GIT

-là 1 VCS - Version Control System (quản lý phiên bản).

-quản lý lịch sử của các file, dòng code,....

-các nhóm có thể cộng tác trong 1 dự án.

-cho phép ta khôi phục lại phiên bản trước đó.

Nguyên tắc làm vc của GIT:

-có 1 kho lưu trữ trung tâm - repository

-có thể lấy dữ liệu về, thực hiện các công việc, cập nhật dữ liệu ngược lên kho trung tâm đó.

-Git cung cấp các công cụ để giải quyết xung đột.

Lợi ích:

-đảm bảo ta ko làm hỏng bất cứ thứ gì vĩnh viễn.

-giúp chúng ta có thể cộng tác vs người, nhóm khác.

-phát triển phiên bản mới mà ko ảnh hưởng đến phiên bản cũ.

-có thể quay lại phiên bản trước bất kỳ lúc nào bạn muốn.

-chia sẻ vs người khác về kết quả của chúng ta.

Các câu lệnh cmd cơ bản vs windows:

- cd : change directory - thay đổi thư mục.
- cd .. : quay về thư mục cha
- dir : hiện thị các tập tin, thư mục đang có
- ls : giống dir, nhưng tô màu để ta phân biệt được thư mục, file, file có thể thực thi.
- mkdir "new folder name" : tạo ra 1 thư mục mới.
- touch "new file name" : tạo ra 1 tập tin mới.
- echo : xuất/in một nội dung nào đó.
- echo "text" > "file name" : ghi "text" vào file "file name". Chú ý: nó sẽ ghi đè override lên toàn bộ nội dung trong file đó.
- echo "text" >> "file name" : ghi text mới vào mang ko override nd cũ.
- cat : hiện thị nội dung bên trong file.
- diff file1 file2 : so sánh sự khác biệt giữa file 1 và file2
- rm "file name" : xóa file
- rm -d "folder name" : xóa thư mục. Tuy nhiên, phải là 1 thư mục rỗng.
- rm -r "folder name" : xóa thư mục. là 1 thư mục có dữ liệu.

==========

==========

-repository : (repo) kho lưu trữ

-commit : một đơn vị làm việc

-branch : nhánh
-main/master: tên của repo chính (main repo)
-merge/rebase : kết hợp 2 nhánh lại vs nhau.
-develop : tên của nhánh, hoặc tên của 1 lập trình viên.

Các câu lệnh vs Git:
- git --help : hiện trợ giúp, hướng dẫn.
- git --version : hiện thị thông tin phiên bản git nếu đã được download.
- git status : hiện thị trạng thái kho lưu trữ.
- git log : hiện thị lịch sử các commit.
- git init [repo name] : tạo ra 1 kho lưu trữ repo trống.
- git clone [repo name] [clone name] : để tạo 1 bản sao được liên kết với repo của chúng ta.
- git config -l : xem cấu hình hiện tại.
- git config -l [--scope] [option_name] [value] :
  scope : --system -> ảnh hưởng đến tất cả người dùng hệ thống.
  --global -> liên quan đến tất cả repo(s) của người dùng.
  --local -> liên quan đến 1 repo hiện tại đang làm vc.
- Việc cấu hình - config sẽ giúp ta biết được những thông tin của người tác động vào git.
- git add [file name(s)] : thêm tệp vào Index
- git add . : thêm tất cả các tệp (Lưu ý: git add . thêm tất cả các thay đổi trong thư mục vào staging area(index) trước khi commit. Index sẽ giữ 1 bản sao của các tệp, thư mục)
- git commit -m"comment" : tạo commit đưa lên repo
- git status : so sánh sự khác biệt giữa 3 cây: working directory(là ko gian làm việc của mình, nơi ta thấy và chỉnh sửa các tệp, thư mục), index(staging area), head(là con trỏ trỏ đến commit hiện tại, khi thực hiện commit, git tạo ra 1 bản ghi (commit) mới và di chuyển con trỏ head đến commit mới đó.)
- git diff : ss sự khác biệt với commit cuối cùng.

==========
==========

Bài 07. Cấu hình GITIGNORE để bỏ qua các file không cần giám sát

Cho đuôi của file, hoặc tên thư mục vào file có tên là .gitignore để bỏ qua những file đó khi add . (file .gitignore là do mình tự tạo)

==========
==========

## Bài 08. Cách tương tác với Remote Repository

Central Repository: Đây là nơi chứa trung tâm của mã nguồn dự án. Tất cả các nhà phát triển khác đều đưa ra và đưa vào central repository để chia sẻ công việc của họ. Central repository thường được đặt trên một máy chủ có thể truy cập được từ xa.

Central repository thường được đặt trên một máy chủ hoặc dịch vụ lưu trữ trực tuyến. Khi bạn tạo một central repository, bạn đang tạo một nơi chứa chính để lưu trữ mã nguồn của dự án và chia sẻ nó với các nhà phát triển khác.

Local repositories (thư mục lưu trữ cục bộ) của nhà phát triển là nơi mà mỗi nhà phát triển lưu trữ phiên bản của dự án trên máy tính cá nhân của họ. Đây là một số điểm chính về local repositories: Working Directory, Staging Area, Commit History, Branches, Interaction with Central Repository.

- git init --bare : tạo 1 central repo
  ( --bare là quan trọng ở đây, nó cho biết bạn đang tạo ra một bare repository, tức là một repository không có working directory, chủ yếu được sử dụng làm central repository.)
- git clone [repo_name] [clone_name] : sao chép và liên kết repo_name
- git fetch : lấy các thông tin về commit mới từ central
- git pull : lấy dữ liệu từ central về local repo
- git push : đẩy các commit từ local về central

## Bài 09. Xử lý xung đột trong Git - Merge Conflict

https://www.youtube.com/watch?v=Pg6DSWGsiv8&list=PLyxSzL3F7485Xgn7novgNxnou8QU6i485&index=9

## Bài 10. Câu lệnh GIT CHECKOUT chuyển đổi giữa các commit trong Git

HEAD(là con trỏ trỏ đến commit hiện tại, khi thực hiện commit, git tạo ra 1 bản ghi (commit) mới và di chuyển con trỏ head đến commit mới đó.)

- git check out abc123 : quay lại, di chuyển tới bất kỳ commit nào đó. Trong trường hợp này, abc123 là hash của commit mà bạn muốn khôi phục.

## Bài 11. Branches - cách làm việc với nhiều nhánh trong Git

- git branch <branch_name> : tạo branch
- git checkout <branch_name> : chuyển sang nhánh khác
- git branch -l : xem nhánh hiện tại ta đang làm việc(được in màu khác) và danh sách các nhánh khác nếu có.

Ngoài ra, nếu muốn đổi tên nhánh master -> main: git branch -M main

## Bài 12. Git Merge - Kết hợp nội dung từ các nhánh

- Đứng ở nhánh A: git merge B : sẽ merge A vs B, có thể sẽ xảy ra conflicts.
- git log --oneline : được sử dụng để hiển thị lịch sử commit trong một định dạng ngắn gọn, chỉ hiển thị mã hash của commit và thông điệp commit (commit message) trên mỗi dòng.

## Bài 13. Git Rebase - tái cơ sở cho một nhánh trong Git

Quá trình rebase sẽ đưa các commit từ nhánh feature_branch và áp dụng lên cuối cùng của nhánh master. Điều này giúp duy trì một lịch sử commit sạch sẽ và không có merge commit.

- git rebase <tên nhánh> :Lệnh này sử dụng để rebase nhánh hiện tại của bạn lên đằng trước nhánh được chỉ định. Ví dụ, nếu bạn đang ở trên nhánh feature_branch và muốn rebase lên nhánh master, bạn có thể sử dụng: git rebase master
- git rebase --continue : Khi xảy ra xung đột trong quá trình rebase và bạn đã giải quyết xung đột, bạn sử dụng lệnh này để tiếp tục quá trình rebase.
- git rebase --skip : Khi bạn gặp một commit mà bạn muốn bỏ qua (skip) trong quá trình rebase, bạn có thể sử dụng lệnh này. Ví dụ Git báo hiệu xung đột trong quá trình rebase và dừng lại. Nếu bạn quyết định bỏ qua commit hiện tại thì chạy câu lệnh trên.

# Khi bạn làm việc trên một nhánh đã được chia sẻ (shared branch), điều quan trọng là giữ cho lịch sử commit của nhánh đó ổn định và dễ theo dõi cho tất cả các thành viên trong nhóm. Khi bạn sử dụng git rebase và thực hiện các thao tác như --skip, nó có thể thay đổi lịch sử commit của nhánh đó.

Cụ thể, khi bạn sử dụng git rebase --skip để bỏ qua một commit trong quá trình rebase, bạn đang xóa commit đó khỏi lịch sử của nhánh và thay thế nó bằng các commit khác. Điều này có thể tạo ra sự không nhất quán trong lịch sử commit của nhánh, đặc biệt là nếu đã có nhiều người làm việc trên nhánh đó.

--> Nếu bạn thực sự cần phải sử dụng git rebase --skip, hãy đảm bảo rằng mọi người trong nhóm đều biết về sự thay đổi này và có thể điều chỉnh làm việc của họ tương ứng.

CHÚ Ý: Khi rebase hay merge, có thể xảy ra xung đột conflict, sau khi giải quyết xung đột xong, ta phải git add . để xác nhận là đã giải quyết xong, sau đó commit và push là ok.

## Bài 14. Cách xóa nhánh trong GIT

Nếu đứng từ nhánh A để merge nhánh B vào thì B sẽ được gộp vào. Nhưng nhánh B vẫn tồn tại. Do đó, nếu ta ko muốn giữ lại nhánh B thì có thể xóa đi.
- git branch -d <tên nhánh> : xóa nhánh. Lệnh này chỉ xóa đi nhánh local, còn trên remote vẫn tồn tại.
Nếu muốn xóa luôn nhánh đó trên cả remote thì dùng lệnh:
- git push origin -d <tên nhánh>

Bổ xung:
- git branch -a : Lệnh này sẽ hiển thị tất cả các nhánh có sẵn trong repository, bao gồm cả nhánh local và remote.
Ví dụ sẽ có kết quả như sau:
* master
feature_branch
remotes/origin/master
remotes/origin/feature_branch
- git branch -l :Lệnh này sẽ hiển thị chỉ các nhánh local có sẵn trong repository.
Chú ý:
Dấu * đánh dấu nhánh mà bạn đang hiện đại (HEAD đang trỏ đến).
Nhánh local được hiển thị trực tiếp dưới dòng lệnh, trong khi nhánh remote được hiển thị với tiền tố remotes/.
Lệnh git branch -a thường được sử dụng để kiểm tra cả nhánh local và remote, đặc biệt là khi bạn muốn xem tất cả các nhánh có sẵn trong repository, kể cả những nhánh remote mà bạn chưa fetch về.
Lệnh git branch -l thường được sử dụng để chỉ xem thông tin về các nhánh local mà bạn đã tạo và làm việc trên đó.

## Bài 15. Git Reset - Hủy bỏ commit

- git reset --soft <commit id> : di chuyển HEAD về vị trí commit. Trạng thái của stage và tất cả sự thay đổi của file được giữ nguyên.
- git reset <commit id> : di chuyển HEAD về vị trí commit reset, vẫn giữ nguyên tất cả thay đổi của file, nhưng loại bỏ các thay đổi stage.
- git reset --hard <commit id> : di chuyển con trỏ HEAD về vị trí commit reset và loại bỏ tất cả sự thay đổi của file. Nếu dùng --hard --> sẽ không thể khôi phục dữ liệu. Cần cẩn trọng.
Tóm lại:
git reset --soft: Giữ nguyên thay đổi trong "staging area" và "working directory".
git reset: Mặc định là git reset --mixed. Hủy bỏ thay đổi trong "staging area" nhưng giữ nguyên

"working directory".
git reset --hard: Hủy bỏ tất cả thay đổi trong "staging area" và "working directory". Cả "working directory" trở về trạng thái của commit trước đó.

Bài 16. Git Revert - Quay lại các commit trước đây

Quay lại commit trước mà không hủy bỏ commit mới.
- git revert <commit-SHA>
git revert rất hữu ích khi bạn muốn giữ lại lịch sử commit đã được chia sẻ với người khác và không muốn thay đổi nó. Tuy nhiên, nó cũng có thể tạo ra nhiều commit mới và làm cho lịch sử commit trở nên phức tạp hơn.

VD:
Trạng thái ban đầu: A---B (master) -> Commit B thay đổi trạng thái của dự án từ A đến B.
Chạy git revert B: git revert B -> Khi bạn chạy git revert B, Git tạo ra commit R để thay đổi trạng thái từ B trở lại A.
Kết quả là tạo ra một commit mới R đảo ngược sự thay đổi của commit B: A---B---R (master) --> Kết quả là, trạng thái của commit R sẽ giống như nếu commit B không bao giờ tồn tại.
Khi bạn xem xét lịch sử commit của nhánh master sau khi thực hiện git revert B, bạn sẽ thấy rằng commit R được thêm vào, và trạng thái của commit R là giống như trạng thái của commit A. Commit B vẫn tồn tại trong lịch sử commit, nhưng nó không ảnh hưởng đến trạng thái hiện tại của nhánh master nữa.
Điều quan trọng là git revert không làm thay đổi commit gốc (commit B trong trường hợp này), mà tạo ra một commit mới để đảo ngược thay đổi. Điều này giữ cho lịch sử commit linh hoạt và an toàn, đặc biệt là khi bạn đã chia sẻ lịch sử commit với người khác.

---

# Câu lệnh **git checkout -b feature/1-add-cart.model-file develop** sẽ tạo ra một nhánh mới có tên **feature/1-add-cart.model-file** từ nhánh **develop** và chuyển sang nhánh mới đó. Dưới đây là giải thích chi tiết:

- **git checkout -b**: Tạo và chuyển sang một nhánh mới.
- **feature/1-add-cart.model-file**: Tên của nhánh mới. Trong đây, **feature/1-add-cart.model-file** có thể được hiểu là một nhánh đang được phát triển để thêm chức năng liên quan đến việc thêm một file model cho giỏ hàng.
- **develop**: Tên của nhánh nguồn, từ đó bạn đang tạo ra nhánh mới. Trong ngữ cảnh

này, đây có thể là nhánh phát triển chung cho toàn bộ dự án.

# với **-b**, câu lệnh sẽ thực hiện cả hai bước:

1. **Tạo nhánh mới:** Tên nhánh mới sẽ là **feature/1-add-cart.model-file**.
2. **Chuyển sang nhánh mới đó:** Sau khi tạo nhánh mới, bạn sẽ ngay lập tức được chuyển sang nhánh mới tạo ra.

➔ **-**b vừa giúp ta tạo nhánh mới (git branch ten_nhanh_mơi) vừa giúp ta chuyển đến đó git checkout ten_nhanh.

➔ Nếu ko có -b thì lệnh trên sẽ thành chuyển sang nhánh feature... đã được tạo trước đó rồi, nếu ko có thì lỗi.

# Lệnh **git tag "v1.0.0"** được sử dụng để tạo một tag mới với tên là "v1.0.0". Tag trong Git là một cách đánh dấu một commit cụ thể với một tên nhất định, thường được sử dụng để chỉ định các phiên bản phần mềm hay các điểm cố định quan trọng trong lịch sử của dự án. Sau khi chạy lệnh trên, tag "v1.0.0" sẽ được tạo và được gắn vào commit hiện tại trên nhánh bạn đang đứng.

# Lệnh **git push --tags** được sử dụng để đẩy (push) tất cả các tags từ local repository lên remote repository.

- **git push**: Là lệnh để đẩy các thay đổi từ local repository lên remote repository.
- **--tags**: Là tùy chọn để đẩy tất cả các tags.

Khi bạn chạy lệnh này, tất cả các tags có sẵn trong local repository sẽ được đẩy lên remote repository. Điều này bao gồm cả các tags đã được tạo bằng lệnh **git tag** mà bạn đã tạo trước đó.

Nếu bạn chỉ muốn đẩy một tag cụ thể, bạn có thể sử dụng lệnh **git push origin <tên-tag>**

###

Git Flow là một mô hình quản lý nhánh Git được thiết kế để hỗ trợ quy trình phát triển dự án phần mềm hiệu quả và có tổ chức. Mô hình này đặc biệt phù hợp cho các dự án lớn và dài hạn, nơi quy trình làm việc cần sự phân chia rõ ràng giữa các giai đoạn phát triển, kiểm thử và triển khai.

Dưới đây là các nhánh chính và quy trình làm việc của Git Flow:

**Nhánh Chính:**

1. **main (hoặc master):**
    - Nhánh chính và ổn định của dự án.
    - Chứa mã nguồn ổn định đã được kiểm thử và chấp nhận.

- Mỗi lần thực hiện release, tag sẽ được đặt tại commit trên nhánh này.

2. **develop:**
    - Nhánh phát triển chung cho dự án.
    - Các tính năng mới và cải tiến được phát triển ở đây.
    - Mỗi khi có thay đổi trên **develop**, làm việc đó nên được tích hợp và kiểm thử kỹ lưỡng.

**Nhánh Hỗ Trợ:**

3. **feature/xxx:**
    - Được tạo từ **develop**.
    - Sử dụng để phát triển một tính năng cụ thể.
    - Khi tính năng hoàn thành, được merge lại vào **develop**.
4. **release/xxx:**
    - Được tạo từ **develop** khi chuẩn bị cho một phiên bản phát hành.
    - Kiểm thử cuối cùng, sửa lỗi nếu có, và làm các công việc liên quan đến chuẩn bị phát hành.
    - Khi chuẩn bị xong, merge vào cả **develop** và **main**, đặt tag cho phiên bản.
5. **hotfixes/xxx:**
    - Được tạo từ **main**.
    - Sử dụng khi cần sửa ngay lập tức lỗi trên production.
    - Chỉ chứa các sửa lỗi cụ thể, không chứa các tính năng mới.
    - Khi sửa lỗi hoàn thành, merge vào cả **main** và **develop**.

**Quy Trình Làm Việc:**

1. **Phát triển tính năng:**
    - Tạo một nhánh **feature** từ **develop**.
    - Phát triển và kiểm thử tính năng trên nhánh **feature**.
    - Merge nhánh **feature** vào **develop** khi tính năng hoàn thành.
2. **Chuẩn bị và phát hành:**
    - Tạo nhánh **release** từ **develop**.
    - Chuẩn bị phiên bản, sửa lỗi nếu cần.
    - Merge nhánh **release** vào **develop** và **main**.
    - Đặt tag cho phiên bản.
3. **Sửa lỗi trên production:**
    - Tạo nhánh **hotfix** từ **main**.
    - Sửa lỗi trên nhánh **hotfix**.
    - Merge nhánh **hotfix** vào **main** và **develop**.

Mô hình Git Flow giúp đảm bảo tính ổn định và kiểm soát trong quá trình phát triển, đồng

thời cho phép các nhóm làm việc độc lập trên các tính năng mà không làm ảnh hưởng đến nhau. Nó cũng tạo ra một lịch sử commit dễ đọc và theo dõi.

# Nếu bạn chỉ muốn sao lưu (backup) một nhánh cụ thể trong kho lưu trữ Git của mình, bạn có thể sử dụng lệnh **git branch** để tạo một nhánh mới từ nhánh cần sao lưu. Dưới đây là các bước thực hiện:

1. **Tạo Nhánh Backup:**

git branch backup_branch <tên_nhánh_cần_sao_lưu>

- **<tên_nhánh_cần_sao_lưu>**: Tên của nhánh bạn muốn sao lưu.

Lệnh này sẽ tạo một nhánh mới có tên **backup_branch** từ nhánh cần sao lưu, nhưng nó chỉ tạo ra một tham chiếu mới và không tạo bản sao thực tế của tất cả các commit.