



Git Commit Message Standard

This document outlines the **best practices** and **commit message format** that our team must follow when making commits in our project repository. Following a clear and consistent commit message standard improves **readability**, **collaboration**, and **project history tracking**.

1. Commit Message Structure

The standard commit message format is:

```
<type>(scope): <description>
```

```
[body]
```

```
[footer]
```

Explanation of the Structure:

Element	Description	Mandatory/Optional
<code><type></code>	Specifies the type of change being made	✅ Mandatory
<code>(scope)</code>	Describes what part of the project is affected	? Optional
<code><description></code>	A short summary of the change	✅ Mandatory
<code>[body]</code>	Detailed explanation of the change	? Optional
<code>[footer]</code>	Reference to related issues or breaking changes	? Optional

2. Commit Types

Use the appropriate commit type from the table below:

Type	Description	Example
feat	Introducing a new feature	feat(auth): add login functionality
fix	Fixing a bug	fix(profile): resolve crash on update
docs	Updating documentation	docs(readme): update installation guide
style	Code formatting changes (no logic changes)	style(button): adjust padding
refactor	Code refactoring without feature changes	refactor(api): simplify response handling
perf	Performance improvement changes	perf(database): optimize query speed
test	Adding or updating tests	test(api): add unit tests for auth
build	Changes to build system or dependencies	build(deps): upgrade to Node v18
ci	Changes to CI/CD configuration	ci(github-actions): update workflows
chore	Other changes that don't modify code or tests	chore: update package-lock.json
revert	Reverting a previous commit	revert: revert feat(auth) login implementation

3. Writing Effective Descriptions

✓ Best Practices:

1. Use imperative mood:



Write your commit messages as if giving instructions:

- ✓ Good: add new feature to login page
- ✗ Bad: added new feature to login page

2. Keep it short and concise:

The **description** should be **under 50 characters**.

3. Avoid unnecessary punctuation:

-  Good: `fix(api): resolve timeout issue`
-  Bad: `fix(api): resolve timeout issue.`

4. Add a blank line before the body:

If you add a body, leave a **blank line** between the description and the body.

4. Writing the Body (Optional)

The body should answer **what the change is** and **why it's needed**, but **not how** it was done. Keep it concise.

Example:

```
fix(auth): resolve missing token bug
```

The bug caused users to be logged out unexpectedly.
This fix ensures tokens are refreshed correctly.

5. Footer (Optional)

Use the footer for:

- **References to related issues:** `Closes #1234` , `Refs #5678`
- **Breaking changes:**
If your change introduces a breaking change, mention it in the footer:
`BREAKING CHANGE: Removes support for legacy API`

Example:





```
feat(api): add new endpoint for user profiles
```

```
BREAKING CHANGE: Removes /v1/user endpoint
```

6. Example Commit Messages

Commit Type	Example Message
feat	feat(profile): add profile picture upload
fix	fix(auth): resolve session expiration issue
docs	docs(readme): update project setup instructions
style	style(navbar): fix alignment issues
refactor	refactor(db): improve query efficiency
perf	perf(api): reduce response time for GET requests
test	test(auth): add integration tests for login
build	build(deps): bump lodash from 4.17.20 to 4.17.21
ci	ci(github-actions): add code coverage report

7. Quick Tips for Writing Commit Messages

1.  **Use English for all commit messages** to maintain consistency.
2.  **Follow the format strictly** for consistency across all commits.
3.  **Link related issues** in the footer to maintain traceability.
4.  **Keep your commit messages meaningful and contextual.**

8. Common Mistakes to Avoid

Don't write vague messages:

- Bad: `update file`
- Good: `docs(readme): update contribution guide`

Don't use past tense:

- Bad: `fixed bug in login`
- Good: `fix(auth): resolve login error`

Don't skip commit messages:

Always explain **what** you changed and **why**.

9. Git Commit Message Template

Here's a **template** your team can use when committing:

```
<type>(<scope>): <short summary>
```

```
[Optional detailed explanation in the body]
```

```
[Optional footer with references or breaking changes]
```

10. Practical Example with Body and Footer

```
feat(auth): add password recovery feature
```

This feature allows users to recover their password via a secure email link. This improves user experience and security.

Closes #4567

BREAKING CHANGE: Removed legacy password reset method