hướng dẫn cách hoạt động của git

1. Kiểm tra nhánh hiện tại

```
git branch
```

- Git sẽ liệt kê tất cả các nhánh có trên máy của bạn. **Nhánh hiện tại sẽ có một dấu sao `*` ở**
  Ví dụ:

```
* develop
  main
```

Hiện tại, bạn đang ở nhánh `develop`.

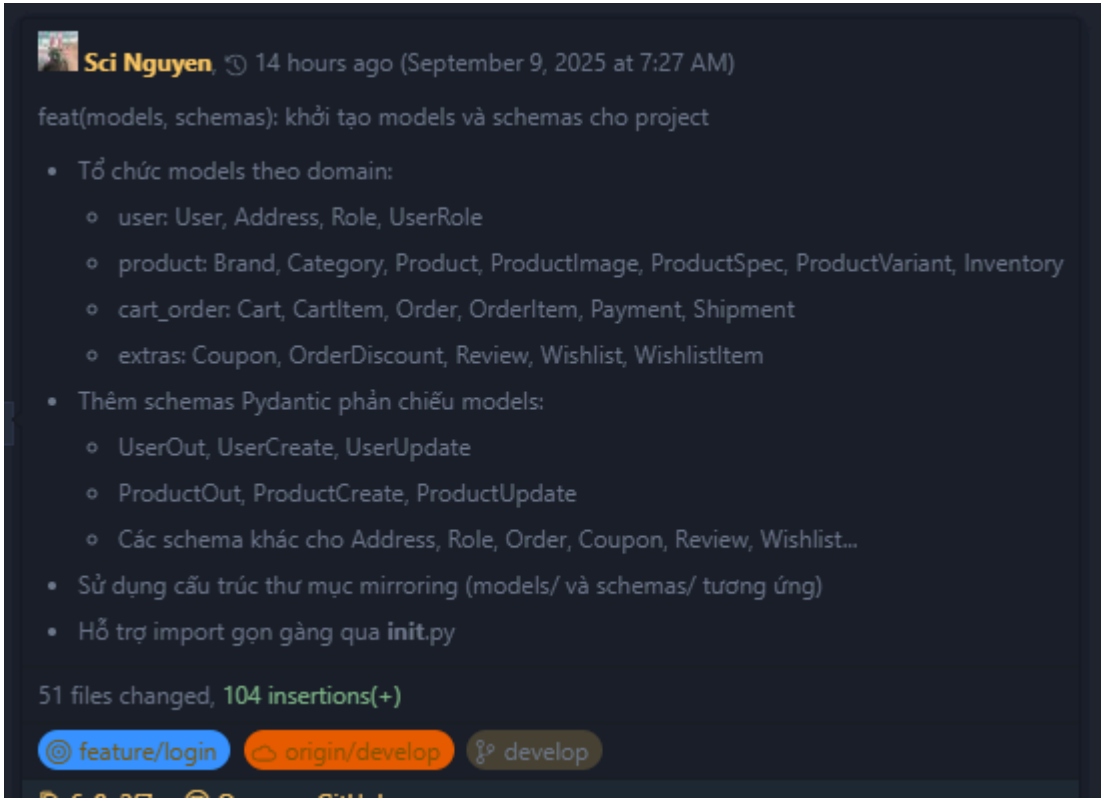
2. Tạo nhánh con - Lưu ý là luôn tạo nhánh con trước rồi mới code

```
git checkout -b {tên nhánh con}
```

Ví dụ:

```
git checkout -b feature/login
```

*Nếu bạn muốn kiểu tra nhánh cha của nhánh con vừa mới tạo chúng ta dùng extension **Git Graph***



*Bạn sẽ thấy là tên feature/login vừa mới tạo và nhánh đằng sau là develop thì **develop là cha của nhánh feature/login***

3. Khi hoàn thành xong code

```
git status
```

Dùng để kiểm tra tất cả các file đã tương tác tới

4. Chuẩn bị thay đổi cho Commit (Staging Changes)

Trước khi tạo một "bản ghi lịch sử" (commit), bạn cần cho Git biết chính xác những thay đổi nào bạn muốn đưa vào bản ghi đó. Hành động này gọi là "staging" - đưa các thay đổi vào một khu vực chờ.

Hãy tưởng tượng **khu vực chờ (Staging Area)** giống như giỏ hàng của bạn khi đi siêu thị. Bạn chọn những món đồ (các file thay đổi) và bỏ vào giỏ hàng (`git add`) trước khi ra quầy thanh toán (`git commit`).

Cách 1: Đưa TẤT CẢ thay đổi vào khu vực chờ

Nếu bạn muốn đưa tất cả các file bạn vừa chỉnh sửa, thêm mới, hoặc xóa vào khu vực chờ, hãy dùng lệnh sau:

```
git add .
```

Lệnh này sẽ tự động tìm và thêm tất cả các thay đổi trong thư mục dự án của bạn vào "giỏ hàng".

Cách 2: Chỉ đưa những thay đổi CỤ THỂ vào khu vực chờ

Nếu bạn chỉ muốn commit một vài file nhất định trong số rất nhiều file đã thay đổi, hãy chỉ định tên của chúng:

```
git add <tên-file-1> <đường-dẫn/tên-file-2>
```

Ví dụ: Bạn sửa 5 file nhưng chỉ muốn commit 2 file là `README.md` và `app.js`. Bạn sẽ dùng lệnh:

```
git add README.md src/app.js
```

Tóm lại: `git add` là lệnh để bạn *lựa chọn* những thay đổi nào sẽ được bao gồm trong lần commit tiếp theo. Đây là bước chuẩn bị quan trọng trước khi ghi lại lịch sử dự án.

5. Ghi lại lịch sử (Commit) - Tạo một "Điểm Lưu" cho dự án

Hãy tưởng tượng `commit` giống như việc bạn tạo một **điểm lưu (save point)** khi chơi game.

Sau khi bạn vượt qua một màn khó hoặc hoàn thành một nhiệm vụ, bạn sẽ lưu game lại. Với code cũng vậy, sau khi hoàn thành một tính năng hoặc sửa xong một lỗi, bạn sẽ tạo một `commit` để "lưu" lại trạng thái tốt đó của dự án một cách chính thức vào lịch sử.

Mỗi commit là một bản chụp (snapshot) của dự án tại một thời điểm.

Cách thực hiện

Sau khi bạn đã "bỏ các thay đổi vào giỏ hàng" bằng lệnh `git add`, bạn dùng lệnh sau để thực hiện commit:

```
git commit -m "Ghi chú cho lần commit này"
```

Phân tích lệnh:

- `git commit`: Đây là lệnh để thực hiện việc "lưu".
- `-m`: Là viết tắt của "message". Nó báo cho Git biết rằng bạn muốn đính kèm một tin nhắn ngay trên dòng lệnh.
- `"Ghi chú cho lần commit này"`: Đây là phần **cực kỳ quan trọng**. Nó là lời nhắn bạn để lại để giải thích cho lần lưu này bạn đã làm gì.

Cách Viết Ghi Chú Commit "Chuẩn" Như Lập Trình Viên Chuyên Nghiệp

Hãy nghĩ về commit message như một **tiêu đề email** hoặc một **dòng tít của bài báo**. Người đọc chỉ cần liếc qua là phải hiểu ngay nội dung chính. Viết commit message tốt không chỉ giúp bạn mà còn giúp cả đội làm việc hiệu quả hơn.

Đây là một quy tắc chung rất phổ biến gọi là "Conventional Commits".

1. Cấu trúc của một "Tiêu đề" Commit

Phần quan trọng nhất và thường dùng nhất là dòng đầu tiên, giống như tiêu đề email vậy. Nó có cấu trúc:

`loại(phạm_vi): mô tả ngắn gọn`

- **`loại`** **(Bắt buộc):** Cho biết bạn đã làm gì? (Sửa lỗi, thêm tính năng,...)
- **`phạm_vi`** **(Không bắt buộc):** Cho biết bạn thay đổi ở khu vực nào? (ví dụ: `login`, `thanh-toan`, `header`...)
- **`mô tả`** **(Bắt buộc):** Nói rõ hơn về thay đổi, viết ở thì hiện tại.

2. Các "Loại" thay đổi thường gặp (Giống như nhãn dán cho commit)

Hãy chọn một nhãn dán phù hợp nhất cho thay đổi của bạn:

- ✨ `feat`: **Thêm một tính năng mới** (ví dụ: thêm chức năng đăng nhập bằng Google).
- 🦎 `fix`: **Sửa một lỗi** (ví dụ: sửa lỗi nút bấm không hoạt động).
- 📚 `docs`: **Thay đổi về tài liệu** (ví dụ: viết thêm hướng dẫn, sửa comment trong code).
- 🎨 `style`: **Thay đổi về định dạng code** (sửa dấu chấm phẩy, thụt đầu dòng,... không ảnh hưởng đến logic).
- 🔨 `refactor`: **Tái cấu trúc code** (sắp xếp lại code cho gọn hơn nhưng không sửa lỗi hay thêm tính năng).
- 🚀 `perf`: **Cải thiện hiệu năng** (làm cho code chạy nhanh hơn).
- ✅ `test`: **Thêm hoặc sửa các bài kiểm thử (test)**.
- 🧹 `chore`: **Các việc vặt khác** (cấu hình, cài đặt công cụ, sửa file `.gitignore`...).

3. Ví dụ đơn giản

Hãy xem sự khác biệt:

Thay vì viết... 😡	Hãy viết... 👍
`git commit -m "xong"`	`git commit -m "feat(profile): Thêm chức năng thay đổi ảnh đại diện"`
`git commit -m "sửa lỗi"`	`git commit -m "fix(login): Sửa lỗi không đăng nhập được bằng email"`
`git commit -m "update readme"`	`git commit -m "docs: Cập nhật hướng dẫn cài đặt trên Windows"`

4. Khi nào cần viết thêm "Nội dung chi tiết"? (Tùy chọn)

Khi `mô tả ngắn` là không đủ để giải thích, bạn có thể thêm phần nội dung chi tiết, giống như viết nội dung cho email.

Quy tắc:

1. Viết dòng tiêu đề.
2. **Để trống một dòng.**
3. Viết giải thích chi tiết ở các dòng tiếp theo.

Ví dụ về commit của bạn, được trình bày lại cho dễ hiểu:

```
git commit -m "chore: Cấu hình Docker cho toàn bộ dự án

- Thêm Dockerfile cho backend (Flask) và frontend (React).
- Tạo file docker-compose để chạy môi trường dev và prod.
- Cấu hình .gitignore để bỏ qua các file không cần thiết.
- Tái cấu trúc thư mục để phù hợp với Docker."
```

Cách viết này giúp các công cụ tự động đọc commit (như khi tạo changelog) hiểu được đâu là tiêu đề, đâu là nội dung, làm cho lịch sử dự án của bạn trở nên cực kỳ sạch sẽ và chuyên nghiệp.

6. Đẩy nhánh lên GitHub (git push)

Hãy tưởng tượng bạn vừa viết xong một tài liệu Word trên máy tính của mình (giống như `git commit`). Bây giờ, bạn muốn **tải nó lên Google Drive** để đồng nghiệp có thể xem và làm việc cùng.

Hành động `git push` chính là việc "tải lên" đó. Nó sẽ gửi tất cả những commit (điểm lưu) mà bạn đã tạo ở máy bạn lên kho lưu trữ (repository) trên GitHub.

Cách thực hiện

Lệnh phổ biến nhất bạn sẽ dùng là:

Bash

```
git push origin <tên-nhánh-của-bạn>
```

Phân tích lệnh:

- `git push`: Lệnh để thực hiện việc đẩy code đi.

- `origin`: Đây là "biệt danh" mặc định cho kho lưu trữ gốc của bạn trên GitHub (nơi bạn đã clone code về). Cứ hiểu đơn giản nó là "máy chủ GitHub".
- `<tên-nhánh-của-bạn>`: Tên của nhánh mà bạn muốn đẩy lên.

Ví dụ:

- Để đẩy nhánh `main`: Bash
  ```
  git push origin main
  ```
- Để đẩy một nhánh tính năng có tên là `feature/them-chuc-nang-dang-nhap`: Bash
  ```
  git push origin feature/them-chuc-nang-dang-nhap
  ```

Trường hợp đặc biệt: Lần đầu tiên đẩy một nhánh mới

Khi bạn tạo một nhánh mới ở máy và đẩy nó lên GitHub lần đầu tiên, Git có thể sẽ gợi ý cho bạn một lệnh dài hơn một chút:

Bash

```
git push --set-upstream origin <tên-nhánh-của-bạn>
```

- **Đừng lo lắng!** Lệnh này chỉ đơn giản là làm 2 việc:
  1. **Đẩy code** của nhánh đó lên GitHub.
  2. **Tạo một liên kết** giữa nhánh ở máy bạn và nhánh mới được tạo trên GitHub.
- Bạn chỉ cần chạy lệnh này **một lần duy nhất** cho mỗi nhánh mới. Từ những lần sau, bạn chỉ cần dùng lệnh `git push` ngắn gọn là đủ.

Quy trình làm việc đầy đủ:

Đây là chuỗi hành động hoàn chỉnh từ lúc sửa code cho đến lúc đưa lên GitHub:

1. **Chọn thay đổi:**

Bash

```
git add .
```

2. **Lưu tại máy (tạo điểm lưu):**

```
git commit -m "feat: Thêm trang giới thiệu sản phẩm"
```

3. **Chia sẻ lên server:**

```
git push origin main
```

Tóm lại: `git push` là lệnh để bạn đăng tải và chia sẻ những commit đã lưu ở máy bạn lên kho chứa chung trên GitHub, giúp đồng bộ code và cho phép người khác thấy được công việc của bạn.