Integriertes Terminal einrichten

Warum ein eigenes Git-Terminal in PhpStorm sinnvoll ist

Die grafischen Werkzeuge von PhpStorm decken den überwiegenden Teil des täglichen Git-Workflows ab – Commit-Dialog, Diff-Ansicht, Branch-Menü. Doch bestimmte Aufgaben lassen sich nur oder deutlich effizienter über die Kommandozeile lösen: interaktives Rebasing, komplexe `git log`-Filter, Plumbing-Befehle oder das schnelle Ausprobieren eines Befehls, bevor er zur Gewohnheit wird. Ein korrekt konfiguriertes *integriertes Terminal* verbindet beide Welten, ohne dass du zwischen Fenstern wechseln musst.

Unter Windows 11 ist diese Konfiguration nicht trivial, da mehrere Shells parallel existieren – **Git Bash**, **PowerShell** und **CMD** – und jede ihre eigenen Stärken sowie Eigenheiten bei Zeilenenden, Pfaden und Zeichenkodierung mitbringt.

Terminal-Einstellungen öffnen

Die Konfiguration findest du unter:

```
Datei → Einstellungen → Werkzeuge → Terminal
```

Hier legst du fest, welche Shell standardmäßig gestartet wird, in welchem Verzeichnis das Terminal öffnet und wie es sich optisch sowie funktional verhält.

Die richtige Shell auswählen

Für die Arbeit mit Git unter Windows kommen drei Kandidaten in Frage. Die Wahl beeinflusst, welche Befehle nativ funktionieren und wie Pfade interpretiert werden.

Shell	Git-Integration	Besonderheiten	Empfehlung
Git Bash	Nativ, POSIX-kompatibel	Unix-Befehle wie `grep`, `sed`, `ls -la` verfügbar	*Standardwahl* für Git-Arbeit
PowerShell	Über Git for Windows nutzbar	Objektorientierte Pipeline, gute Skriptfähigkeiten	Für Automatisierung/Skripte
CMD	Funktional, aber eingeschränkt	Keine moderne Syntax, wenig Komfort	Nur bei Altlasten nötig

Für nahezu alle Aufgaben in diesem Kurs ist **Git Bash** die sinnvollste Voreinstellung, da sie exakt dem Verhalten entspricht, das auch auf Linux- und macOS-Systemen üblich ist – ein klarer Vorteil, wenn du Anleitungen aus der offiziellen Git-Dokumentation eins zu eins übernehmen möchtest.

```
flowchart TD
    A["Aufgabe waehlen"] --> B{"Reine Git Befehle?"}
    B -->|Ja| C["Git Bash verwenden"]
    B -->|Nein| D{"Windows Automatisierung?"}
    D -->|Ja| E["PowerShell verwenden"]
    D -->|Nein| F["CMD nur bei Bedarf"]

    style C fill:#2e7d32,color:#ffffff
    style E fill:#1565c0,color:#ffffff
    style F fill:#616161,color:#ffffff
```

Shell-Pfad korrekt hinterlegen

Damit PhpStorm Git Bash findet, muss der Pfad zur ausführbaren Datei exakt stimmen. Nach einer Standardinstallation von *Git for Windows* lautet er üblicherweise:

```
C:\Program Files\Git\bin\bash.exe
```

Trage diesen Pfad im Feld **Shell-Pfad** ein. Alternativ kannst du über die Schaltfläche mit dem Ordnersymbol direkt zur Datei navigieren, was Tippfehler vermeidet. Sollte PhpStorm die Shell nicht automatisch vorschlagen, ist dies meist ein Hinweis darauf, dass die Installation nicht in der PATH-Variable registriert wurde – ein Punkt, der bereits im vorherigen Kapitel geprüft wurde.

Optionale Startparameter kannst du im Feld *Shell-Pfad* direkt anhängen, etwa `--login -i`, um eine interaktive Login-Shell zu erzwingen und damit sicherzustellen, dass alle Umgebungsvariablen

aus der `.bashrc` geladen werden.

Startverzeichnis festlegen

Standardmäßig öffnet PhpStorm das Terminal im Stammverzeichnis des aktuellen Projekts. Das ist in der Regel die gewünschte Einstellung, da du sofort im richtigen Repository-Kontext arbeitest. Über die Option **Startverzeichnis** lässt sich dies bei Bedarf überschreiben – sinnvoll etwa bei Monorepo-Strukturen, in denen du meist in einem Unterordner arbeitest.

Darstellung und Bedienkomfort

Für die tägliche Arbeit lohnt sich ein Blick auf folgende Einstellungen:

- **Schriftart und -größe**: Eine Monospace-Schrift mit guter Lesbarkeit für Zeichen wie `|`, `~` und `^`, die in Git-Befehlen häufig vorkommen.
- **Farbschema**: PhpStorm übernimmt standardmäßig das Editor-Farbschema; für Terminals ist oft ein dunkleres, kontrastreicheres Schema angenehmer.
- **Zeilenumbruch und Verlauf**: Die Anzahl der im Puffer gespeicherten Zeilen sollte großzügig bemessen sein, damit lange `git log`-Ausgaben nicht vorzeitig verworfen werden.
- **Kopieren bei Auswahl**: Diese Option beschleunigt das Übertragen von Commit-Hashes oder Branch-Namen erheblich, da kein explizites `Strg+C` nötig ist.

Mehrere Terminals und Split-Ansicht

PhpStorm erlaubt beliebig viele Terminal-Tabs sowie eine horizontale oder vertikale Aufteilung innerhalb eines Tabs. In der Praxis hat sich folgende Aufteilung bewährt:

1. Ein Tab für allgemeine Git-Befehle (`status`, `log`, `diff`).
2. Ein Tab für laufende Prozesse wie `php artisan serve` oder Testläufe.
3. Bei Bedarf ein dritter Tab für Composer- oder npm-Befehle.

Diese Trennung verhindert, dass lange laufende Prozesse den Blick auf Git-Ausgaben verdecken.

Zusammenspiel mit der Windows-Konfiguration

Da das integrierte Terminal letztlich dieselbe Git-Installation nutzt wie die eigenständige Git Bash aus Kapitel 2, gelten alle dort getroffenen Einstellungen unverändert weiter – insbesondere:

- `core.autocrlf` für die Behandlung von Zeilenenden,
- der konfigurierte Standard-Editor,
- der Credential Manager für die Authentifizierung.

Ein häufiger Stolperstein: Wird PhpStorm mit einem anderen Benutzerkontext oder über eine Verknüpfung mit abweichenden Umgebungsvariablen gestartet, kann das integrierte Terminal eine andere `PATH`-Variable sehen als eine manuell gestartete Git Bash. Im Zweifel hilft ein einfacher Test:

```
which git
git --version
echo $PATH
```

Weichen die Ausgaben von jenen ab, die du außerhalb von PhpStorm erhältst, liegt meist ein Konfigurationsproblem der Shell-Startdateien (`.bashrc`, `.bash_profile`) vor.

Eigene Anpassungen dauerhaft einbinden

Wer regelmäßig mit Git-Aliasen oder Prompt-Anpassungen arbeitet, sollte diese nicht in PhpStorm selbst, sondern in der jeweiligen Shell-Konfigurationsdatei hinterlegen, damit sie unabhängig vom verwendeten Werkzeug greifen. Für Git Bash ist dies typischerweise die Datei:

```
~/.bashrc
```

Ein Beispiel für eine sinnvolle Ergänzung, die den aktuellen Branch direkt im Prompt anzeigt:

```
parse_git_branch() {
    git branch 2>/dev/null | sed -n '/\* /s///p'
}
export PS1='\u@\h \W$(parse_git_branch) \$ '
```

Nach dem Speichern lädst du die Datei im Terminal neu:

```
source ~/.bashrc
```

Diese Anpassung ist besonders wertvoll, weil sie *unabhängig* von PhpStorm funktioniert – ein Vorteil, sobald du auch außerhalb der IDE arbeitest.

Terminal versus grafische Werkzeuge

Das integrierte Terminal ersetzt die GUI nicht, sondern ergänzt sie gezielt. Eine grobe Orientierung, wann welches Werkzeug vorzuziehen ist:

Aufgabe	Empfohlenes Werkzeug
Einzelne Dateien stagen, Commit schreiben	*PhpStorm-Commit-Fenster*
Diffs visuell vergleichen	*PhpStorm-Diff-Ansicht*
Interaktiver Rebase mit vielen Commits	*Terminal*
Schnelle Statusprüfung zwischendurch	*Terminal* (`git status`)
Merge-Konflikte mit Drei-Wege-Ansicht	*PhpStorm-Merge-Werkzeug*
Plumbing- oder Debug-Befehle	*Terminal*

Diese Faustregel wird dich durch den gesamten weiteren Kurs begleiten: Immer dort, wo Präzision und Geschwindigkeit der Tastatur gefragt sind, greifst du zum Terminal; immer dort, wo visuelle Übersicht zählt, nutzt du die grafischen Werkzeuge von PhpStorm.

Kurzcheck zur Konfiguration

Bevor du mit dem nächsten Kapitel fortfährst, sollten folgende Punkte erfüllt sein:

- ☐ Git Bash ist als Standard-Shell im integrierten Terminal hinterlegt.
- ☐ `git --version` liefert im Terminal dieselbe Ausgabe wie außerhalb von PhpStorm.
- ☐ Das Startverzeichnis entspricht dem Projektstamm.
- ☐ Farbschema und Schriftgröße sind angenehm lesbar eingestellt.
- ☐ Eigene `.bashrc`-Anpassungen werden korrekt geladen.

Damit steht dir ein vollwertiges, produktiv nutzbares Terminal zur Verfügung, das nahtlos in den weiteren Kurs übergeht – etwa bei der Arbeit mit *Changelists* im nächsten Abschnitt.

Revision #1
Created 2026-07-26 17:08:38 UTC by art10m
Updated 2026-07-26 17:08:49 UTC by art10m