Contributing Guideline

This document describes the contribution guidelines and rules for Git repositories.

Starting from Scratch

When starting a new project, we also initialize a new repository.

1. Run `git init -b develop`
2. Ensure your author name and email is set with

   ```
   git config user.name "my name"
   git config user.email "my@email.com"
   ```

3. Disable fast-forward merges:

   ```
   git config branch.main.mergeoptions "--no-ff"
   git config branch.develop.mergeoptions "--no-ff"
   ```

4. The following files must be added to every new repository:
   - `README.md` - it includes the following information:
     - Succinct description of the purpose of the code that lies in this repository
     - Development Setup
     - A reference to this contributing guidelines document (Mentioning it is enough)
   - `CHANGELOG.md` - at first this is empty, but at every release the new version and the changes need to be listed
5. The first commit message should be "initial commit" and contains the above-mentioned files. From then on, the Commit Message Guidelines must be followed.

Branching Model

There are three types of branches:

- `main` branch: It contains potentially releasable and tested software. There are no commits made on `main` except merges from `develop`. Each merge commit receives a Git Tag with the respective version.
- `develop` branch: This is the integration branch. Once it has been decided that a release is ready, this is merged to `main` with a merge commit. There are no commits made on `develop` except merges from feature branches and updating `CHANGELOG.md`.
- Feature branches: These are named `feature/<descriptive-feature-name>` and are created from the `develop` branch. Once a feature is finished and stable, it is merged into `develop` with a merge commit. The name of the feature branch should be descriptive. Words are separated by `-`. The complete branch name should not be longer than 80 characters.

Versioning

If the repository contains a library, package, etc. or similar, then the versioning scheme must follow SemVer (Semantic Versioning).

If the repository contains a deployable product, the versioning scheme is as follows:

```
<yyyy>-<mm>-<dd>.<i>
                  |
                  └─≫ Incrementing number for releases on the same day. Starts with '0'.
```

Regardless of versioning scheme, this version must be mentioned in the `CHANGELOG.md` and set as a Git tag on the `main` branch.

Commit Message Guidelines

We have very precise rules over how our Git commit messages must be formatted. This format leads to easier to read commit history.

Each commit message consists of a **header** and a **body**.

```
<header>
<BLANK LINE>
<body>
```

The `header` is mandatory and must conform to the Commit Message Header format.

The `body` is optional. When the body is present it must be at least 20 characters long and must conform to the Commit Message Body format.

Any line of the commit message cannot be longer than 100 characters.

Commit Message Header

```
<type>: <short summary>
  │      │
  │      └⫸ Summary in present tense. Not capitalized. No period at the end.
  │
  └⫸ Commit Type: build|docs|feat|fix|refactor|test
```

The `<type>` and `<summary>` fields are mandatory.

Type

Must be one of the following:

- **build**: Changes that affect the build system or external dependencies
- **docs**: Documentation only changes
- **feat**: A new feature
- **fix**: A bug fix
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **test**: Adding missing tests or correcting existing tests

Summary

Use the summary field to provide a succinct description of the change:

- use the imperative, present tense: "change" not "changed" nor "changes"
- don't capitalize the first letter
- no dot (.) at the end

Commit Message Body

Just as in the summary, use the imperative, present tense: "fix" not "fixed" nor "fixes".

Explain the motivation for the change in the commit message body. This commit message should explain *why* you are making the change. You can include a comparison of the previous behavior with the new behavior in order to illustrate the impact of the change.

Merge Commits

A merge commit does not have to follow the above guidelines. A merge commit has the default content Git already generates:

```
Merge branch '<source branch>'
```