Git Guideline

Commit convention:

- Quy ước khi tạo message khi commit
  - **Sử dụng dạng mệnh lệnh trong subject.**

```
Good:
Add README.md ✅;

Bad:
Added README.md ❌;
Adding README.md ❌;
```

  - **Viết hoa chữ cái đầu tiên của subject.**

```
Good:
Add user authentication ✅;
Bad:
add user authentication ❌;
```

  - **Không kết thúc dòng subject bằng dấu chấm.**

```
Good
Update unit tests ✅;
Bad
Update unit tests. ❌;
```

  - **Giới hạn độ dài của subject không quá 50 ký tự , đảm bảo tính ngắn gọn và rõ ràng(nếu cần mô tả thêm thì sử dụng description).**
  - **Phần description không quá 72 ký tự và ngăn cách với subject bằng 1 dòng trống.**

    - **Nếu nội dung của commit có nhiều hơn 1 đoạn (paragraph), hãy sử dụng các dòng trống để phân tách chúng.**
    - **Nếu cần thiết, bạn có thể sử dụng bullet points thay cho paragraph.**
- Cấu trúc cơ bản của 1 commit có dạng như sau:

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

- Trong đó :
    - `<type>`: Phần bắt buộc, xác định loại thay đổi mà commit này thực hiện. Các giá trị thường gặp là:
        - `feat`: Khi thêm các tính năng mới;
        - `fix`: Sửa các lỗi của phần mềm;
        - `refactor`: Được sử dụng để thay đổi code nhưng vẫn đảm bảo chức năng tổng thể của nó.
        - `choro`: Các cập nhật không ảnh hưởng đến production code, liên quan đến việc điều chỉnh tool, cấu hình (config) hoặc thư viện.
        - `docs`: Bổ sung hoặc sửa đổi các tài liệu;
        - `perf`: Thay đổi code để nâng cao hiệu suất (performance);
        - `style`: Các điều chỉnh liên quan đến định dạng và khoảng trắng (cách trình bày của code);
        - `test`: Bao gồm hoặc sửa chữa các test;
        - `build`: Các sửa đổi ảnh hưởng đến hệ thống build hoặc các phần phụ thuộc bên ngoài;
        - `ci`: Thay đổi các tệp hoạc tập lệnh cấu hình CI;
        - `env`: Mô tả các điều chỉnh hoặc bổ sung cho tệp cấu hình trong quy trình CI, chẳng hạn như các tham số cấu hình vùng chứa (container)

- `[optional scope]`: Phạm vi tùy chọn, cung cấp thông tin cụ thể hơn về phần nào của codebase bị ảnh hưởng. Phạm vi được đặt trong dấu ngoặc đơn `()`, ví dụ: `feat(parser):`.
- `<description>`: Phần bắt buộc, là một mô tả ngắn gọn, rõ ràng về thay đổi mà commit này thực hiện. Mô tả phải theo sau dấu hai chấm `:` và một khoảng trắng.
- `[optional body]`: **l**à phần tùy chọn, cung cấp thêm thông tin chi tiết về thay đổi. Body nên bắt đầu sau một dòng trống kể từ dòng mô tả. Body có thể chứa nhiều đoạn văn và có thể bao gồm bất kỳ thông tin nào giúp người đọc hiểu rõ hơn về commit.
- `[optional footer(s)]`:
  - là phần tùy chọn, chứa các thông tin bổ sung như các tham chiếu đến các issue, người review, hoặc mô tả chi tiết về thay đổi phá vỡ (breaking changes).
  - **Footers** thường bắt đầu sau một dòng trống kể từ dòng cuối cùng của body. Mỗi footer bao gồm một token và một giá trị, ví dụ: `Reviewed-by: Z`.
  - Một loại footer đặc biệt là `BREAKING CHANGE:` dùng để mô tả các thay đổi API. Nó có thể đứng độc lập hoặc đi kèm với một dấu chấm than `!` trong phần

  `<type>[optional scope]`.

Branch convention:

- **Code Flow Branches**
  - **Development (dev)**: Tất cả các tính năng mới và sửa lỗi đều được đưa vào nhánh này.
  - **QA/Test (test)**: Chứa tất cả các mã đã sẵn sàng cho kiểm thử QA.
  - **Main (main)**: Nhánh production, nhánh mặc định được trình bày nếu repository được deploy.
- **Quy ước**

1. **Viết thường (Lowercase)**: không sử dụng chữ in hoa trong tên nhánh, chỉ dùng chữ thường(trừ trường hợp tên nhánh đặt theo ký tự viết tắt theo ID của ticket/task).
2. **Sử dụng gạch nối (Hyphen separated)**: nếu tên nhánh nhiều hơn 1 từ, hãy phân tách chúng bằng dấu gạch nối tuân theo nguyên tắc kebab-case. Tránh việc sử dụng PascalCase, CamelCase hoặc Snake_case trong khi tạo tên nhánh.
3. **Chỉ sử dụng ký tự a-z, 0-9**: Chỉ sử dụng các ký tự chữ và số cùng với dấu gạch nối trong tên nhánh của bạn. Tránh mọi ký tự không phải là chữ và số.
4. **Không sử dụng dấu gạch nối liên tục (–)**: Nếu bạn có các loại nhánh (branch type) thì bạn nên sử dụng dạng tiền tố (prefix) và dấu gạch chéo (/) để ngăn cách. Ví dụ: feature/feature-1, bugfix/bugfix-1, hotfix/hotfix-1,...
5. **Tránh việc kết thúc tên nhánh bằng dấu gạch nối**: Nó không có ý nghĩ vì dấu gạch nối được dùng để ngăn cách các từ và không có từ nào ở cuối cần để phân tách.
6. **Sử dụng các tên mô tả, ngắn gọn và rõ ràng** để giải thích những gì được thực hiện trên nhánh của bạn.

```
Bad branch names:
fixSidebar
feature-new-sidebar-
FeatureNewSidebar
feat_add_sidebar

Good branch names:

bugfix/sidebar
feature/new-sidebar
hotfix/interval-query-param-on-get-historical-data
```

- **Tiền tố quy ước tên nhánh (Branch Names Convention Prefixes)**

- **feature**: tiền tố này được sử dụng để truyền tải một tính năng mới sẽ được phát triển. Ví dụ: `feature/GiangDQ_add-filters`;
- **release**: được sử dụng để chuẩn bị cho 1 bản phát hành(release) mới. Tiền tố `release/` thường được sử dụng để thực hiện các tác vụ cuối cùng trước khi thực hiện hợp nhất(merging) các bản cập nhật mới nhất từ nhánh chính(master) để tạo 1 bản release. Ví dụ: `release/GiangDQ_v1.0.0.1-beta`;
- **bugfix**: được sử dụng để truyền tải thông điệp bạn đang giải quyết 1 lỗi trong code và lỗi này thường liên quan đến 1 vấn đề hoặc chức năng nào đó. Ví dụ: `bugfix/GiangDQ_sign-in-flow`;
- **hotfix**: tương tự như bugfix, nhưng nó liên quan đến việc sửa một lỗi nghiêm trọng trên môi trường production. Ví dụ: `hotfix/GiangDQ_cors-error`;
- **docs**: được sử dụng trong trường hợp bạn cần viết một số tài liệu. Ví dụ: `docs/GiangDQ_quick-start`;

SourceTree:

1. Quy ước phân nhánh:

- Sử dụng 2 nhánh chính:
  - `master`: Chứa code ổn định, sẵn sàng triển khai.
  - `dev`: Chứa code mới nhất đang được phát triển.
- Tạo nhánh con từ `dev` cho mỗi tính năng mới: `git checkout -b <tên nhánh con> dev`.
- Sử dụng tên nhánh rõ ràng và mô tả ngắn gọn.

2. Phân chia công việc:

- Mỗi người làm việc trên 1 nhánh con riêng.
- Chia nhỏ tính năng lớn thành các nhánh con nhỏ hơn nếu có nhiều người tham gia.

3. Chuẩn bị merge vào dev:

- **Trường hợp 1:** `dev` không có thay đổi mới:
  - `git merge <tên nhánh con> dev`.
- **Trường hợp 2:** `dev` có thay đổi mới:

    - `git pull` để cập nhật `dev`.
    - `git checkout <tên nhánh con>`.
    - `git rebase dev`.
        - Giải quyết xung đột code nếu có.
        - `git rebase --continue` để tiếp tục rebase.
        - Hoặc `git rebase --abort` để hủy rebase.
    - `git push -f origin <tên nhánh con>`.
- Sử dụng `git log` để kiểm tra lịch sử commit.

4. Merge vào dev:

- **Cách 1: merge: (Ưu tiên xài cách này)**
    - `git checkout dev`.
    - `git merge <tên nhánh con>`.
    - **Ưu điểm:** Đơn giản, nhanh chóng.
    - **Nhược điểm:** History git phức tạp, khó theo dõi.
- **Cách 2: rebase, squash và merge:**
    - `git checkout <tên nhánh con>`.
    - `git rebase dev`.
    - `git squash -u <commit-hash>`.
        - Sửa đổi commit message cho phù hợp.
        - `git rebase --continue` để tiếp tục rebase.
    - `git push -f origin <tên nhánh con>`.
    - `git checkout dev`.
    - `git merge <tên nhánh con>`.
    - **Ưu điểm:** History git gọn gàng, dễ theo dõi.
    - **Nhược điểm:** Phức tạp hơn cách 1, cần lưu ý sửa commit message.

5. Merge vào master:

- Sau khi hoàn thiện code, merge vào `master` tương tự như merge vào `dev`.

6. Hotfix:

- Tạo nhánh con từ `master`: `git checkout -b <tên nhánh> master`.
- Sửa lỗi.
- `git merge <tên nhánh> --no-ff master dev`.
- Push lên cả `master` và `dev`.