Installationsoptionen richtig auswählen

Ziel der Installation

Git for Windows bringt nicht nur den Git-Befehl mit, sondern auch mehrere Begleitwerkzeuge:

- **Git Bash** als Unix-ähnliche Kommandozeile,
- Git für **PowerShell**, Windows Terminal und CMD,
- einen **SSH-Client**,
- den **Git Credential Manager** für sichere Anmeldungen,
- optionale Kontextmenüeinträge im Windows-Explorer,
- Werkzeuge für Zeilenenden, Zertifikate und Dateisystembesonderheiten.

Die Installationsoptionen bestimmen, wie gut Git später mit Windows 11, PhpStorm und GitHub zusammenspielt. Für einen typischen PHP-Entwicklungsrechner sind die nachstehenden Empfehlungen eine sichere, alltagstaugliche Grundlage.

> **Hinweis:** Bezeichnungen und Reihenfolge einzelner Dialoge können je nach Version von Git for Windows leicht abweichen. Die Entscheidungen und ihre Bedeutung bleiben jedoch weitgehend gleich.

Vor der Installation

Lade Git ausschließlich von der offiziellen Seite herunter:

https://git-scm.com/download/win

Für die meisten aktuellen Windows-11-Systeme wählst du die **64-Bit-Version**. Eine ARM64-Version ist passend, wenn dein Windows-Gerät tatsächlich auf ARM-Hardware läuft, etwa bestimmte Snapdragon-basierte Geräte.

Prüfe im Zweifel unter **Einstellungen → System → Info → Systemtyp**, welche Architektur dein System verwendet.

Die Installation darf in der Regel mit dem Standardkonto erfolgen. Administratorrechte sind nur nötig, wenn Git systemweit für alle Benutzer installiert werden soll oder Unternehmensrichtlinien dies verlangen.

Komponenten auswählen

Der Dialog **„Select Components“** legt fest, welche Zusatzfunktionen installiert werden.

Empfohlene Auswahl

Option	Empfehlung	Begründung
Additional icons	Optional	Fügt eine Git-Bash-Verknüpfung auf dem Desktop hinzu. Nicht erforderlich, aber für Einsteiger praktisch.
Windows Explorer integration	Aktivieren	Ermöglicht Git Bash und Git GUI direkt aus Explorer-Ordnern.
Git Bash Here	Aktivieren	Öffnet Git Bash im aktuell ausgewählten Ordner. Sehr nützlich.
Git GUI Here	Optional	Git GUI ist ein älteres grafisches Werkzeug. Für PhpStorm-Nutzer meist nicht notwendig.
Git LFS	Aktivieren	Installiert Unterstützung für große Dateien. Auch wenn du Git LFS noch nicht nutzt, schadet die Option nicht.
Associate .git configuration files	Aktivieren	Ordnet Git-Konfigurationsdateien einem Editor zu.
Associate .sh files	Optional	Praktisch, wenn du Shell-Skripte bearbeitest; nicht zwingend erforderlich.
Check daily for Git for Windows updates	Optional	Sinnvoll auf privaten Rechnern; in verwalteten Unternehmensumgebungen oft unerwünscht.
Add a Git Bash Profile to Windows Terminal	Aktivieren	Erzeugt ein Git-Bash-Profil in Windows Terminal.

Explorer-Integration sinnvoll begrenzen

Aktiviere bevorzugt **„Git Bash Here“**, aber überlege bei **„Git GUI Here“**, ob du den Explorer wirklich mit zusätzlichen Einträgen erweitern möchtest.

Für die tägliche Arbeit reichen meist:

- Git Bash im Windows Terminal,
- das integrierte Terminal in PhpStorm,
- die grafischen Git-Funktionen von PhpStorm.

Die Explorer-Integration ist vor allem dann hilfreich, wenn du spontan in einem Projektordner arbeiten möchtest.

Standardeditor für Git festlegen

Git benötigt gelegentlich einen Texteditor, etwa für:

- Commit-Nachrichten ohne `-m`,
- Merge-Commit-Nachrichten,
- interaktive Rebase-Pläne,
- Bearbeitung von Konfigurationsdateien.

Im Dialog **„Choosing the default editor used by Git“** stehen meist mehrere Editoren zur Auswahl.

Empfehlung: Visual Studio Code oder ein vertrauter Editor

Wenn Visual Studio Code installiert ist und du ihn gelegentlich nutzt, ist **Visual Studio Code** eine gute Wahl. Er öffnet Commit-Nachrichten und Rebase-Dateien verständlich und komfortabel.

Wenn du keinen zusätzlichen Editor einsetzen möchtest, ist **Vim** technisch zuverlässig, für Einsteiger aber ungewohnt. Die Bedienung unterscheidet sich deutlich von normalen Windows-Editoren und kann beim ersten Commit irritieren.

Für einen einsteigerfreundlichen Start gilt daher:

1. **Visual Studio Code**, wenn vorhanden und vertraut.
2. **Nano**, falls vom Installer angeboten und du eine einfache Terminal-Lösung bevorzugst.

3. **Vim** nur, wenn du Vim bereits kennst oder gezielt lernen möchtest.

PhpStorm kann ebenfalls als Git-Editor konfiguriert werden. Für viele Aufgaben öffnet PhpStorm aber ohnehin eigene Dialoge. Die globale Git-Editorwahl bleibt dennoch relevant, insbesondere bei Terminal-Befehlen.

> **Praxisregel:** Wähle keinen Editor, den du nicht wenigstens schließen und speichern kannst. Ein blockierter Editor ist eine häufige Ursache für scheinbar „hängende“ Git-Befehle.

PATH-Umgebung richtig einrichten

Der Dialog **„Adjusting your PATH environment“** ist besonders wichtig. Er entscheidet, aus welchen Terminals der Befehl `git` erreichbar ist.

Typischerweise bietet der Installer drei Varianten an.

Nur Git Bash verwenden

Sinngemäß lautet die Option:

> „Use Git from Git Bash only“

Git wird nur innerhalb von Git Bash verfügbar gemacht.

Nicht empfohlen, wenn du mit PhpStorm, PowerShell oder Windows Terminal arbeitest. Viele Werkzeuge erwarten, dass `git.exe` über die Windows-Umgebungsvariable `PATH` erreichbar ist.

Git aus der Kommandozeile und Drittsoftware verwenden

Sinngemäß:

>

> „Git from the command line and also from 3rd-party software“

Dies ist die **empfohlene Standardauswahl**.

Git wird damit verfügbar in:

- Git Bash,
- PowerShell,
- Windows Terminal,
- CMD,
- PhpStorm,
- vielen Entwicklungswerkzeugen und Automatisierungsskripten.

Diese Einstellung ergänzt den PATH gezielt um Git, ohne unnötig viele Unix-Werkzeuge global verfügbar zu machen.

Git und Unix-Werkzeuge global verfügbar machen

Sinngemäß:

> „Use Git and optional Unix tools from the Command Prompt“

Diese Variante stellt zusätzlich viele Unix-Befehle wie `find`, `grep` oder `sort` systemweit bereit.

Das kann praktisch wirken, birgt aber ein Risiko: Unix-Werkzeuge können Windows-Befehle gleichen Namens überlagern oder Skripte unerwartet beeinflussen. Besonders in Unternehmensumgebungen und bei älteren Automatisierungen kann das zu schwer nachvollziehbaren Problemen führen.

Für die meisten Nutzer nicht empfohlen. Nutze Unix-Werkzeuge stattdessen innerhalb von Git Bash oder WSL, falls du Linux-Werkzeuge benötigst.

SSH-Client auswählen

Git benötigt SSH, wenn du Repositories über SSH mit GitHub verbinden möchtest, beispielsweise über eine URL wie:

```
git@github.com:organisation/projekt.git
```

Der Installer fragt meist nach dem SSH-Programm.

Empfohlene Wahl: gebündeltes OpenSSH

Wähle:

> „Use bundled OpenSSH“

Das mit Git for Windows ausgelieferte OpenSSH ist abgestimmt, aktuell und funktioniert zuverlässig in Git Bash, PowerShell, Windows Terminal und PhpStorm.

Wann das externe OpenSSH sinnvoll sein kann

Windows 11 enthält häufig bereits einen eigenen OpenSSH-Client. Die Option **„Use external OpenSSH“** ist sinnvoll, wenn:

- dein Unternehmen zentral verwaltete SSH-Konfigurationen bereitstellt,
- ein vorhandener SSH-Agent verbindlich verwendet werden muss,
- du bewusst nur den Windows-Systemclient einsetzen möchtest,
- Sicherheitsrichtlinien eine bestimmte OpenSSH-Version vorgeben.

Für einen persönlichen Entwicklungsrechner oder den Einstieg ist das gebündelte OpenSSH die robustere Wahl.

> **Wichtig:** Verwende nicht unkontrolliert mehrere SSH-Konfigurationen. Wenn Git Bash, PowerShell und PhpStorm unterschiedliche Schlüssel oder SSH-Programme verwenden, entstehen schwer verständliche Authentifizierungsprobleme.

HTTPS-Transport-Backend auswählen

Für HTTPS-Verbindungen zu GitHub muss Git Zertifikate prüfen. Der Installer bietet meist zwei Varianten an.

OpenSSL-Bibliothek verwenden

Sinngemäß:

> „Use the OpenSSL library“

Git bringt eine eigene Zertifikatsverwaltung mit. Diese Wahl ist auf vielen Systemen zuverlässig und war lange die Standardempfehlung.

Windows Secure Channel verwenden

Sinngemäß:

> „Use the native Windows Secure Channel library“

Diese Option nutzt den Windows-Zertifikatsspeicher. Zertifikate, die Windows bereits als vertrauenswürdig kennt, werden auch von Git akzeptiert.

Welche Wahl ist richtig?

Für einen üblichen privaten Windows-11-Rechner funktionieren **beide Varianten**. Die Standardauswahl des Installers ist in der Regel eine vernünftige Wahl.

Windows Secure Channel ist besonders passend, wenn:

- dein Unternehmen eigene Root-Zertifikate verteilt,
- ein Proxy HTTPS-Verbindungen kontrolliert,
- Zertifikate zentral über Windows verwaltet werden.

OpenSSL ist passend, wenn:

- du eine möglichst eigenständige Git-Installation bevorzugst,
- keine Unternehmenszertifikate eingebunden werden müssen,
- du die Standardauswahl beibehalten möchtest.

>

> **Sicherheitsregel:** Deaktiviere die Zertifikatsprüfung nicht mit Einstellungen wie `http.sslVerify=false`. Das löst keine Vertrauensprobleme, sondern umgeht eine wichtige Schutzfunktion.

Zeilenenden korrekt konfigurieren

Der Dialog **„Configuring the line ending conversions“** betrifft einen häufigen Unterschied zwischen Windows und Unix-Systemen.

Windows verwendet traditionell die Zeilenendung **CRLF**, während Linux, macOS und die meisten Serverumgebungen **LF** verwenden.

Technisch:

$$
\text{Windows-Zeilenende} = \text{CRLF}
$$

$$
\text{Unix-Zeilenende} = \text{LF}
$$

Git kann beim Auschecken und Speichern automatisch zwischen beiden Darstellungen umwandeln.

Empfohlene Auswahl für Windows-Entwicklung

Wähle üblicherweise:

> „Checkout Windows-style, commit Unix-style line endings“

Diese Auswahl entspricht typischerweise:

```
git config --global core.autocrlf true
```

Das Verhalten lautet:

- Beim Auschecken werden Textdateien im Arbeitsverzeichnis als CRLF bereitgestellt.
- Beim Commit speichert Git Textdateien mit LF im Repository.

Damit bleiben Repository-Inhalte auf Servern und in gemischten Teams konsistent, während Windows-Editoren trotzdem mit CRLF arbeiten können.

Alternative für strikt normalisierte Projekte

Manche Teams verwenden bewusst LF auch im Windows-Arbeitsverzeichnis. Dann kann folgende Einstellung passend sein:

```
git config --global core.autocrlf input
```

Sie konvertiert beim Commit nach LF, verändert aber Zeilenenden beim Auschecken nicht.

Diese Variante ist verbreitet, wenn:

- Editor und Projekt konsequent auf LF eingestellt sind,
- `.gitattributes` die Zeilenenden verbindlich steuert,
- Container, WSL oder Linux-nahe Werkzeuge im Einsatz sind.

Für den Einstieg unter Windows 11 ist `core.autocrlf=true` meist die unkompliziertere Wahl. Später wird die teamweite, reproduzierbare Steuerung über `.gitattributes` wichtiger als eine persönliche globale Einstellung.

Terminal-Emulator für Git Bash wählen

Git Bash benötigt ein Terminalfenster. Der Installer bietet meist zwei Optionen:

- **MinTTY**
- die klassische **Windows-Konsole**

Empfehlung: MinTTY

Wähle:

> „Use MinTTY“

MinTTY bietet eine angenehmere Nutzung von Git Bash:

- bessere Textauswahl und Kopierfunktionen,
- gute Unicode-Unterstützung,
- flexiblere Fensterdarstellung,
- konsistenteres Unix-artiges Verhalten.

Windows-Konsole nur bei Sonderfällen

Die Windows-Konsole kann sinnvoll sein, wenn du alte Windows-Programme direkt innerhalb von Git Bash verwenden musst und diese Probleme mit MinTTY haben.

Für die normale Git-Arbeit, PHP-Projekte und PhpStorm ist **MinTTY** die bessere Wahl. Zusätzlich kannst du Git Bash später im Windows Terminal verwenden.

Verhalten von `git pull` festlegen

Neuere Git-for-Windows-Installer fragen nach dem Standardverhalten von `git pull`.

Ein Pull besteht konzeptionell aus zwei Schritten:

$$
\texttt{git pull} = \texttt{git fetch} + \text{Integration lokaler Änderungen}
$$

Die Integration kann per Merge, Rebase oder ausschließlich als Fast-Forward erfolgen.

Empfohlene Wahl für den Einstieg: Fast-Forward oder Merge

Wenn angeboten, ist **Fast-forward or merge** eine sichere Wahl. Git führt einen Fast-Forward aus, wenn möglich, und erstellt anderenfalls einen Merge.

Das ist transparent und verändert vorhandene lokale Commits nicht nachträglich.

Rebase nur mit bewusstem Teamstandard

Die Option **Rebase** kann eine lineare Historie erzeugen, schreibt aber lokale Commits beim Pull um. Das ist nützlich, wenn dein Team diesen Ablauf verbindlich nutzt und du Rebase sicher beherrschst.

Für den Anfang ist sie nicht die beste Standardwahl. Rebase wird später gezielt und kontrolliert eingesetzt.

Fast-Forward only als strenge Schutzoption

Fast-forward only bricht den Pull ab, sobald lokale und entfernte Historie auseinanderlaufen. Das verhindert automatische Merge-Commits, verlangt aber eine bewusste Entscheidung:

```
git pull --rebase
```

oder:

```
git merge origin/main
```

Diese Einstellung eignet sich gut für Teams mit klaren Regeln, kann Einsteiger aber zunächst häufiger mit Abbrüchen konfrontieren.

Credential Helper aktivieren

GitHub akzeptiert für Git-Operationen über HTTPS kein Kontopasswort mehr. Stattdessen kommen beispielsweise Personal Access Tokens oder browserbasierte Anmeldungen zum Einsatz.

Wähle im Dialog zum Credential Helper:

> „Git Credential Manager“

Der **Git Credential Manager** speichert Zugangsdaten nicht als Klartext in einer Projektdatei. Er verwendet unter Windows den geschützten Windows-Anmeldeinformationsmanager und unterstützt moderne Anmeldeflüsse.

Das ist besonders hilfreich für:

- HTTPS-URLs von GitHub,
- Anmeldung über den Browser,
- tokenbasierte Authentifizierung,
- mehrere GitHub-Konten,
- Zugriff auf private Organisationen.

Die Zugangsdaten kannst du später in Windows unter **Anmeldeinformationsverwaltung** prüfen oder entfernen.

> **Wichtig:** Ein Personal Access Token ist wie ein Passwort zu behandeln. Speichere ihn nie in Quellcode, Commit-Nachrichten, Screenshots oder unverschlüsselten Textdateien.

Zusätzliche Optionen am Ende der Installation

Je nach Installer-Version erscheinen noch weitere Auswahlmöglichkeiten.

Dateisystem-Caching aktivieren

Aktiviere:

> „Enable file system caching“

Diese Option verbessert die Leistung, insbesondere bei größeren Repositories. Git kann Dateiinformationen effizienter wiederverwenden.

Für typische Entwicklungsprojekte ist das die richtige Wahl.

Symbolische Links aktivieren

Die Option **„Enable symbolic links“** sollte nur bewusst aktiviert werden.

Symbolische Links funktionieren unter Windows 11 grundsätzlich besser als früher, können aber abhängig von Benutzerrechten, Entwicklermodus, Netzlaufwerken und Sicherheitsrichtlinien problematisch sein.

Aktiviere sie, wenn:

- dein Projekt tatsächlich symbolische Links versioniert,
- du den Windows-Entwicklermodus aktiviert hast oder passende Rechte besitzt,
- alle Teammitglieder und Zielsysteme damit umgehen können.

Lass sie zunächst deaktiviert, wenn du keinen konkreten Bedarf hast. Git kann symbolische Links dann nicht vollständig wie auf Linux behandeln, aber du vermeidest unerwartete Berechtigungsprobleme.

Pseudo-Konsole oder Legacy-Konsole

Manche Installer-Versionen fragen nach dem Verhalten der Konsole, beispielsweise nach **Pseudo Console Support**.

Die moderne Standardauswahl ist in der Regel sinnvoll. Ändere sie nur, wenn du konkrete Kompatibilitätsprobleme mit alten Konsolenanwendungen beobachtest.

Empfohlene Konfiguration im Überblick

Für einen typischen PHP-Workflow mit Windows 11, GitHub und PhpStorm empfiehlt sich folgende Zusammenfassung:

Installationsbereich	Empfohlene Entscheidung
Explorer-Integration	Git Bash Here aktivieren
Git LFS	Aktivieren
Windows Terminal-Profil	Aktivieren
Standardeditor	Visual Studio Code oder ein vertrauter Editor
PATH	Git aus Kommandozeile und Drittsoftware verwenden
SSH	Gebündeltes OpenSSH verwenden
HTTPS-Zertifikate	Standardauswahl; in Unternehmen häufig Windows Secure Channel

Installationsbereich	Empfohlene Entscheidung
Zeilenenden	Windows auschecken, Unix committen
Git-Bash-Terminal	MinTTY verwenden
Pull-Verhalten	Fast-forward oder Merge
Credential Helper	Git Credential Manager
Dateisystem-Caching	Aktivieren
Symbolische Links	Nur bei konkretem Bedarf aktivieren

Nach der Installation prüfen

Öffne **PowerShell**, Windows Terminal oder Git Bash und prüfe zuerst, ob Git gefunden wird:

```
git --version
```

Eine erfolgreiche Ausgabe ähnelt diesem Muster:

```
git version 2.x.x.windows.x
```

Prüfe danach, von welchem Pfad Git gestartet wird.

In PowerShell:

```
Get-Command git
```

In Git Bash:

```
which git
```

Kontrolliere außerdem die wichtigsten vom Installer gesetzten Einstellungen:

```
git config --global --get core.autocrlf
git config --global --get credential.helper
git config --global --get pull.rebase
```

Nicht jede Installation setzt alle Werte sichtbar als globale Konfiguration. Eine leere Ausgabe ist daher nicht automatisch ein Fehler; manche Vorgaben stammen aus der Systemkonfiguration oder bleiben auf dem Git-Standardwert.

Alle wirksamen Konfigurationswerte inklusive ihrer Herkunft zeigst du mit diesem Befehl an:

```
git config --list --show-origin
```

Häufige Fehlentscheidungen

Git ist nur in Git Bash verfügbar

Ursache: Bei der PATH-Auswahl wurde „Git Bash only“ gewählt.

Folge: PowerShell, PhpStorm oder andere Werkzeuge finden `git` möglicherweise nicht.

Lösung: Git erneut installieren und die Option für Kommandozeile und Drittsoftware wählen. Alternativ kann der PATH manuell korrigiert werden; eine Neuinstallation ist für Einsteiger jedoch meist sicherer.

GitHub fragt bei jedem Push erneut nach Zugangsdaten

Mögliche Ursachen:

- Git Credential Manager wurde nicht aktiviert,
- gespeicherte Zugangsdaten sind ungültig,
- es wird eine HTTPS-URL verwendet, aber kein Token oder Browser-Login abgeschlossen,
- mehrere GitHub-Konten werden vermischt.

Lösung: Den Git Credential Manager aktivieren und gespeicherte GitHub-Anmeldeinformationen in der Windows-Anmeldeinformationsverwaltung prüfen.

Unerwartete Änderungen in fast allen Dateien

Typische Ursache: Zeilenenden wurden automatisch umgewandelt oder das Projekt verwendet eigene Regeln.

Lösung: Nicht sofort alles committen. Prüfe zuerst:

```
git status
git diff --ignore-space-at-eol
```

Untersuche anschließend die Datei `.gitattributes`, falls sie vorhanden ist. Sie kann die globale Einstellung `core.autocrlf` für dieses Repository übersteuern.

SSH funktioniert im Terminal, aber nicht in PhpStorm

Typische Ursache: PhpStorm verwendet einen anderen SSH-Client, einen anderen Schlüssel oder einen anderen Agenten.

Lösung: Später in PhpStorm unter den Git- und SSH-Einstellungen prüfen, welcher SSH-Executable-Pfad und welcher Authentifizierungsmodus aktiv sind. Verwende möglichst konsistent denselben Schlüsselbestand.

Entscheidungshilfe für spätere Änderungen

Die meisten Installationsentscheidungen lassen sich nachträglich ändern. Du musst Git nicht bei jeder kleinen Anpassung neu installieren.

Beispiele:

```
git config --global core.autocrlf true
```

```
git config --global pull.rebase false
```

```
git config --global credential.helper manager
```

Für größere Änderungen, insbesondere an PATH, SSH-Komponente oder Explorer-Integration, ist eine erneute Ausführung des Git-for-Windows-Installers oft der sauberste Weg. Der Installer erkennt eine vorhandene Installation und bietet üblicherweise eine Anpassung oder Aktualisierung an.

> **Merksatz:** Wähle bei der Installation sichere, konservative Standards. Git Bash, Git im PATH, Git Credential Manager und eine kontrollierte Zeilenendenstrategie bilden die wichtigste Grundlage für die weitere Arbeit.

Revision #1
Created 2026-07-25 11:13:43 UTC by art10m
Updated 2026-07-25 11:13:43 UTC by art10m