Gute Commit-Nachrichten schreiben ✍

Eine Commit-Nachricht ist mehr als nur eine Pflichtübung – sie ist **Dokumentation für dein zukünftiges Ich** und für jeden, der jemals mit deinem Code arbeiten wird. Schlecht formulierte Nachrichten machen die Git-Historie unbrauchbar, während gute Nachrichten dir helfen, Änderungen schnell zu verstehen, Fehler zu finden und den Projektverlauf nachzuvollziehen.

Warum sind gute Commit-Nachrichten so wichtig?

Stell dir vor, du suchst in drei Monaten einen Bug und schaust in deine Git-Historie. Du siehst Folgendes:

```
fix
update
asdf
done
nochmal gefixt
```

Das hilft dir **überhaupt nicht**. Du musst jeden Commit einzeln öffnen und den Code-Diff analysieren, um zu verstehen, was passiert ist. Bei einer sauberen Historie hingegen siehst du sofort:

```
Benutzer-Login: Passwort-Validierung hinzugefügt
Kontaktformular: E-Mail-Versand bei leeren Feldern verhindert
Navigation: Dropdown-Menü schließt sich jetzt bei Klick außerhalb
```

Hier erkennst du **auf einen Blick**, was jeder Commit bewirkt – ohne den Code überhaupt anzuschauen.

Die wichtigsten Regeln für Commit-Nachrichten

1. Verwende eine aussagekräftige Betreffzeile

Die erste Zeile deiner Commit-Nachricht ist die **Betreffzeile** – sie wird in Listen, Logs und Übersichten angezeigt. Sie sollte:

- **Kurz und prägnant** sein (idealerweise maximal 50–72 Zeichen)
- **Im Imperativ** formuliert sein (als würdest du Git einen Befehl geben)
- **Das „Was" beschreiben**, nicht das „Wie"

> □□ **Tipp:** Vervollständige gedanklich den Satz: *„Wenn dieser Commit angewendet wird, wird er...*" – und dann kommt deine Betreffzeile.

2. Trenne Betreff und Beschreibung durch eine Leerzeile

Wenn du mehr Details hinzufügen möchtest, lässt du nach der Betreffzeile eine **leere Zeile** und schreibst dann einen ausführlicheren Text. Viele Tools (auch PhpStorm und GitHub) behandeln die erste Zeile speziell – sie wird als Überschrift angezeigt.

3. Erkläre das „Warum", nicht nur das „Was"

Der Code zeigt, *was* geändert wurde. Die Commit-Nachricht sollte erklären, *warum* du diese Änderung gemacht hast – besonders wenn es nicht offensichtlich ist.

4. Halte dich an eine konsistente Sprache

Entscheide dich für **eine Sprache** (Deutsch oder Englisch) und bleibe dabei. In der Open-Source-Welt und bei internationalen Teams ist Englisch Standard, aber für persönliche oder deutschsprachige Teamprojekte ist Deutsch völlig in Ordnung.

Beispiele: Schlechte vs. gute Commit-Nachrichten

❌ Schlechte Beispiele

Nachricht	Problem
`fix`	Was wurde gefixt? Wo? Warum?
`update`	Extrem nichtssagend – jeder Commit ist ein „Update"
`asdf` / `test` / `wip`	Keine Information, wirkt unprofessionell
`Änderungen gemacht`	Offensichtlich – aber *welche* Änderungen?
`Bug gefixt in der Datei login.php Zeile 47 wo die Variable falsch war`	Zu lang, zu viele Details, die sich ändern können
`Login`	Zu vage – was ist mit dem Login?

✅ Gute Beispiele

Nachricht	Warum gut?
`Passwort-Validierung bei Login hinzugefügt`	Klar, spezifisch, sagt was passiert
`404-Fehler bei nicht existierenden Produktseiten behoben`	Beschreibt das Problem und die Lösung
`Bootstrap 5 auf Version 5.3 aktualisiert`	Konkret und nachvollziehbar
`Kontaktformular: Pflichtfeld-Prüfung für E-Mail ergänzt`	Nutzt Präfix zur Kategorisierung
`Performance: Datenbankabfragen in Produktliste optimiert`	Gibt Kontext durch Kategorie

Eine bewährte Struktur für Commit-Nachrichten

Für einfache Änderungen reicht eine einzelne Zeile. Bei komplexeren Commits empfiehlt sich dieses Format:

```
Kurze Zusammenfassung im Imperativ (max. 50-72 Zeichen)

Optionaler ausführlicher Text, der erklärt:
- WARUM diese Änderung nötig war
- Was der Kontext ist
- Welche Entscheidungen getroffen wurden

Falls relevant: Referenzen zu Issues, Tickets o.ä.
```

Konkretes Beispiel:

```
Warenkorb: Mengenänderung aktualisiert jetzt den Gesamtpreis

Bisher wurde der Gesamtpreis im Warenkorb erst nach einem
Seiten-Reload aktualisiert, wenn der Nutzer die Menge eines
Produkts geändert hat. Das war verwirrend und führte zu
Support-Anfragen.

Die Lösung nutzt einen AJAX-Request, der bei jeder
Mengenänderung den neuen Preis vom Server holt und
das DOM aktualisiert.
```

Praktische Konventionen und Präfixe

Viele Teams nutzen **Präfixe**, um Commits zu kategorisieren. Das ist besonders hilfreich, wenn du später die Historie filterst oder durchsuchst. Hier eine einfache Variante, die auch für Solo-Projekte gut funktioniert:

Präfix	Verwendung	Beispiel
`Feature:`	Neue Funktionalität	`Feature: Benutzer können Profilbild hochladen`
`Fix:`	Fehlerbehebung	`Fix: Login-Button reagiert wieder auf mobilen Geräten`
`Refactor:`	Code-Verbesserung ohne neue Funktion	`Refactor: Datenbank-Klasse in separate Datei ausgelagert`
`Style:`	Optische Änderungen, Formatierung	`Style: Einheitliche Einrückung in allen PHP-Dateien`
`Docs:`	Dokumentation	`Docs: README um Installationsanleitung erweitert`
`Chore:`	Wartung, Dependencies	`Chore: Composer-Pakete aktualisiert`

> □□ **Hinweis:** Diese Präfixe sind *Konventionen*, keine festen Regeln. Wichtig ist, dass du **konsistent** bleibst – entweder immer Präfixe nutzen oder nie.

Commit-Nachrichten in PhpStorm schreiben

In PhpStorm hast du beim Commit-Dialog ein Textfeld für die Nachricht. Ein paar Tipps speziell für PhpStorm:

- **Drücke Enter für eine neue Zeile**, um Betreff und Beschreibung zu trennen
- PhpStorm zeigt dir eine **Warnung**, wenn deine Betreffzeile zu lang wird (über 72 Zeichen)
- Du kannst unter **Settings → Version Control → Commit** einstellen, dass PhpStorm dich erinnert, wenn die Nachricht leer oder sehr kurz ist
- Mit **Strg+Shift+K** (bzw. **Cmd+Shift+K** auf macOS) öffnest du direkt das Commit-Fenster

---

Zusammenfassung: Die goldenen Regeln □□

1. **Schreibe im Imperativ** – „Füge hinzu", nicht „Hinzugefügt" oder „Fügt hinzu"
2. **Halte die Betreffzeile kurz** – maximal 50–72 Zeichen

3. **Sei spezifisch** – „Login-Validierung" statt „Update"
4. **Erkläre das Warum** – besonders bei nicht offensichtlichen Änderungen
5. **Ein Commit = eine logische Änderung** – nicht drei verschiedene Dinge in einem Commit mischen
6. **Bleibe konsistent** – gleiche Sprache, gleiche Konventionen im ganzen Projekt

Eine gute Commit-Historie ist wie ein gut geführtes Logbuch: Sie erzählt die Geschichte deines Projekts und hilft dir, auch nach Monaten noch zu verstehen, warum du bestimmte Entscheidungen getroffen hast. □□

Revision #1
Created 2026-06-16 16:12:04 UTC by art10m
Updated 2026-06-16 16:13:38 UTC by art10m