Git / GitHub Policy

1 General

Git history should be kept semi-linear. Ideally, branches should be rebased before being merged. A simple merge commit is also accepted, as long as branch do not intertwine (i.e. merging A $\rightarrow$ B and B $\rightarrow$ A multiple times).

Force pushes are disallowed on shared branches unless discussed beforehand with affected people.

2 Branches

Branch names should be prefixed by a type, for example:

- `feat/model-preview`: implements a new feature
- `fix/color-inversion`: fixes a bug or issue
- `refactor/migrate-to-postgres`: modifies the code structure without adding new features

If the branch is linked to an existing issue or task, the branch name should contain its reference. For example, `feat/JIRA-42-path-simplification`.

3 Pull Requests

Branches must be merged into the `main` branch through a pull request. Pull requests should be kept relatively short and must be reviewed by at least one other person. They must have a meaningful title, summarizing the features or fixes it contains, and can have a more in depth description to help reviewers understand the contents. If a pull request is related to an existing issue or task, its title should contain its reference. For example, `JIRA-42: Path geometry simplification`. A branch that has been merged must be deleted.

4 Commits

4.1 Content

A commit should be an atomic change targeting only 1 specific subject.

Build artifacts, cache directories, large files, personal configuration, environment variables and secrets must not be committed. If possible, such files should be added to `.gitignore` to avoid accidentally committing them. External libraries, especially large packages, should either not be included in the repository (with installation steps), or integrated as Git submodules[1].

4.2 Message

Commit messages must follow the conventional commit format[2]. In short, commit messages must be structured as follows:

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

[optional body]

[optional footer(s)]
```

Listing 1: Conventional commit message format

[1] https://github.blog/open-source/git/working-with-submodules/
[2] https://www.conventionalcommits.org/en/v1.0.0/

The `type` element describes the content of the commit. The following values are accepted:

- `fix`: patches a bug / issue
- `feat`: introduces a new feature
- `doc` or `docs`: only affects documentation
- `chore`: small changes to the repository which does not directly affect code or documentation
- `ci`: modifies CI/CD workflow configuration
- `refactor`: modifies existing code without changing the results / features

Some other types can be accepted but the above list should suffice in most cases.

The description (or subject) should be kept short. An optional body can be added to provide more details. It should be written in the imperative present tense:

- ✅ `add database schema`
- ❌ `changed path resolution`
- ❌ `fixes rounding error`

The body can contain special keywords to indicate that it completes a specific issue or task, for example:

- `closes JIRA-123`
- `fixes #4`
- `...`

4.3 Authors

In cases where a commit is written by multiple people, footers can be added to indicate co-authors:

```
1 feat: add segment data structures
2
3 Co-Authored-By: John Doe <john.doe@example.com>
```

Listing 2: Simple co-author footer

If a commit is partly written by an LLM, a footer must be added to name it as a co-author, for example:

```
1 feat: add segment data structures
2
3 Co-Authored-By: Claude <noreply@anthropic.com>
```

Listing 3: LLM co-author footer