How to Write Commit Messages

Applied Research Using the DOT Framework

1. What - Domain Research

1.1. Application Domain

Git commit messages are the textual descriptions developers attach to changes in source code. They are crucial for team communication, historical traceability, debugging, and project maintenance. Poor commit messages make understanding code history harder for collaborators and for yourself in the future.

Free Code Camp

Problem in Context

Many projects contain ambiguous messages such as “stuff”, “fix”, or “work” — which reduce clarity and impede collaborative workflows and automated tooling.

Pull Checklist

1.2. Available Work - Existing Knowledle

There are existing conventions and best practices documented by developer communities and standards bodies:

Conventional Commits

Presents a structured format for messages that helps tools and humans reason about changes.

Conventional Commits

Common Commit Guidelines

Header/body/footer formats improve readability and repository health.

Git Kraken

Best-practice articles

Emphasize clarity, conciseness, and useful context.

Baeldung

These constitute well-established knowledge that any guideline should incorporate.

1.3. Innovation Domain

This document synthesizes these existing conventions and real practices into an applied guideline tailored to most developer workflows, bridging the theoretical conventions and real-world use.

2. Why - Purpose & Trade-offs

2.1. Research Purpose

The goal of this applied research is to create a practical and validated guideline on how to write Git commit messages that improve clarity, team communication, and maintainability without being overly cumbersome to developers.

2.2. Trade-offs

Writing good commit messages often competes with developer efficiency. Too strict a format can slow down fast-moving teams; too loose a guideline can yield low-quality history. Balancing clarity vs productivity and consistency vs flexibility is the primary trade-off addressed. This aligns with design research trade-offs where optimizing between context fit and existing standards is essential.

The DOT Framework

3. How - Research Strategies & Methods

The DOT Framework encourages mixing strategies to triangulate evidence and insight.

3.1. Library Research

Methods:

Review existing authoritative sources on commit message practices.

Findings:

- Use imperative mood ("Add", "Fix", etc.) for succinct summary.

Git Kraken

- Separate subject and body with blank lines.

Medium

- Conventional Commits formalize structured messages.

Conventional Commits

This establishes foundational best practices from community and tooling sources.

3.2. Field Observations

Methods:

Sample commit logs from open repositories (e.g., GitHub) and review developer patterns.

Observation:

Many messages are uninformative (e.g., "update", "fix stuff"), making history harder to trace. This mirrors findings that a high percentage of commits could be improved in quality.

arxiv

Insight:

There is a gap between formal guidelines in existing work and typical developer habits in practice.

3.3. Lab-style Validation

Method:

Apply drafted guidelines in small team settings or controlled Git logs to observe clarity gains (e.g., capture how easily reviewers understand changes).

Result:

Early evidence (empirical practice) indicates clearly structured messages aid reviewers and reduce misunderstandings (anecdotal from standard practice benchmarking).

4. Findings - What Good Commit Messages Look Like

4.1. Key Components

A strong commit message typically includes:

Header (Mandatory)
Structured as ():

Body (Optional):
Explains the why and impact of the change

Footer (Optional):
References issues or breaking changes.

4.2. Structural Rules

- Use imperative tense (“Fix bug”, not “Fixed bug”).
- Keep subject lines ≤ 50 characters when possible.
- Separate header and body with a blank line.
- Wrap body lines at about 72 characters.

These conventions ensure readability across tools and collaborative contexts.

5. Guideline - Recommended Format

Below is the practical commit format synthesized from the research:

<type>(): <brief summary>
<Blank Line>
<body explaining the why and impact>
<Blank Line>
<footer with references (if needed)>

Examples:

- feat(auth): add OAuth2 login support.
- fix(ui): correct button alignment on dashboard
- docs(readme): update usage instructions

6. Conclusion

Using the DOT framework allowed this research to anchor guidelines in:

- existing community standards (Library evidence)
- real developer behaviors and pain points (Field observations)
- preliminary validation of clarity improvements (Lab insights)

The result is a practical, evidenced guideline for writing Git commit messages that helps teams document their work clearly and consistently.