Kapitel 4: Commits erstellen und verstehen

Einleitung: Commits sind das Herzstück von Git – sie sind die „Speicherpunkte" deines Projekts. Jeder Commit ist ein Schnappschuss deines gesamten Projekts zu einem bestimmten Zeitpunkt, versehen mit einer Nachricht, die beschreibt, was du geändert hast. Die Kunst besteht darin, sinnvolle Commits zu erstellen: nicht zu groß (damit sie übersichtlich bleiben), nicht zu klein (damit sie bedeutsam sind), und mit klaren Nachrichten versehen. In diesem Kapitel lernst du nicht nur, wie du Commits erstellst, sondern auch, wie du sie nachträglich korrigieren kannst, falls dir ein Fehler unterläuft. PhpStorm bietet dafür eine sehr komfortable Oberfläche, die den Prozess deutlich vereinfacht.

- Dein erster Commit in PhpStorm – Schritt für Schritt □□
- Gute Commit-Nachrichten schreiben ✍
- Die Git-Historie in PhpStorm anzeigen und verstehen □□
- Den letzten Commit in PhpStorm korrigieren □□

Dein erster Commit in PhpStorm – Schritt für Schritt □□

Du hast dein Projekt als Git-Repository initialisiert und möglicherweise schon eine `.gitignore`-Datei angelegt. Jetzt wird es Zeit, deinen **ersten Commit** zu machen – also einen Schnappschuss deines aktuellen Projektstands in der Git-Historie zu speichern.

Der Ablauf im Überblick

Bevor wir in die Details gehen, hier der grundsätzliche Prozess:

```
flowchart LR
    A["1. Dateien bearbeiten\nim Working Directory"] --> B["2. Dateien zur\nStaging Area hinzufuegen"]
    B --> C["3. Commit erstellen\nmit Nachricht"]
    C --> D["4. Aenderung ist\nin der Historie"]
```

In PhpStorm ist dieser Prozess besonders komfortabel, weil du alles in einer einzigen Oberfläche erledigen kannst.

Schritt-für-Schritt-Anleitung

1. **Das Commit-Fenster öffnen**
   Es gibt mehrere Wege, das Commit-Fenster zu öffnen:
   - **Tastenkürzel:** `Strg + K` (Windows/Linux) bzw. `Cmd + K` (macOS) – der schnellste Weg
   - **Menü:** Gehe zu **Git → Commit…**
   - **Toolbar:** Klicke auf das grüne Häkchen-Symbol in der oberen Symbolleiste

Es öffnet sich links (oder als separates Fenster, je nach Einstellung) das **Commit-Tool-Window**.

2. **Dateien für den Commit auswählen**
   Im Commit-Fenster siehst du eine Liste aller Dateien, die sich seit dem letzten Commit geändert haben. Diese sind in Kategorien unterteilt:
   - **Unversioned Files:** Neue Dateien, die Git noch nicht kennt (rot markiert)
   - **Modified Files:** Bereits getrackte Dateien, die du verändert hast (blau markiert)
   - **Deleted Files:** Dateien, die gelöscht wurden

   > □□ **Tipp:** Bei deinem *allerersten* Commit werden alle Dateien als „Unversioned" angezeigt, weil Git sie noch nie gesehen hat.

   So wählst du Dateien aus:
   - Setze ein **Häkchen** vor jede Datei, die du in diesen Commit aufnehmen möchtest
   - Um *alle* Dateien auszuwählen, klicke auf das Häkchen ganz oben bei „Changes" oder „Unversioned Files"
   - Du kannst auch einzelne Dateien gezielt ein- oder ausschließen

   PhpStorm zeigt dir bei jeder Datei eine **Vorschau der Änderungen** (Diff), wenn du sie anklickst. So kannst du nochmal prüfen, was genau du committen wirst.
3. **Eine aussagekräftige Commit-Nachricht schreiben**
   Unterhalb der Dateiliste findest du ein Textfeld mit dem Platzhalter *„Commit Message*".
   Hier beschreibst du, **was du geändert hast und warum**.
   Für deinen ersten Commit ist eine einfache Nachricht völlig in Ordnung, zum Beispiel:

   ```
   Initial commit: Projektstruktur angelegt
   ```

   oder etwas ausführlicher:

   ```
   Erstes Commit: Grundlegende PHP-Dateien und Konfiguration hinzugefügt
   ```

   > □□ **Best Practice:** Schreibe die Nachricht so, dass du (oder andere) in drei Monaten noch verstehst, was dieser Commit enthält. Mehr dazu später im Kurs bei den Best Practices.

4. **Optionale Einstellungen prüfen**
   Unter dem Nachrichtenfeld findest du einige Optionen, die du für den Anfang meist ignorieren kannst:
   - **Amend commit:** Fügt Änderungen zum *letzten* Commit hinzu (für Anfänger erstmal nicht relevant)
   - **Author:** Zeigt deinen konfigurierten Namen und E-Mail – sollte bereits korrekt sein
   - **Before Commit:** Hier kannst du automatische Prüfungen aktivieren (z. B. Code-Analyse, Reformatierung)

   Für deinen ersten Commit kannst du diese Optionen auf den Standardeinstellungen belassen.
5. **Den Commit abschließen**

Wenn du mit deiner Dateiauswahl und Nachricht zufrieden bist, klicke auf den **Commit**-Button unten im Fenster.

> ⚠ **Hinweis:** Du siehst möglicherweise auch einen Button **„Commit and Push"**. Dieser würde den Commit direkt auf einen Remote-Server (z. B. GitHub) hochladen. Da du noch kein Remote-Repository eingerichtet hast, wähle zunächst nur **„Commit"**.

PhpStorm zeigt dir kurz eine Bestätigung an, z. B.:

```
1 file committed: Initial commit: Projektstruktur angelegt
```

Nach dem Commit: Überprüfen, ob alles geklappt hat

Um sicherzugehen, dass dein Commit erfolgreich war, kannst du die **Git-Historie** in PhpStorm öffnen:

- Gehe zu **Git → Show Git Log** (oder klicke unten auf den Tab *„Git"* und dann auf *„Log"*)
- Du siehst jetzt eine Liste aller Commits – bei deinem ersten Commit natürlich nur einen einzigen Eintrag
- Klickst du auf den Commit, zeigt dir PhpStorm an, welche Dateien enthalten sind und was geändert wurde

---

Zusammenfassung des Prozesses

Schritt	Aktion	Tastenkürzel
Commit-Fenster öffnen	Git → Commit…	`Strg/Cmd + K`
Dateien auswählen	Häkchen setzen	–
Nachricht schreiben	Im Textfeld eingeben	–
Commit abschließen	Auf „Commit" klicken	–
Historie prüfen	Git → Show Git Log	`Alt + 9`

Was passiert eigentlich im Hintergrund? □□

Wenn du in PhpStorm auf „Commit" klickst, führt PhpStorm im Hintergrund zwei Git-Befehle aus:

```
git add <ausgewählte-dateien>
git commit -m "Deine Commit-Nachricht"
```

Der erste Befehl verschiebt die Dateien in die **Staging Area**, der zweite erstellt den eigentlichen **Commit** im Repository. PhpStorm fasst diese beiden Schritte für dich zusammen, sodass du sie nicht einzeln ausführen musst – aber es ist gut zu wissen, was „unter der Haube" passiert.

Du hast jetzt deinen ersten Commit erstellt! □□ Ab jetzt kannst du jederzeit weitere Änderungen machen und neue Commits erstellen – und dabei immer auf frühere Versionen zurückgreifen, falls nötig.

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. □□

Die Git-Historie in PhpStorm anzeigen und verstehen □□

Nachdem du einige Commits gemacht hast, möchtest du natürlich auch **nachschauen können**, was du wann geändert hast. PhpStorm bietet dafür ein mächtiges, visuell ansprechendes Werkzeug: das **Git Log**. Hier erfährst du, wie du es nutzt und die Informationen richtig interpretierst.

Das Git-Log-Fenster öffnen

Es gibt mehrere Wege, um zur Commit-Historie zu gelangen:

1. **Über das Menü:** Gehe zu **Git → Show Git Log** (oder in älteren Versionen **VCS → Git → Show History**).
2. **Über das Tool-Fenster:** Am unteren Rand von PhpStorm findest du den Reiter **Git**. Klicke darauf, und du siehst automatisch den **Log**-Tab.
3. **Tastenkombination:** Mit `Alt + 9` (Windows/Linux) bzw. `Cmd + 9` (macOS) öffnest du das Git-Tool-Fenster direkt.

Das Log-Fenster zeigt dir eine **chronologische Liste aller Commits** – die neuesten oben, die ältesten unten.

Die Ansicht verstehen

Das Git-Log in PhpStorm ist in mehrere Bereiche aufgeteilt:

```
flowchart TB
    subgraph LOG["Git Log Fenster"]
        A["Commit-Liste\nalle Commits chronologisch"] --> B["Commit-Details\nNachricht, Autor, Datum"]
        B --> C["Geaenderte Dateien\nListe der betroffenen Files"]
        C --> D["Diff-Ansicht\nkonkrete Aenderungen im Code"]
```

```
    end
```

Die Commit-Liste (linker/oberer Bereich)

Hier siehst du für jeden Commit:

- **Commit-Nachricht** – die erste Zeile deiner Nachricht (deshalb ist eine aussagekräftige erste Zeile so wichtig!)
- **Autor** – wer den Commit gemacht hat
- **Datum und Uhrzeit** – wann der Commit erstellt wurde
- **Commit-Hash** – eine eindeutige ID (z. B. `a3b8f2c`), mit der du den Commit referenzieren kannst
- **Branch-Tags** – farbige Labels zeigen an, zu welchem Branch ein Commit gehört

> □□ **Tipp:** Die grafische Darstellung links neben den Commits zeigt dir die **Branch-Struktur** – also wo Branches abzweigen und wieder zusammengeführt werden.

Die Commit-Details (mittlerer Bereich)

Wenn du einen Commit in der Liste anklickst, siehst du rechts daneben oder darunter:

- Die **vollständige Commit-Nachricht** (auch mehrzeilige)
- Den **vollständigen Commit-Hash**
- Informationen über den **Autor** und den **Committer** (können unterschiedlich sein)

Die geänderten Dateien

Unterhalb der Commit-Details findest du eine **Liste aller Dateien**, die in diesem Commit geändert wurden. Jede Datei hat ein Symbol, das die Art der Änderung anzeigt:

Symbol/Farbe	Bedeutung
□□ Grün / `+`	Datei wurde **neu hinzugefügt**
□□ Blau / `~`	Datei wurde **geändert**
□□ Rot / `-`	Datei wurde **gelöscht**
□□ Lila	Datei wurde **umbenannt** oder verschoben

Die konkreten Änderungen ansehen (Diff)

Das eigentlich Spannende ist natürlich: **Was genau wurde geändert?** Dafür nutzt du die Diff-Ansicht:

1. **Datei auswählen:** Klicke in der Liste der geänderten Dateien auf eine Datei.
2. **Diff öffnen:** Doppelklicke auf die Datei oder drücke `Strg + D` (Windows/Linux) bzw. `Cmd + D` (macOS).
3. **Änderungen lesen:** PhpStorm zeigt dir eine **Side-by-Side-Ansicht**:
   - **Links:** Der alte Stand (vor dem Commit)
   - **Rechts:** Der neue Stand (nach dem Commit)

   Gelöschte Zeilen sind rot markiert, hinzugefügte grün. So siehst du auf einen Blick, was sich geändert hat.

> **Tipp:** Du kannst die Ansicht auch auf „Unified" umstellen, wenn du lieber alles untereinander statt nebeneinander sehen möchtest. Das geht über das Zahnrad-Symbol in der Diff-Ansicht.

Nützliche Filter und Suchfunktionen

Bei größeren Projekten wird die Historie schnell lang. PhpStorm bietet dir praktische **Filter**, um den Überblick zu behalten:

- **Nach Text suchen:** Gib im Suchfeld einen Begriff ein, um Commits zu finden, deren Nachricht diesen Begriff enthält.
- **Nach Autor filtern:** Klicke auf das Filter-Symbol und wähle „User", um nur Commits eines bestimmten Autors zu sehen.
- **Nach Datum filtern:** Du kannst auch einen Zeitraum eingrenzen, z. B. „letzte Woche" oder ein bestimmtes Datum.
- **Nach Pfad filtern:** Besonders praktisch – du kannst die Historie auf eine **einzelne Datei oder einen Ordner** beschränken. Rechtsklicke dafür im Projektbaum auf eine Datei und wähle **Git → Show History for Selection**.

Historie einer einzelnen Datei anzeigen

Manchmal interessiert dich nicht die gesamte Projekt-Historie, sondern nur: **Was ist mit dieser einen Datei passiert?**

1. Rechtsklicke auf die Datei im Projektbaum.
2. Wähle **Git → Show History**.
3. Du siehst jetzt nur die Commits, die **diese Datei betreffen**.

Noch detaillierter wird es mit **Annotate** (auch „Blame" genannt):

1. Öffne die Datei im Editor.
2. Rechtsklicke in den linken Rand (wo die Zeilennummern stehen).
3. Wähle **Annotate with Git Blame**.

Jetzt siehst du **für jede Zeile**, in welchem Commit sie zuletzt geändert wurde, von wem und wann. Das ist unglaublich praktisch, um herauszufinden, wann und warum eine bestimmte Code-Stelle entstanden ist.

Praktisches Beispiel

Angenommen, du hast in den letzten Tagen mehrere Commits gemacht und möchtest nachschauen, wann du die Datei `database.php` zuletzt geändert hast:

1. Öffne das Git-Log mit `Alt + 9`.
2. Gib im Suchfeld oben `database.php` ein oder nutze den Pfad-Filter.
3. Du siehst jetzt alle Commits, die diese Datei betreffen.
4. Klicke auf einen Commit und dann doppelt auf `database.php` in der Dateiliste.
5. Die Diff-Ansicht zeigt dir genau, welche Zeilen du damals geändert hast.

Zusammenfassung

Aktion	Weg in PhpStorm
Gesamte Historie anzeigen	**Git → Show Git Log** oder `Alt + 9`

Aktion	Weg in PhpStorm
Änderungen eines Commits sehen	Commit anklicken → Datei doppelklicken
Historie einer Datei	Rechtsklick → **Git → Show History**
Wer hat welche Zeile geschrieben?	Rechtsklick im Editor → **Annotate with Git Blame**
Nach Commits suchen	Suchfeld im Log-Fenster nutzen

Mit diesen Werkzeugen hast du deine Projektgeschichte immer im Griff – du kannst jederzeit nachvollziehen, *was* sich *wann* und *warum* geändert hat. □□

Den letzten Commit in PhpStorm korrigieren □□

Es passiert jedem: Du hast gerade auf „Commit" geklickt und merkst sofort, dass du eine Datei vergessen hast oder sich ein peinlicher Tippfehler in die Commit-Nachricht eingeschlichen hat. Die gute Nachricht: Git bietet genau für diesen Fall eine elegante Lösung – das sogenannte **Amend** (englisch für „ändern" oder „verbessern"). Und PhpStorm macht es dir besonders einfach, diese Funktion zu nutzen.

Was bedeutet „Amend" eigentlich?

Wenn du einen Commit „amendest", **ersetzt** du den letzten Commit durch einen neuen. Dabei kannst du:

- **Dateien hinzufügen**, die du vergessen hattest
- **Dateien entfernen**, die versehentlich dabei waren
- **Die Commit-Nachricht ändern**
- **Änderungen an bereits enthaltenen Dateien ergänzen**

Wichtig zu verstehen: Der alte Commit wird nicht wirklich „bearbeitet", sondern durch einen **komplett neuen Commit ersetzt**. Für dich sieht es so aus, als hättest du den Fehler nie gemacht – die Historie bleibt sauber.

> ⚠ **Achtung:** Du solltest Amend nur verwenden, wenn du den Commit **noch nicht gepusht** hast. Sobald ein Commit auf einem Remote-Repository (z. B. GitHub) liegt und andere damit arbeiten könnten, kann das Ändern der Historie zu Problemen führen. Für lokale, noch nicht geteilte Commits ist Amend aber völlig unbedenklich.

Schritt-für-Schritt-Anleitung in PhpStorm

Fall 1: Du hast eine Datei vergessen

1. **Nimm die gewünschten Änderungen vor**
   Bearbeite die vergessene Datei oder füge sie dem Projekt hinzu – ganz normal, wie du es sonst auch tun würdest.
2. **Öffne das Commit-Fenster**
   Drücke `Ctrl + K` (Windows/Linux) bzw. `Cmd + K` (macOS), oder gehe über **Git → Commit…**
3. **Wähle die nachzureichenden Dateien aus**
   Im Commit-Fenster siehst du links die Liste der geänderten Dateien. Setze den Haken bei den Dateien, die du zum letzten Commit hinzufügen möchtest.
4. **Aktiviere die Amend-Option**
   Im Commit-Fenster findest du ein kleines Zahnrad-Symbol ⚙ oder direkt eine Checkbox mit der Beschriftung **„Amend commit"** (manchmal auch **„Amend"** genannt). Diese Option befindet sich typischerweise:
   - Rechts neben dem Eingabefeld für die Commit-Nachricht, oder
   - Im Dropdown-Menü des Commit-Buttons

   Sobald du „Amend" aktivierst, passiert etwas Praktisches: Die **Commit-Nachricht des letzten Commits** wird automatisch ins Textfeld geladen.
5. **Commit-Nachricht anpassen (optional)**
   Wenn die Nachricht bereits passt, lass sie so. Wenn du sie ändern möchtest, kannst du das jetzt tun.
6. **Commit abschließen**
   Klicke auf **Commit** (nicht auf „Commit and Push"!). Der letzte Commit wird nun durch den neuen, korrigierten Commit ersetzt.

Fall 2: Du möchtest nur die Commit-Nachricht ändern

Manchmal hast du keine Dateien vergessen, sondern nur einen Tippfehler in der Nachricht oder möchtest sie präziser formulieren. Das geht noch schneller:

1. **Öffne das Commit-Fenster** (`Ctrl + K` / `Cmd + K`)

2. **Aktiviere „Amend commit"**
   Die alte Nachricht erscheint im Textfeld.
3. **Korrigiere die Nachricht**
   Schreibe die Nachricht so, wie sie sein sollte.
4. **Stelle sicher, dass keine Dateien ausgewählt sind**
   Wenn du wirklich nur die Nachricht ändern willst, sollten keine neuen Dateien angehakt sein. PhpStorm wird dann nur die Nachricht aktualisieren.
5. **Klicke auf Commit**

Visueller Überblick

```
flowchart TD
    A["Letzter Commit ist fehlerhaft\n- Datei vergessen?\n- Nachricht falsch?"] -->
B["Aenderungen vornehmen\noder direkt Commit-Fenster oeffnen"]
    B --> C["Commit-Fenster oeffnen\nStrg+K / Cmd+K"]
    C --> D["Option 'Amend commit'\naktivieren"]
    D --> E{"Was korrigieren?"}
    E -->|Dateien nachreichen| F["Dateien auswaehlen\nund Commit"]
    E -->|Nur Nachricht| G["Nachricht korrigieren\nund Commit"]
    F --> H["Alter Commit wird ersetzt\nHistorie bleibt sauber"]
    G --> H
```

Wo finde ich die Amend-Option genau?

Je nach PhpStorm-Version kann die Oberfläche leicht variieren. Hier die typischen Stellen:

PhpStorm-Version	Position der Amend-Option
Neuere Versionen (2021+)	Direkt unter dem Nachrichtenfeld als Checkbox „Amend" oder im Zahnrad-Menü ⚙ rechts
Ältere Versionen	Im Dropdown-Pfeil neben dem „Commit"-Button → „Commit and Amend"

Wenn du die Option nicht sofort siehst, schau nach einem **Zahnrad-Symbol** oder einem **kleinen Pfeil** neben dem Commit-Button – dort verstecken sich oft erweiterte Optionen.

Was passiert technisch im Hintergrund?

Wenn du in der Kommandozeile arbeiten würdest, entspräche das Ganze diesem Befehl:

```
git commit --amend
```

Oder, wenn du auch Dateien hinzufügen möchtest:

```
git add vergessene_datei.php
git commit --amend
```

Git erstellt dabei einen **neuen Commit mit einer neuen ID** (dem sogenannten *Hash*), der den alten ersetzt. Der alte Commit existiert technisch noch kurz im Hintergrund, wird aber nicht mehr referenziert und irgendwann automatisch aufgeräumt.

Häufige Fragen und Stolperfallen

„Kann ich auch ältere Commits ändern, nicht nur den letzten?"

Ja, aber das ist deutlich komplizierter und erfordert einen sogenannten **Interactive Rebase**. Für den Anfang solltest du dich auf das Amenden des *letzten* Commits beschränken – das deckt 90 % der Fälle ab.

„Ich habe schon gepusht – was jetzt?"

Wenn du einen bereits gepushten Commit amendest und dann erneut pushen willst, wird Git sich beschweren, weil die Historien nicht mehr übereinstimmen. Du könntest einen **Force Push** machen (`git push --force`), aber das ist **gefährlich**, wenn andere mit dem Repository arbeiten. Bei persönlichen Projekten, an denen nur du arbeitest, ist es meist okay – aber sei vorsichtig.

„Was, wenn ich Amend aus Versehen aktiviert habe?"

Keine Panik! Solange du noch nicht auf „Commit" geklickt hast, kannst du die Checkbox einfach wieder deaktivieren. Falls du den Amend-Commit bereits gemacht hast, kannst du mit `git reflog` (im Terminal) den alten Commit wiederfinden – aber das ist ein fortgeschrittenes Thema.

Zusammenfassung

Situation	Lösung in PhpStorm
Datei im letzten Commit vergessen	Datei auswählen → „Amend commit" aktivieren → Commit
Commit-Nachricht korrigieren	„Amend commit" aktivieren → Nachricht ändern → Commit
Bereits gepushter Commit	⚠ Amend vermeiden oder Force Push (nur bei Solo-Projekten)

Das Amenden ist eines der nützlichsten Features für den Alltag – es hält deine Commit-Historie sauber und erspart dir peinliche „Datei vergessen"-Commits. Nutze es ruhig großzügig, solange du noch nicht gepusht hast! □□