IDE-Aktionen und CLI-Befehle zuordnen

Warum diese Zuordnung wichtig ist

PhpStorm übersetzt jede Schaltfläche, jeden Menüpunkt und jede Diff-Ansicht letztlich in einen oder mehrere Git-Befehle, die im Hintergrund ausgeführt werden. Wer nur die Oberfläche bedient, ohne die zugrunde liegenden Kommandos zu kennen, gerät bei Fehlermeldungen, ungewöhnlichen Repository-Zuständen oder in fremden Umgebungen ohne IDE schnell an eine Grenze. Wer umgekehrt nur die Kommandozeile beherrscht, verschenkt die visuellen Vorteile von PhpStorm bei Diffs, Konflikten und Historienübersicht.

Dieses Kapitel schafft daher eine bewusste *Brücke*: Für jede zentrale IDE-Aktion wird der entsprechende Git-Befehl benannt, damit du jederzeit zwischen beiden Werkzeugen wechseln kannst – je nachdem, was in der jeweiligen Situation effizienter oder transparenter ist.

Grundprinzip: PhpStorm ist eine Oberfläche über Git

PhpStorm ruft standardmäßig das auf deinem System installierte Git-Programm auf (siehe Abschnitt *„Git-Programm und Version prüfen“*). Es erzeugt keine eigene, abweichende Versionslogik. Das bedeutet:

- Jeder Commit über die Oberfläche entspricht exakt einem `git commit`.
- Jeder Branch-Wechsel entspricht einem `git switch` bzw. `git checkout`.
- Konflikte, Merges und Rebases folgen denselben internen Regeln wie auf der Kommandozeile.

Diese Deckungsgleichheit ist beabsichtigt und erlaubt es dir, ein und dasselbe Repository nahtlos mit beiden Werkzeugen zu bearbeiten – auch im Wechsel innerhalb eines Arbeitstages.

Zuordnungstabelle: Kernaktionen

Die folgende Übersicht zeigt die wichtigsten Aktionen aus PhpStorms VCS-Menü, dem *Commit*-Werkzeugfenster und dem *Git*-Werkzeugfenster mit ihren CLI-Entsprechungen.

PhpStorm-Aktion	Ort in der Oberfläche	Entsprechender Git-Befehl
Projekt unter Versionskontrolle stellen	*VCS → Enable Version Control Integration*	`git init`
Dateien vormerken	*Commit*-Fenster, Checkbox aktivieren	`git add <Datei>`
Commit erstellen	*Commit*-Fenster → *Commit*	`git commit -m "…"`
Commit und Push in einem Schritt	*Commit*-Fenster → *Commit and Push…*	`git commit -m "…"` gefolgt von `git push`
Letzten Commit ändern	*Commit*-Fenster → *Amend Commit*	`git commit --amend`
Änderungen ansehen	Doppelklick auf Datei im *Commit*-Fenster	`git diff`
Datei aus dem Commit ausschließen	Checkbox deaktivieren	`git restore --staged <Datei>`
Branch erstellen	*Git → Branches… → New Branch*	`git branch <Name>` oder `git checkout -b <Name>`
Branch wechseln	*Git → Branches…* → Branch auswählen → *Checkout*	`git switch <Name>`
Branches zusammenführen	*Git → Branches… → Merge into Current*	`git merge <Name>`
Rebase durchführen	*Git → Branches… → Rebase Current onto Selected*	`git rebase <Name>`
Änderungen abrufen	*Git → Fetch*	`git fetch`
Änderungen integrieren	*Git → Pull…*	`git pull`
Änderungen veröffentlichen	*Git → Push…*	`git push`
Arbeit zwischenlagern	*Git → Uncommitted Changes → Stash Changes…*	`git stash`
Stash anwenden	*Git → Uncommitted Changes → Unstash Changes…*	`git stash pop` bzw. `git stash apply`
Commit rückgängig machen	*Git → Uncommitted Changes* oder Log-Kontextmenü → *Revert Commit*	`git revert <Commit>`
Zu einem Commit zurücksetzen	Log-Kontextmenü → *Reset Current Branch to Here*	`git reset`

PhpStorm-Aktion	Ort in der Oberfläche	Entsprechender Git-Befehl
Datei-Historie ansehen	Rechtsklick auf Datei → *Git* → *Show History*	`git log --follow <Datei>`
Zeilenverantwortung prüfen	*Annotate*-Randleiste	`git blame <Datei>`
Repository klonen	*Get from VCS...*	`git clone <URL>`

Warum die Reihenfolge manchmal abweicht

Ein wichtiger Unterschied betrifft zusammengesetzte Aktionen. Der Button *Commit and Push* etwa führt intern zwei getrennte Git-Operationen aus, die auf der Kommandozeile ebenfalls zwei Befehle erfordern würden. PhpStorm bündelt dies lediglich komfortabel in einem Klick, ohne die eigentliche Git-Semantik zu verändern.

Ähnlich verhält es sich beim Branch-Wechsel mit gleichzeitigem Erstellen: *New Branch from Selected* entspricht `git checkout -b <Name> <Basis>` – ein einzelner CLI-Befehl, der in der IDE über einen kleinen Dialog abgebildet wird.

$$
\text{IDE-Aktion} ;=; \text{ein oder mehrere Git-Befehle in fester Reihenfolge}
$$

Entscheidungshilfe: IDE oder Kommandozeile?

```
flowchart TD
    A["Aufgabe steht an"] --> B{"Visuelle Übersicht nötig?"}
    B -->|Ja, z.B. Diff, Konflikt, Branch-Graph| C["PhpStorm verwenden"]
    B -->|Nein, einfacher Einzelbefehl| D{"Reproduzierbarkeit oder Skript nötig?"}
    D -->|Ja, z.B. CI, Automatisierung| E["Kommandozeile verwenden"]
    D -->|Nein, spontane Aktion| C

    style C fill:#2f6f4f,color:#ffffff
```

```
    style E fill:#37474f,color:#ffffff
```

Als grobe Faustregel gilt:

- **PhpStorm** eignet sich besonders für Diffs, Merge-Konflikte, Historienvergleiche und alles, was von einer visuellen Darstellung profitiert.
- **Die Kommandozeile** eignet sich für seltene, präzise Spezialbefehle, Automatisierung, Skripte und Situationen, in denen du exakt nachvollziehen musst, was passiert – etwa bei der Fehlersuche.

Praktischer Tipp: Befehle in PhpStorm sichtbar machen

PhpStorm protokolliert die tatsächlich ausgeführten Git-Befehle im Werkzeugfenster **Version Control → Console**. Diese Ansicht ist besonders lehrreich: Jede Aktion, die du über Menüs auslöst, erscheint dort als exakter Befehl inklusive Parameter.

> **Empfehlung:** Aktiviere die *Console*-Registerkarte dauerhaft während der ersten Wochen mit PhpStorm. So verinnerlichst du die Zuordnung zwischen Oberfläche und Kommandozeile automatisch – ganz ohne Auswendiglernen.

Diese bewusste Verknüpfung beider Arbeitsweisen bildet die Grundlage für alle folgenden Kapitel, in denen Befehle und IDE-Funktionen gleichrangig und wechselseitig ergänzend eingesetzt werden.

Revision #1
Created 2026-07-26 17:09:34 UTC by art10m
Updated 2026-07-26 17:09:46 UTC by art10m