06. Commit Message Template

```
[HEADER]  <Commit Type> <Scope (files affected by the change)>: <short summary>

[Commit Message Body] Issue, Change, Motive, Comparison

[Commit Message Footer] Reference
```

▼ **Header —**

- The **Type** and **summary** fields are mandatory
- Types may include —

> **project:** a new project is initiated
>
> > **crt:** a new file is created
> >
> > > **feat:** a new feature
> >
> > > **perf:** a code change that improves performance
> >
> > > **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

rename: renamed a file/folder

chore: regular code maintenance

docs: documentation only changes

ref: a new reference/tutorial like file is initiated

sec: a new section of code

subsec: a subsection inside of a section

perf: a code change that improves performance

> **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
>
> **rename:** renamed a file/folder
>
> **chore:** regular code maintenance
>
> **docs:** documentation only changes

- The **scope** field is optional if only one file was changed or, the changes are not that significant (chore, refactor, docs etc).
  - Name only those files whose changes are significant
  - Use relative path (no need to define the parent directory)
  - If filenames are large, write only enough characters to distinctly identify which file was changed

  Note: You can definitely use, `git log --name-status` to see which files were affected by a commit but defining **scope** is an easy way to quickly have an idea about the files that were changed significantly. So if you don't want you can skip **scope** altogether.
- **Summary in the present tense. Capitalized. No period in the end.**

▼ **Commit Message Body —**

In the **Body** of the message,

- 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.
- You can include a comparison of the previous behavior with the new behavior in order to illustrate the impact of the change.
- Do not assume the reviewer understands what the original problem was, ensure you add it.
- Do not think your code is self-explanatory (well try to write self explanatory code nevertheless).

▼ **Commit Message Footer —**

- The footer is the place to reference GitHub issues, Jira tickets, and other PRs that this commit closes or is related to.
- If the commit reverts to a previous commit, it should begin with **revert: followed by the header of the reverted commit**. The content of the commit message body should contain —
    - Information about the SHA of the commit being reverted in the following format: This reverts commit <SHA> and a clear description of the reason for reverting the commit.

Things to keep in mind

1. Use “git init” in a project folder that you know for sure has the correct name otherwise rename the folder before initializing a git project.
2. Reformat your code every time you save the code. use a consistent format style.
3. Commit only when you are sure that the changes you have made are of significant nature. otherwise, there's no need to clog up the commit history with "fixed a typo" like commit messages.
4. Follow a definite commit message convention. Tip: you can use git hooks to remind you of the convention.
5. Remember to regularly push your code to the cloud once you feel that the changes are significant but be very careful before you push your changes upstream unless you want to frequently encounter merging errors.