể đọc/ghi những thay đổi của nhau không?
- Làm thế nào để một người chỉnh sửa và thống nhất được cho tất cả mọi người?
- Điều gì xảy ra nếu hai người cùng cố gắng chỉnh sửa cùng một tệp?
- Điều gì sẽ xảy ra nếu một người mắc lỗi và làm hỏng một tập tin quan trọng?
- Làm sao để biết được ai đang làm việc với mã/file nào?



Kiểm soát phiên bản- Giải pháp

- Hệ thống kiểm soát phiên bản: Phần mềm theo dõi và quản lý những thay đổi đối với một tập hợp các tệp và tài nguyên theo giời gian.
- Kiểm soát phiên bản mọi lúc
  - Được tích hợp vào trình xử lý văn bản/bảng tính/phần mềm trình bày
  - Cho phép "hoàn tác" trở lại "phiên bản trước đó"
- Wiki's
  - Cho phép người dùng dễ dàng tạo, chỉnh sửa và chia sẻ nội dung với nhau, thường được dùng để tạo ra cơ sở dữ liệu thông tin mà ai cũng có thể đóng góp và cập nhật

Phầm mềm kiểm soát phiên bản

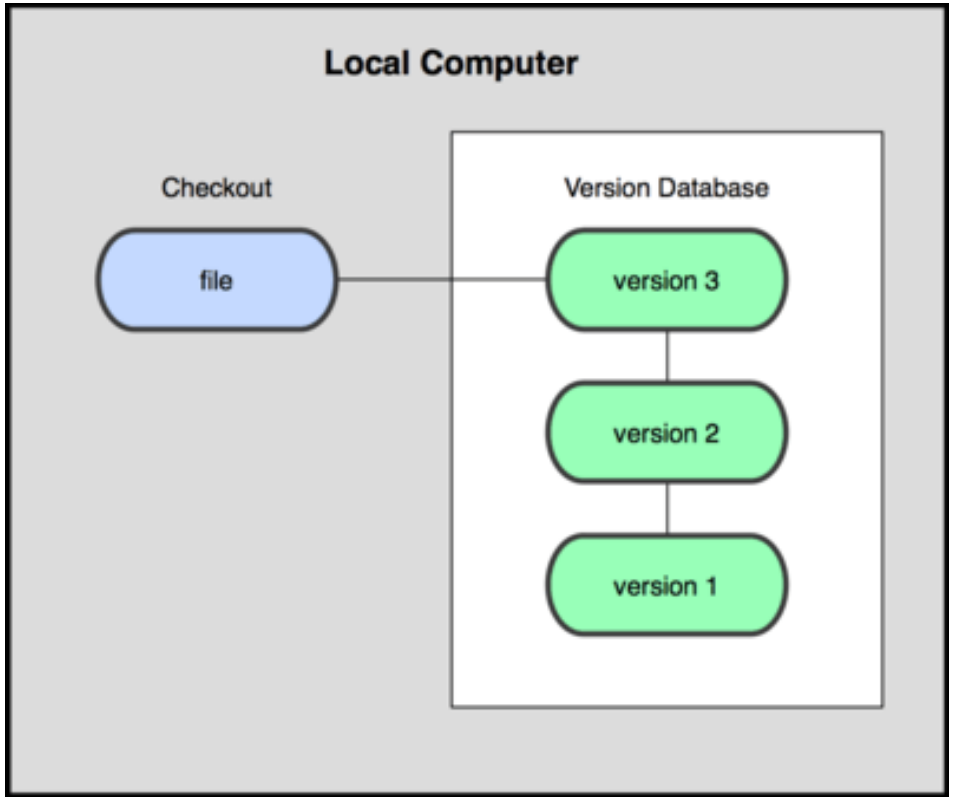
- Nhiều hệ thống kiểm soát phiên bản được thiết kế và sử dụng đặc biệt cho các dự án kỹ thuật phần mềm
  - ví dụ: CVS, Subversion (SVN), Git, BitKeeper, Perforce

Syncro
SVN Client

- Giúp các nhóm làm việc cùng nhau trong các dự án mã hóa
  - Chia sẻ bản sao của các tập tin cho phép tất cả người dùng có thể truy cập
  - Giữ các phiên bản hiện tại của tất cả các tập tin và sao lưu các phiên bản trước đó
  - Xem lại được những thay đổi đã được thực hiện theo thời gian
  - Quản lý xung đột khi nhiều người dùng sửa đổi cùng một tệp
  - Có thể sử dụng cho nhiều loại dữ liệu khác nhau: văn bản, ảnh, v.v.
    - Nhưng thường hoạt động tốt nhất với các tệp văn bản/mã thuần túy

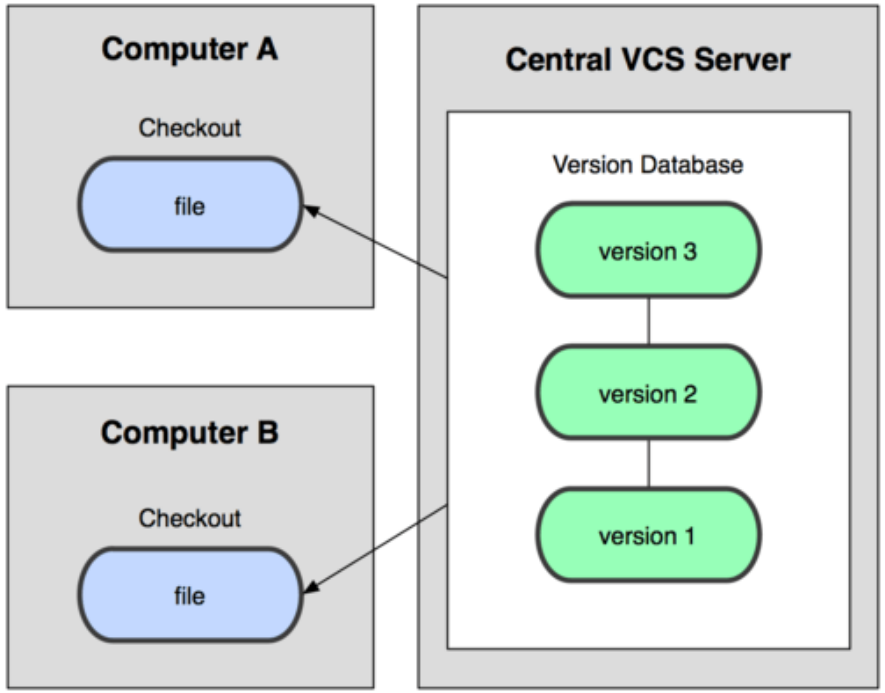
Hệ thống quản lý phiên bản cục bộ

- Version control system - VCS
- Có một database đơn giản lưu trữ tất cả các sự thay đổi của file dưới sự kiểm soát thay đổi



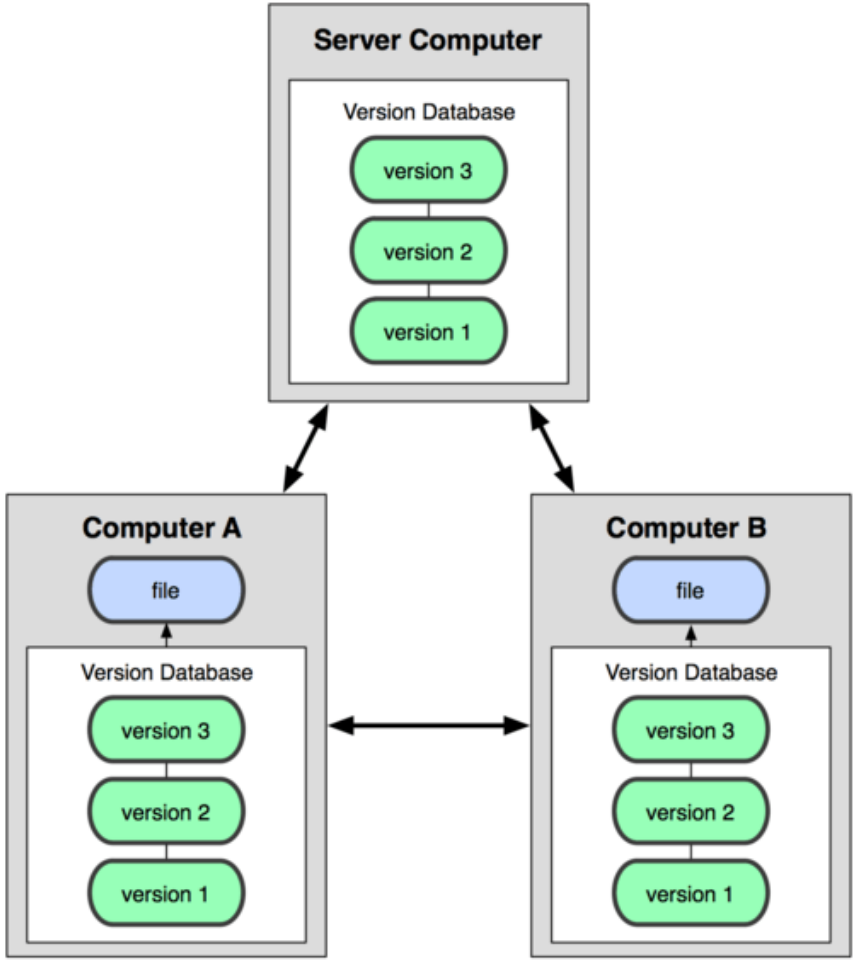
Hệ thống quản lý phiên bản tập trung

- Centralized Version Controls Systems – CVCs
  - Subversion, CVS, Perforce
- Gồm một máy chủ chứa tất cả các tập tin đã được "phiên bản hóa" và danh sách các máy khách có quyền thay đổi các tập tin trên máy chủ.



Hệ thống quản lý phiên bản phân tán

- Distributed Version Control Systems – DVCSs
  - Git, Mercurial, Darcs
- Các máy khách có thể sao chép về máy cục bộ hoặc sao chép ngược trở lại máy chủ



Kho lưu trữ- Repositories

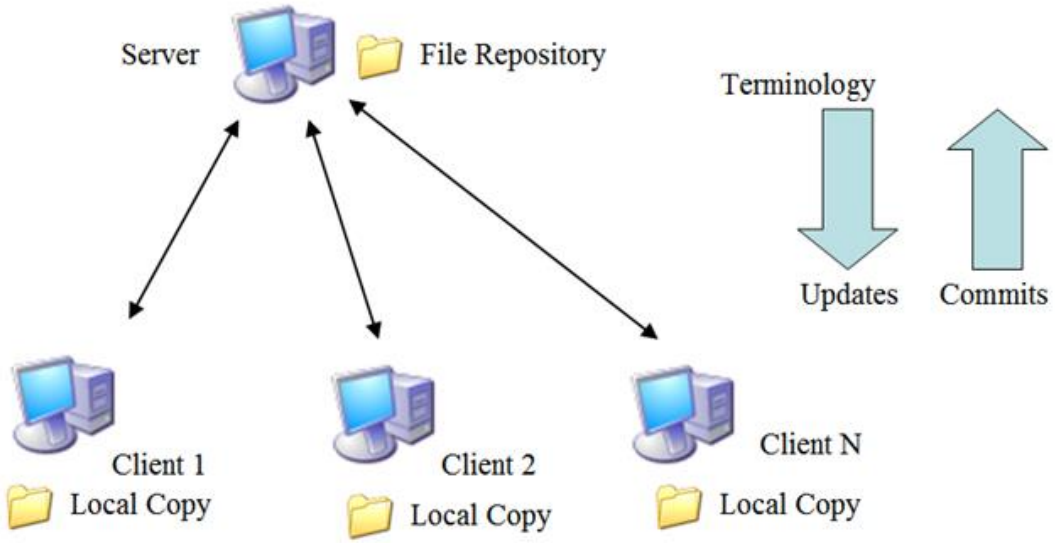
- Kho lưu trữ (hay còn gọi là "repo"): nơi lưu trữ bản sao của tất cả các tập tin.
  - Bạn không chỉnh sửa tập tin trực tiếp trong kho lưu trữ;
  - Bạn chỉnh sửa một bản sao
  - Sau đó bạn commit tập tin đã chỉnh sửa của bạn vào kho lưu trữ
- Trạng thái đang được lưu lại là lịch sử thay đổi của nội dung
- Đặt thư mục muốn quản lý lịch sử dưới sự quản lý của repository => ghi lại lịch sử thay đổi của thư mục và file trong thư mục đó

Kho lưu trữ- Nên đưa gì vào Repo?

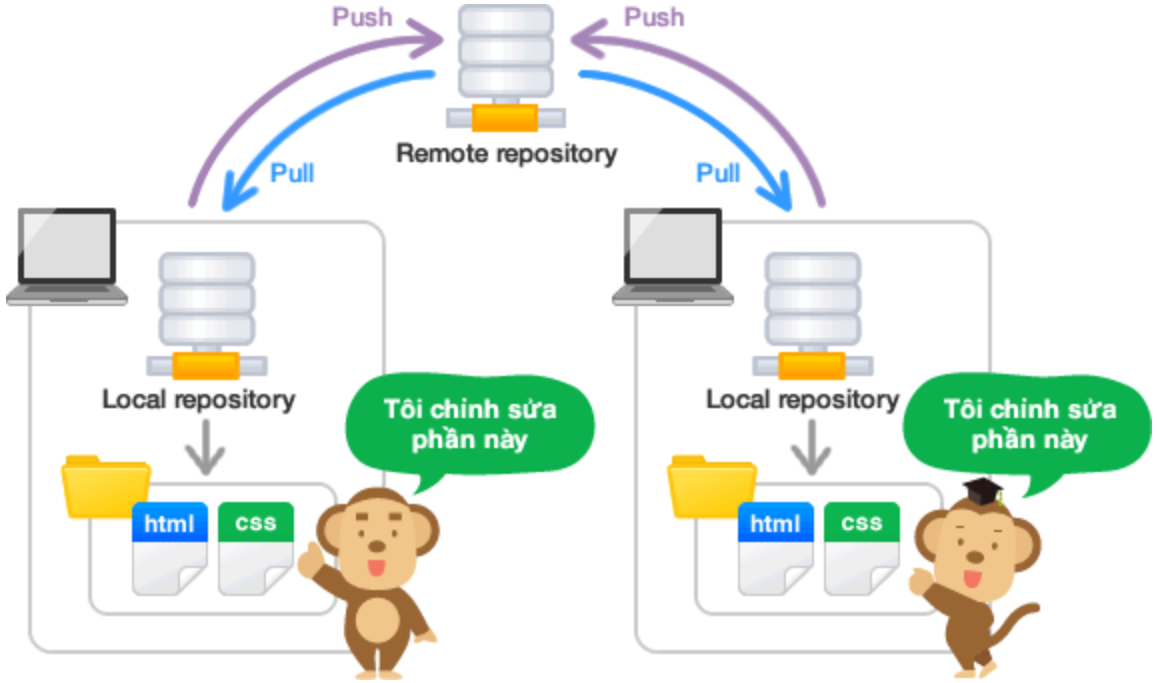
- Mọi thứ cần thiết để tạo dự án:
  - Mã nguồn (Ví dụ: .java, .c, .h, .cpp)
  - Các tập tin cấu hình, xây dựng chương trình (Makefile, build.xml)
  - Các tài nguyên khác: biểu tượng, văn bản, hình ảnh v.v.
- Những tập tin KHÔNG được đưa vào kho lưu trữ (dễ dàng được tạo lại, tiết kiệm dung lượng):
  - Tệp đối tượng (.o)
  - Các tập tin thực thi (.exe)

Kho lưu trữ- Repositories

- Một kho lưu trữ mà tất cả người dùng chia sẻ (CVS, Subversion)



- Mỗi người dùng cũng có thể có bản sao kho lưu trữ riêng của họ (Git, Mercurial)



Kho lưu trữ- Vị trí

PHENIKAA UNIVERSITY
Faculty of Computer Science

- Có thể tạo kho lưu trữ ở bất cứ đâu
  - Có thể đặt kho lưu trữ cục bộ đối với một dự án cá nhân mà bạn chỉ muốn bảo vệ khôi phục
- Tuy nhiên, thông thường kho lưu trữ cần được tạo:
  - Trên một máy tính hoạt động 24/7
    - Mọi người luôn có quyền truy cập vào dự án
  - Trên máy tính có hệ thống tập tin dự phòng (tức là RAID)
    - Không còn phải lo lắng về việc ổ cứng bị hỏng sẽ xóa sạch dự án của bạn nữa!
- Tùy chọn:
  - Phenikaa CS GitLab, GitHub (KHÔNG sử dụng GitHub để làm bài tập về nhà!!!)

Git là gì?

- Hệ thống quản lý phiên bản
- Miễn phí và có phí
- Sử dụng mô hình quản lý phiên bản phân tán

- Có thể sử dụng Git hoàn toàn cục bộ cho mục đích của riêng hoặc Sử dụng Git trên nhiều máy tính, nhiều người dùng hoặc một người dùng trên nhiều máy tính
- Máy chủ git: Github (github.com), GitLab (gitlab.com)
- Sách trực tuyến miễn phí: https://git-scm.com/book/en/v2
- Trang web Git: http://git-scm.com/

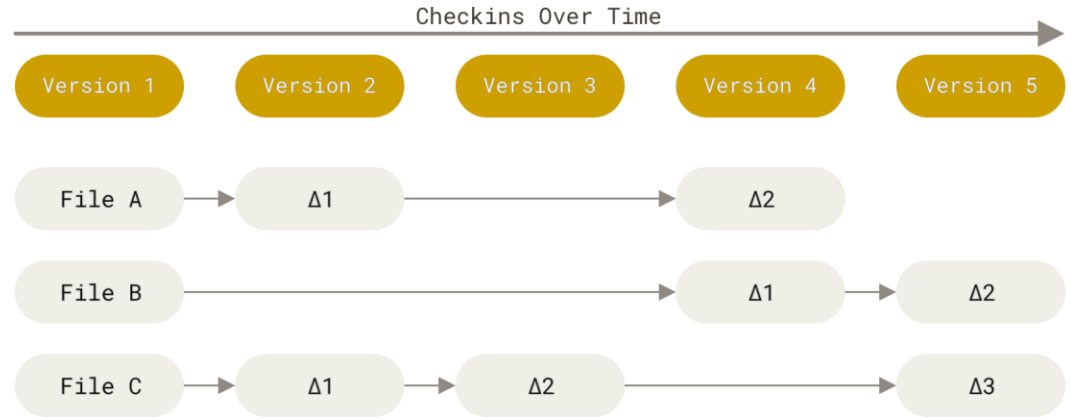
Lịch sử Git

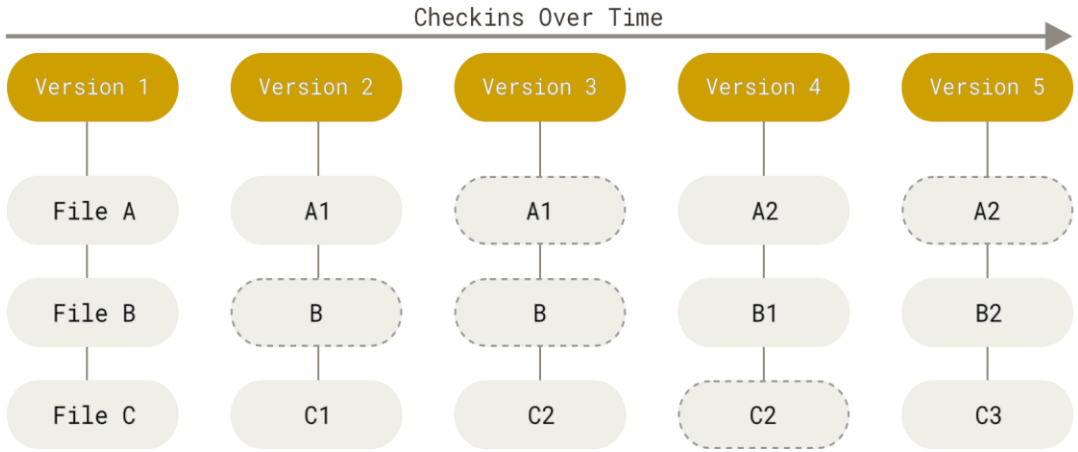
- Xuất phát từ cộng đồng phát triển Linux
- Linus Torvalds, 2005
- Mục tiêu ban đầu:
  - Tốc độ
  - Hỗ trợ phát triển phi tuyến tính (hàng ngàn nhánh song song)
  - Phân phối đầy đủ
  - Có khả năng xử lý các dự án lớn như Linux một cách hiệu quả

Git – Đặc điểm

PHENIKAA UNIVERSITY
Faculty of Computer Science

- Git lưu trữ dữ liệu dưới dạng các ảnh chụp của dự án qua thời gian





Git – Đặc điểm

- Gần như mọi thao tác đều là cục bộ
  - Hầu hết các thao tác trong Git chỉ cần các tệp và tài nguyên cục bộ để thực hiện
  - Tốt độ vượt trội do truy cập dữ liệu ngay trên máy cục bộ
  - Cho phép làm việc ngoại tuyến
- Git có tính toàn vẹn
  - Không thể thay đổi nội dung của bất kỳ tệp hay thư mục nào mà Git không biết.
  - Mọi tệp tin đều được kiểm tra với cơ chế hàm băm (SHA-1 hash)
- Git hầu như chỉ thêm dữ liệu
  - Các thao thác trong Git chỉ thêm dữ liệu vào CSDL của Git => luôn luôn có thể hoàn tác

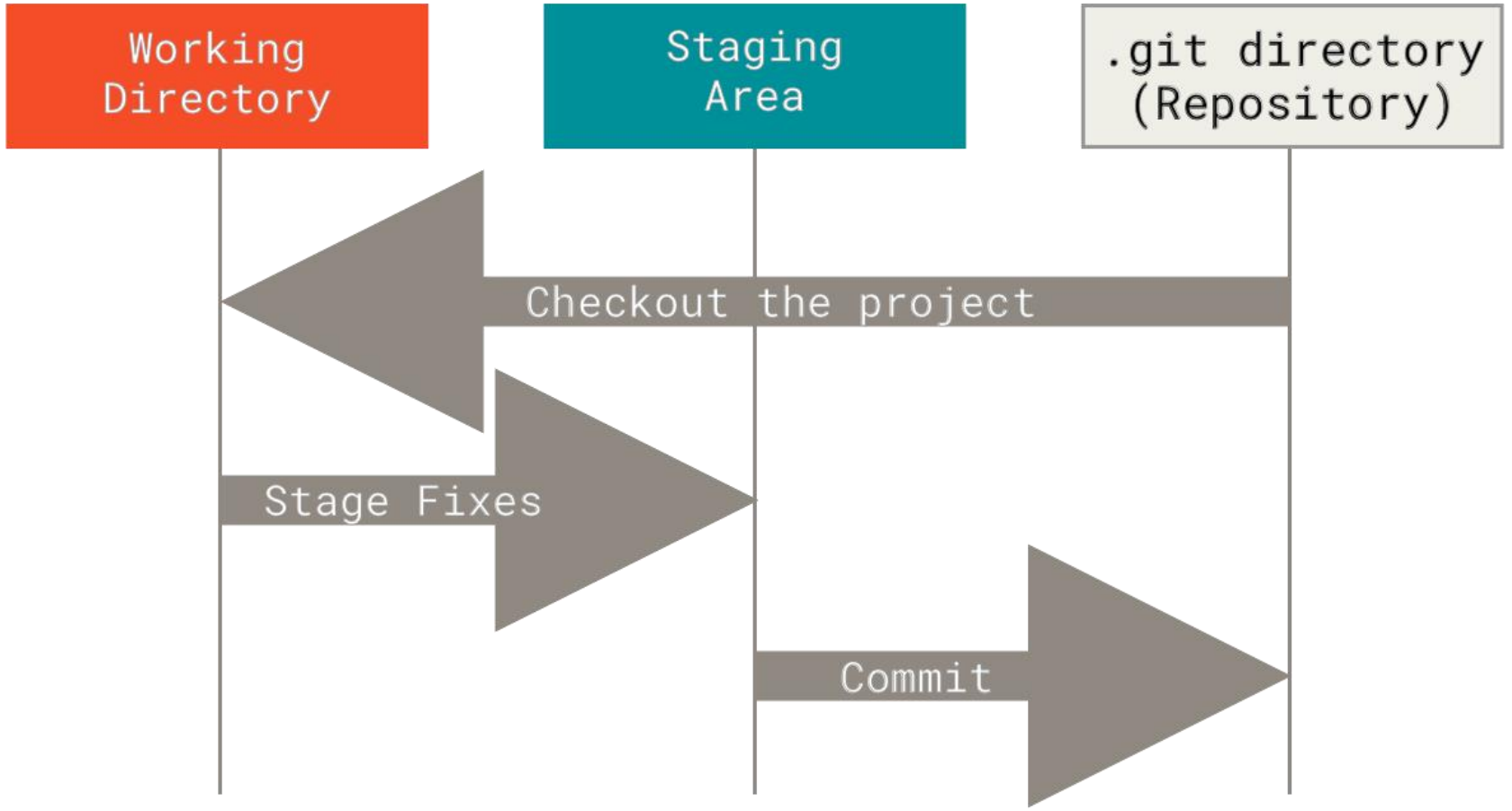
Ba trạng thái của tệp

- **Modified** (Đã sửa đổi): đã thay đổi tệp nhưng chưa commit vào CSDL
- **Staged** (Đã đánh dấu): đã đánh dấu một tệp đã sửa đổi ở phiên bản hiện tại để đưa vào ảnh chụp (commit) tiếp theo
- **Committed** (Đã cam kết): đã được lưu an toàn vào CSDL.

Ba khu vực của một dự án

- **Working tree:** Không gian làm việc trên máy local, nơi thực hiện thay đổi trực tiếp lên các tệp
- **Staging area (index):** khu vực trung gian giữa working tree và git directory. khi chạy lệnh git add, các thay đổi trong working tree sẽ được sao chép vào staging area, cho phép kiểm tra và chuẩn bị trước khi lưu trữ chính thức trong git directory
- **Git directory:** nơi lưu trữ toàn bộ lịch sử và cấu trúc repo, chứa tất cả các phiên bản của dự án, cung cấp các thông tin cần thiết trong trường hợp muốn khôi phục lại trạng thái của dự án

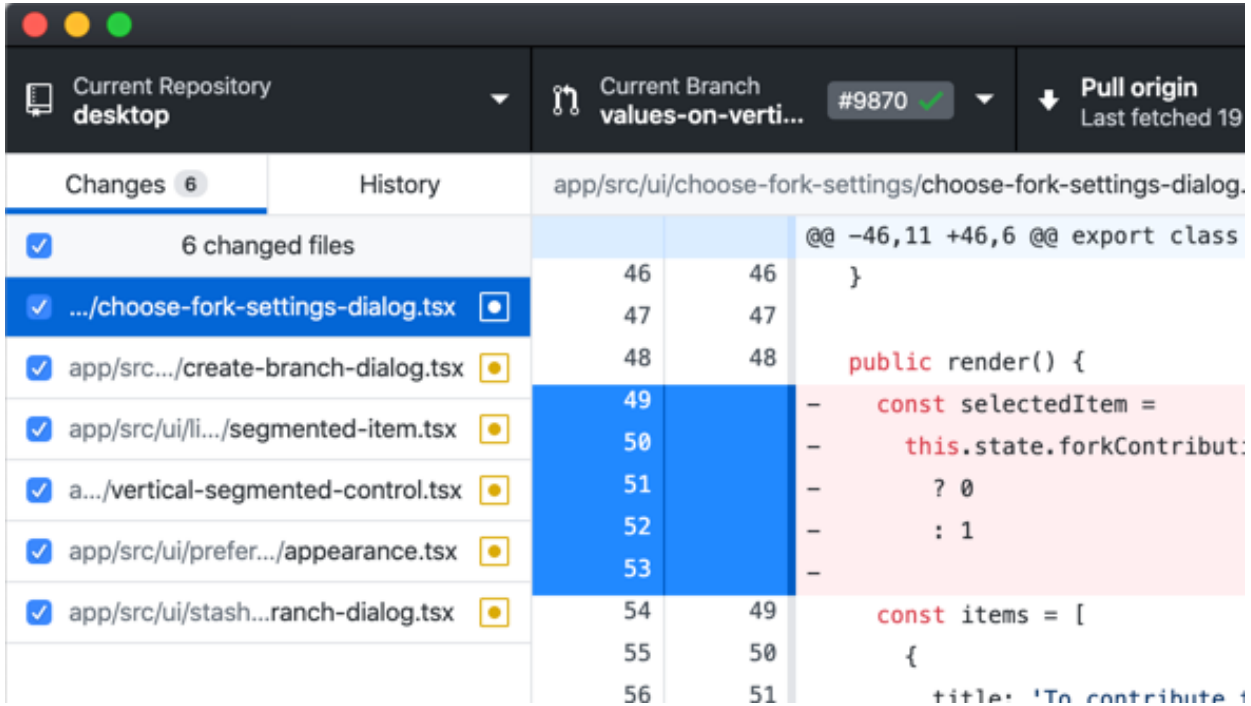
Ba phần chính của một dự án



Sử dụng Git

- Có nhiều cách khác nhau để sử dụng Git:
  - Sử dụng command line
  - Sử dụng các phần mềm có giao diện đồ họa:

    https://git-scm.com/downloads/guis
    - Github destop
    - SourseTree
    - TortoiseGit
    - Git Extenstions



Cài đặt Git

- Linux: https://git-scm.com/download/linux.
  - $ sudo apt install git-all
- MacOS: https://git-scm.com/download/mac.
  - $ git --version
- Windows: https://git-scm.com/download/win

Cấu hình môi trường Git

- Thiết lập tên và email cho Git để sử dụng khi commit:

```
$ git config --global user.name “Nguyen Thi Thuy Lien”
$ git config --global user.email thuyliennt@gmail.com
```

  - Xem lại thông tin đã thiết lập: `git config --list`.
  - `--global` thiết lập thông tin toàn cục => sử dụng ở mọi nơi, mọi lúc. Bỏ cờ --global nếu muốn thay đổi thiết lập ở một dự án/thư mục cụ thể.

- Thiết lập trình soạn thảo văn bản mặc định sử dụng với Git (default = vim):

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

Nhận trợ giúp

- Nhận trợ giúp khi sử dụng Git (hỏi lệnh/ý nghĩa lệnh):

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

Ví dụ: $ git help config

- Nhận trợ giúp nhanh/ngắn gọn

```
$ git <verb> -h
```

Qui trình làm việc

1. Modify files in your working directory.Stage files, adding snapshots of them to your staging area.

2. Do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory (your local copy of the repo).

❖ Notes:

- If a particular version of a file is in the git directory, it's considered committed.
- If it's modified but has been added to the staging area, it is staged.
- If it was changed since it was checked out but has not been staged, it is modified.

Git repository

❖Tạo một bản sao cục bộ của một kho lưu trữ: Hai tình huống phổ biến: (chỉ thực hiện một trong hai lựa chọn)

- Tạo Git repo trong thư mục hiện tại:

```
$ git init
$ git add readme.txt
$ git commit –m “initial project version”
```

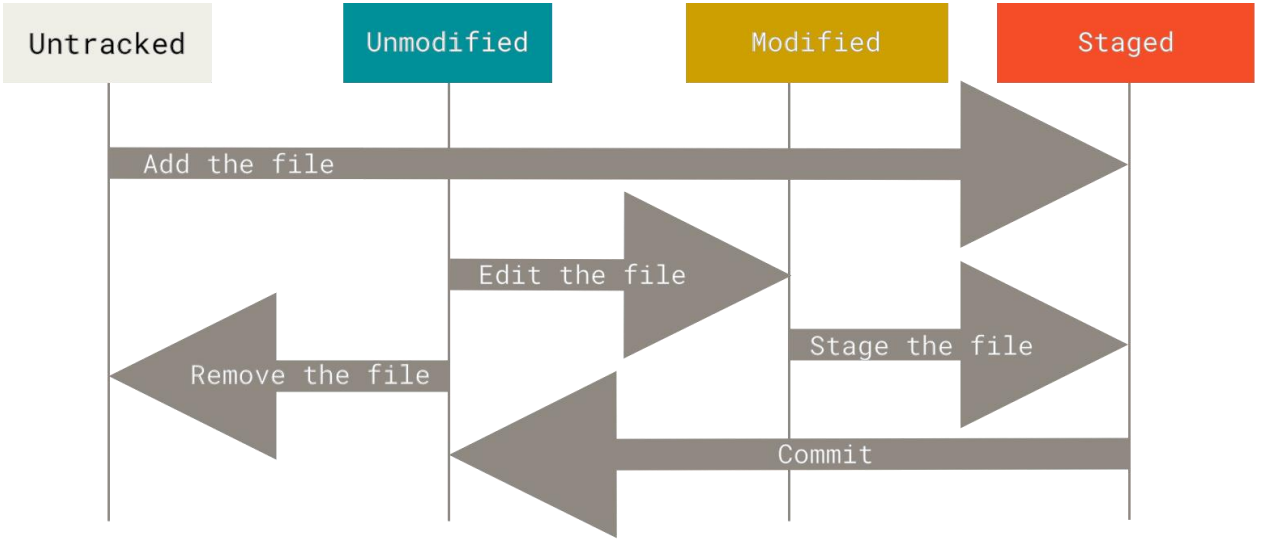
- Sao chép một kho lưu trữ đã tồn tại vào thư mục hiện tại của bạn:

```
$ git clone <url> [local dir name]
```

Ghi lại những thay đổi vào Repo

- Khi thay đổi/chỉnh sửa dự án đến một trạng thái nhất định => ghi lại
- Mỗi tệp trong thư mục làm việc có 2 trạng thái: tracked và untracked
  - Untracked: Các tệp trong thư mục làm việc và chưa từng được ghi lại (snapshot và không ở trong staging area)
  - Tracked: các tệp mà Git biết về chúng
- Vòng đời trạng thái của các tệp tin
  - Khi clone repo, mọi tệp đều là tracked và unmodified
  - Các file được chỉnh sửa => Modifed
  - Chọn lọc những file chỉnh sửa và commit

  ⇒Git ghi nhận sự thay đổi và chuyển trạng thái tệp về Unmodified



Kiểm tra trạng thái các tập tin

- Lệnh **git status** cho phép kiểm tra trạng thái của các tập tin trong thư mục làm việc và khu vực staging:

```
$ git status
$ git status -s
On branch main
nothing to commit, working tree clean
```

- Thêm nội dung vào file readme.txt và kiểm tra lại trạng thái:

```
$ echo "My project" > readme.txt
$ git status
On branch main
Changes not staged for commit:
        (use "git add <file>..." to include in what will be committed)
                modified: readme.txt
No changes added to commit (use "git add" to track)
```

- Bắt đầu theo dõi/stage các tệp tin mới: **git add** => tập tin được stage và sẽ được commit trong lần tới

Bỏ qua theo dõi các tệp tin

- Một số tệp không muốn Git tự đọng thêm hoặc chỉ hiển thị trên máy cá nhân (tệp được tạo tự động, tệp nhật ký hoặc tệp cấu hình….)
- Tạo file ignore có tên `.gitignore` thiết lập bỏ qua các tệp tin không cho Git theo dõi các tệp này bằng cách liệt kê tên tệp hoặc sử dụng các mẫu để khớp tên tệp

```
$ touch .gitignore
$ echo "*.[oa]" >.gitignore
```

- Cách viết mẫu (regix)
  - Dòng trống, hoặc bắt đầu bằng # => bỏ qua
  - ! => Phủ định mẫu
  - / => đường dẫn thư mục
  - Sử dụng các ký tự đại diện: *, [abc], [0-9]

```
*.a
!lib.a
doc/*.txt
```

Xem những thanh đổi đã được stage

- Để xem **sự khác biệt** giữa bạn **thư mục làm việc** và **khu vực staging**

```
$ git diff
```

- Để thấy sự khác biệt giữa **khu vực staging** và **bản sao cục bộ của kho lưu trữ**:

```
$ git diff --cached
```

Commit thay đổi

- Commit các tệp tin đã được stage:

```
$ git commit
```

- Nhập nội dung ghi chú cho bản cập nhật mới được commit (trình soạn thảo mặc định Vim)

```
# 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
```

- Commit trực tiếp với ghi chú bằng cách sử dụng cờ -m

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

- Commit tất cả các tệp thay đổi (Git tự động stage mọi file đang được theo dõi), sử dụng cờ -a:

```
$ git commit -a -m "Add new benchmarks"
```

Xóa tệp tin

- Để xóa một tệp khỏi Git, phải xóa tệp đó khỏi danh sách các tệp được theo dõi rồi commit (git sẽ xóa tệp khỏi thư mục làm việc và vùng stage)

```
$ git rm PROJECTS.md
```

- Để xóa một tệp khỏi Git và vẫn giữ tệp trong thư mục làm việc (chỉ xóa tệp khỏi vùng stage), sử dụng tùy chọn --cached

```
$ git rm --cached README
```

Di chuyển tệp tin

- Di chuyển/đổi tên tệp tin:

```
$ git mv file_from file_to
```

- Tương đương:

```
$ mv README.md README
$ git rm README.md
$ git add README
```

Xem lịch sử commit

PHENIKAA UNIVERSITY
Faculty of Computer Science

Để xem nhật ký của tất cả các thay đổi trong **kho lưu trữ cục bộ**:

```
$ git log   hoặc
$ git log --oneline (để hiển thị phiên bản ngắn hơn)
```

1677b2d Đã chỉnh sửa dòng đầu tiên của readme

258efa7 Đã thêm dòng vào readme

0e52da7 Cam kết ban đầu

- `git log -5` (chỉ hiển thị 5 bản cập nhật gần đây nhất, v.v.)

Lưu ý: các thay đổi sẽ được liệt kê theo commitID #, (SHA-1 hash)

Lưu ý: những thay đổi được thực hiện đối với kho lưu trữ từ xa trước lần cuối bạn sao chép/kéo từ kho lưu trữ đó cũng sẽ được bao gồm ở đây

Hoàn tác

- Hoàn tác lần commit cuối cùng, sử dụng tùy chọn –amend

```
$ git commit -m 'Initial commit'
$ git add forgotten_file
$ git commit --amend
```

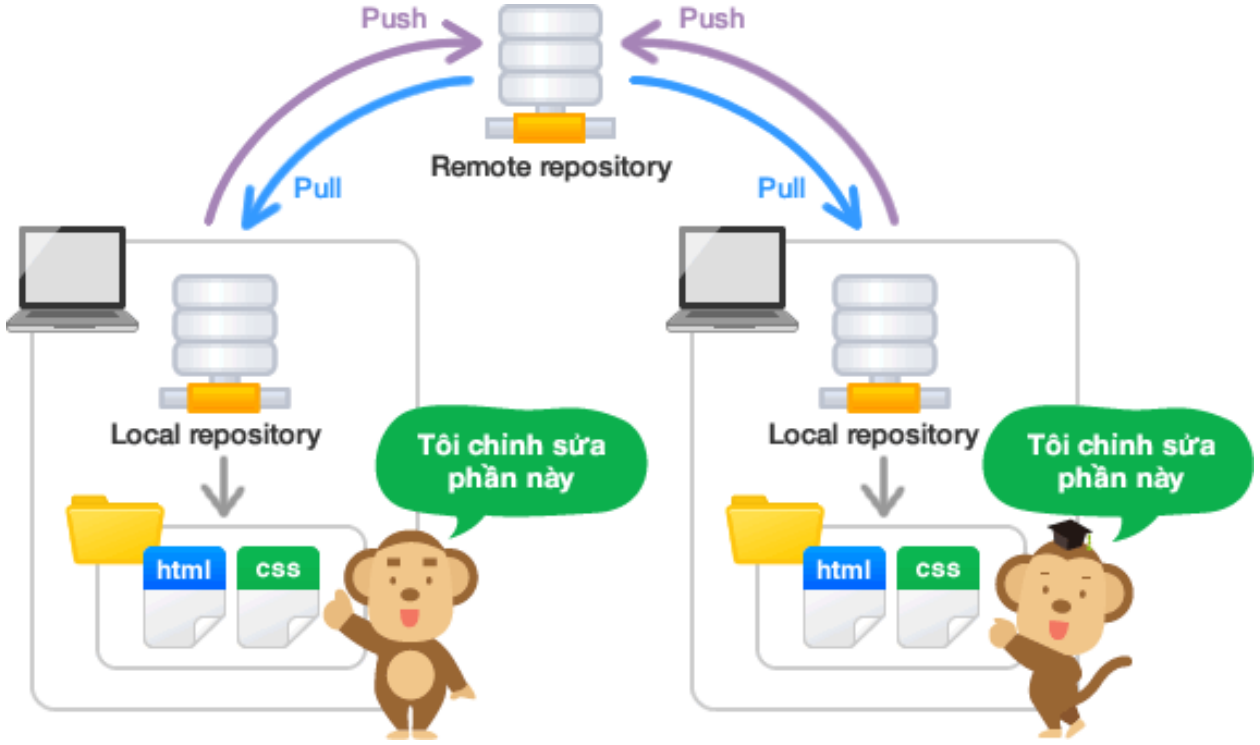
  - Trình chỉnh sửa commit-message chứa thông điệp của commit trước đó khởi động, có thể chỉnh sửa htoong điệp để ghi đè commit trước đó

- Hoàn tác ghi nhận sửa đổi/hoàn tác chỉnh sửa của tập tin

```
$ git restore --staged filename
$ git restore filename
```

Remote repository vs Local repository

- **Remote repository**: là repository để chia sẻ giữa nhiều người và bố trí trên server chuyên dụng
- **Local repository**: là repository bố trí trên máy cá nhân dành cho một người sử dụng



Làm việc với Remote repo

- Thêm remote repo:

```
$ git clone remote_repos
$ git remote
```

- Đẩy thay đổi đến remote repo:

```
$ git push origin
```

- Lấy dữ liệu từ remote repo để lấy những thay đổi gần đây nhất (sửa xung đột nếu cần, thêm và cam kết chúng vào kho lưu trữ cục bộ)

```
$ git fetch origin
$ git pull origin
```

Ghi chú: **`origin`** = một bí danh cho URL bạn đã sao chép từ

`master` = nhánh từ xa mà bạn đang kéo từ/đẩy tới,

(chi nhánh cục bộ mà bạn đang kéo tới/đẩy ra là chi nhánh hiện tại của bạn)

Lệnh Git thường dùng

Lệnh	Mô tả
`git clone url [dir]`	sao chép kho lưu trữ git để bạn có thể thêm vào đó
`git add files`	thêm nội dung tập tin vào vùng dàn dựng
`git commit`	ghi lại ảnh chụp nhanh của khu vực dàn dựng
`git status`	xem trạng thái của các tập tin của bạn trong thư mục làm việc và khu vực dàn dựng
`git diff`	hiển thị sự khác biệt giữa những gì được dàn dựng và những gì được sửa đổi nhưng không được dàn dựng
`git help [command]`	nhận thông tin trợ giúp về một lệnh cụ thể
`git pull`	lấy từ một kho lưu trữ từ xa và thử hợp nhất vào nhánh hiện tại
`git push`	đẩy các nhánh và dữ liệu mới của bạn vào kho lưu trữ từ xa
others: `init, reset, branch, checkout, merge, log, tag`	

Tránh các vấn đề thường gặp

- Không chỉnh sửa kho lưu trữ (thư mục .git) theo cách thủ công. Nó không được thiết kế để con người có thể sửa đổi.
- Cố gắng không thực hiện nhiều thay đổi lớn cùng một lúc. Thay vào đó, hãy thực hiện nhiều lần commit, mỗi lần có một mục đích logic duy nhất. Điều này sẽ giảm thiểu xung đột hợp nhất.
- Luôn luôn git pull trước khi chỉnh sửa một tệp. Rất dễ quên điều này. Nếu bạn quên, bạn có thể sẽ chỉnh sửa một phiên bản lỗi thời, điều này có thể gây ra xung đột hợp nhất khó chịu.
- Đừng quên git push sau khi bạn đã thực hiện và cam kết thay đổi. Chúng không được sao chép vào kho lưu trữ từ xa cho đến khi bạn thực hiện push.

Phân nhánh

Để tạo một nhánh có tên là thử nghiệm:

```
$ git branch experimental
```

Để liệt kê tất cả các nhánh: (* hiển thị nhánh bạn hiện đang ở)

```
$ git branch
```

Để chuyển sang nhánh thử nghiệm:

```
$ git checkout experimental
```

Sau đó, những thay đổi giữa hai nhánh sẽ khác nhau để hợp nhất những thay đổi từ nhánh thử nghiệm vào nhánh chính:

```
$ git checkout master
$ git merge experimental
```

Ghi chú: **`git log -- graph`** có thể hữu ích để hiển thị các nhánh.

Lưu ý: Các nhánh này nằm trong*kho lưu trữ cục bộ của bạn*!

Tóm tắt

- Bạn *sẽ* sử dụng phần mềm kiểm soát phiên bản khi làm việc trên các dự án, cả ở đây và trong ngành
- Lời khuyên: chỉ cần thiết lập một kho lưu trữ, ngay cả đối với các dự án nhỏ, nó sẽ giúp bạn tiết kiệm thời gian và công sức

PHENIKAA UNIVERSITY

Faculty of Computer Science

Q&A