# GitHub Commits richtig schreiben

Jedes Mal, wenn du Änderungen an einem gemeinsamen Projekt vornimmst, solltest du eine Commit-Message schreiben, um deinem Team Bescheid zu sagen! So schreibst du die besten Commit-Messages!

| Commit-Typ | Bedeutung | Wann verwenden? |
|---|---|---|
| `feat` | **Neue Funktionalität** | Nutze diesen Typ für neue Features, wie z. B. das Hinzufügen einer neuen Anmeldeseite.<br>Beispiel: `feat: add login page` |
| `fix` | **Fehlerbehebung** | Nutze diesen Typ für Bugfixes, wie z. B. das Beheben von Problemen mit der Skalierung von Bildern auf mobilen Geräten.<br>Beispiel: `fix: resolve bug with image loading flicker` |
| `docs` | **Dokumentation** | Änderungen an der Dokumentation.<br>Beispiel: `docs: update README with new setup instructions` |
| `style` | **Code-Formatierung** | Änderungen, die nur das Format betreffen (z. B. Leerzeichen, Formatierungen), ohne Code-Änderungen.<br>Beispiel: `style: fix indentation in main.js` |
| `refactor` | **Code-Änderungen ohne Bugfix oder neue Funktion** | Refaktorisierung von Code, z. B. Vereinfachung oder Umstrukturierung.<br>Beispiel: `refactor: simplify login flow logic` |
| `perf` | **Code Performance Optimierung** | Änderungen zur Verbesserung der Leistung.<br>Beispiel: `perf: Code shortened to unter 400 lines` |
| `test` | **Tests** | Hinzufügen oder Ändern von Tests.<br>Beispiel: `test: add unit tests for login component` |

**Nicht vergessen!**

Diese Infos machen deine Commit-Nachricht klar und nachvollziehbar – für dich und dein Team, auch in Zukunft.

- ☐ Deskriptiver Titel
- ☐ Detaillierte Beschreibung (Was erwartet? Was passiert?)
- ☐ Schritte zum Reproduzieren (bei Fehlern)
- ☐ Screenshots oder Logs (falls hilfreich)
- ☐ Angabe von Umgebung (Browser, OS, etc.)
- ☐ Verwendung von Labels (z. B. Bug, Verbesserung)