Git in der PATH-Variable prüfen

Lernziel

Nach dieser Einheit kannst du sicher feststellen, ob Git unter Windows 11 systemweit über die `PATH`-Variable erreichbar ist. Du erkennst typische Fehlermeldungen, verstehst den Unterschied zwischen verschiedenen Terminals und weißt, wie du eine fehlerhafte PATH-Konfiguration sauber korrigierst.

Was bedeutet `PATH`?

`PATH` ist eine Windows-Umgebungsvariable. Sie enthält eine geordnete Liste von Verzeichnissen, in denen Windows nach ausführbaren Programmen sucht.

Wenn du in einem Terminal Folgendes eingibst:

```
git --version
```

sucht Windows unter anderem in allen Verzeichnissen aus `PATH` nach einer Datei wie `git.exe`.

Ist das Git-Installationsverzeichnis enthalten, wird Git gefunden und gestartet. Fehlt es, erscheint eine Fehlermeldung wie:

```
git : Der Begriff „git“ ist entweder nicht als Name eines Cmdlet,
einer Funktion, einer Skriptdatei oder eines ausführbaren Programms erkannt.
```

Die PATH-Variable macht es also möglich, Git aus *jedem beliebigen Ordner* aufzurufen, ohne den vollständigen Programmpfad eingeben zu müssen.

Ohne PATH-Eintrag müsste ein Git-Aufruf beispielsweise so aussehen:

```
& "C:\Program Files\Git\cmd\git.exe" --version
```

Das ist technisch möglich, aber im Alltag unpraktisch.

Git in PowerShell prüfen

Öffne **Windows PowerShell** oder ein PowerShell-Profil im Windows Terminal und führe aus:

```
git --version
```

Bei einer erfolgreichen Installation sieht die Ausgabe ähnlich aus:

```
git version 2.47.1.windows.1
```

Die genaue Versionsnummer kann bei dir abweichen.

Prüfe anschließend, *welche* ausführbare Datei PowerShell verwendet:

```
Get-Command git
```

Eine typische Ausgabe enthält einen Pfad wie diesen:

```
CommandType     Name       Version    Source
-----------     ----       -------    ------
Application     git.exe    2.47.1...  C:\Program Files\Git\cmd\git.exe
```

Besonders wichtig ist die Spalte **Source**. Sie zeigt, welche `git.exe` tatsächlich ausgeführt wird.

Nur den vollständigen Git-Pfad ausgeben

Für eine kompakte Prüfung eignet sich:

```
(Get-Command git).Source
```

Erwartete Ausgabe:

```
C:\Program Files\Git\cmd\git.exe
```

Git in der Eingabeaufforderung prüfen

Öffne die klassische **Eingabeaufforderung** — `cmd.exe` — und führe aus:

```
git --version
```

Danach kannst du mit `where` prüfen, welche Git-Installation gefunden wird:

```
where git
```

Beispiel:

```
C:\Program Files\Git\cmd\git.exe
```

Falls mehrere Treffer erscheinen, sind mehrere Git-Programme oder Git-Pfade in deiner PATH-Variable vorhanden:

```
C:\Program Files\Git\cmd\git.exe
C:\Tools\Git\cmd\git.exe
```

> **Empfehlung:** Verwende nach Möglichkeit nur eine aktive Git-for-Windows-Installation. Mehrere Versionen können zu schwer nachvollziehbaren Unterschieden zwischen Terminal, IDE und Automatisierung führen.

Git in Git Bash prüfen

Öffne **Git Bash** über das Startmenü und führe aus:

```
git --version
```

Ergänzend zeigt dieser Befehl den aufgelösten Pfad:

```
which git
```

Typische Ausgabe:

```
/mingw64/bin/git
```

Git Bash verwendet eine Unix-ähnliche Pfaddarstellung. Der angezeigte Pfad entspricht typischerweise einer Datei innerhalb deiner Git-for-Windows-Installation.

Git Bash funktioniert häufig auch dann noch, wenn Git in PowerShell oder CMD nicht gefunden wird. Der Grund: Git Bash startet mit einer eigenen, von Git for Windows vorbereiteten Umgebung. Deshalb ist eine erfolgreiche Prüfung in Git Bash **kein vollständiger Beweis**, dass Git auch in der Windows-PATH-Variable korrekt eingetragen ist.

Prüfe Git daher mindestens einmal zusätzlich in PowerShell oder CMD.

Die PATH-Variable anzeigen

In PowerShell

Um die PATH-Einträge der aktuellen Sitzung anzuzeigen:

```
$env:Path -split ';'
```

Suche in der Ausgabe nach einem Eintrag wie:

```
C:\Program Files\Git\cmd
```

Oder filtere direkt nach Git:

```
$env:Path -split ';' | Where-Object { $_ -match 'Git' }
```

Eine typische Ausgabe lautet:

```
C:\Program Files\Git\cmd
```

In CMD

In der Eingabeaufforderung kannst du die gesamte Variable ausgeben:

```
echo %PATH%
```

Die Ausgabe ist lang und verwendet Semikolons als Trennzeichen:

```
C:\Windows\system32;C:\Windows;C:\Program Files\Git\cmd;...
```

Welcher Git-Pfad gehört in PATH?

Bei einer Standardinstallation von **Git for Windows** ist meist dieser Eintrag sinnvoll:

```
C:\Program Files\Git\cmd
```

Darin befindet sich unter anderem:

```
C:\Program Files\Git\cmd\git.exe
```

Dieser Pfad stellt Git für PowerShell, CMD, Windows Terminal, PhpStorm und viele andere Windows-Programme bereit.

Manchmal findest du auch:

```
C:\Program Files\Git\bin
```

oder:

```
C:\Program Files\Git\mingw64\bin
```

Diese Verzeichnisse können funktionieren, sind aber für den allgemeinen Windows-Einsatz normalerweise nicht die beste erste Wahl. Der Ordner `cmd` ist der übliche und von Git for Windows vorgesehene PATH-Eintrag.

> **Wichtig:** Füge nicht wahllos mehrere Unterordner derselben Git-Installation zu PATH hinzu. Mehrere Treffer für Hilfsprogramme können unerwartete Nebeneffekte verursachen.

Prüfschritt: Git außerhalb des Installationsordners ausführen

Wechsle in einen beliebigen Ordner, der nichts mit Git zu tun hat, etwa dein Benutzerverzeichnis:

```
Set-Location $HOME
git --version
```

Oder in CMD:

```
cd %USERPROFILE%
git --version
```

Wenn Git jetzt erfolgreich ausgeführt wird, funktioniert die PATH-Auflösung unabhängig vom aktuellen Arbeitsordner.

Als zusätzlicher Test kannst du ein Git-Kommando ohne Repository ausführen:

```
git config --global --list
```

Wenn Git erreichbar ist, zeigt der Befehl deine globale Konfiguration an oder bleibt ohne Ausgabe, falls noch keine Werte gesetzt wurden. Es darf dabei keine Meldung erscheinen, dass `git` unbekannt sei.

Häufige Fehlermeldungen richtig einordnen

„git“ ist nicht erkannt

PowerShell kann beispielsweise melden:

```
git : Der Begriff „git“ ist nicht als Name eines Cmdlet,
einer Funktion, einer Skriptdatei oder eines ausführbaren Programms erkannt.
```

CMD verwendet oft diese Formulierung:

```
'git' is not recognized as an internal or external command,
operable program or batch file.
```

Bedeutung: Windows findet keine passende `git.exe` über PATH.

Mögliche Ursachen:

- Git for Windows wurde nicht installiert.
- Bei der Installation wurde keine PATH-Integration ausgewählt.
- Das Terminal war bereits geöffnet, bevor Git installiert wurde.
- Der PATH-Eintrag wurde manuell entfernt oder beschädigt.
- Du arbeitest in einer Umgebung mit abweichenden oder eingeschränkten Variablen.

„git“ wird gefunden, aber die falsche Version startet

Wenn `where git` mehrere Pfade ausgibt, kann eine alte Installation Vorrang haben. Prüfe daher:

```
where git
git --version
```

In PowerShell:

```
Get-Command git -All
```

Die Reihenfolge in PATH ist entscheidend: Windows verwendet im Regelfall den ersten passenden Fund.

Git funktioniert in Git Bash, aber nicht in PowerShell

Das weist meist auf einen fehlenden oder unvollständigen PATH-Eintrag hin. Git Bash bringt ihre eigene Umgebung mit; PowerShell übernimmt dagegen die Windows-Umgebungsvariablen.

Prüfe in PowerShell:

```
Get-Command git
```

Wird kein Pfad ausgegeben, musst du Git for Windows reparieren oder den PATH-Eintrag ergänzen.

PATH-Konfiguration sicher korrigieren

Bevorzugter Weg: Git for Windows erneut ausführen

Der zuverlässigste Weg ist meist, den Git-for-Windows-Installer erneut zu starten und die bestehende Installation zu **ändern** oder zu **reparieren**.

Achte im Installationsdialog auf die Option zur PATH-Integration. Wähle die Einstellung, die Git über die Windows-Kommandozeile und Programme von Drittanbietern verfügbar macht. Die Bezeichnung kann je nach Installer-Version leicht variieren, enthält aber typischerweise einen Hinweis auf:

```
Git from the command line and also from 3rd-party software
```

Diese Auswahl fügt den benötigten Eintrag üblicherweise automatisch hinzu.

Manueller Weg: PATH-Eintrag ergänzen

Wenn Git korrekt installiert ist, aber der Eintrag fehlt, kannst du ihn manuell ergänzen.

1. Öffne das Startmenü und suche nach **Umgebungsvariablen**.
2. Wähle **Systemumgebungsvariablen bearbeiten**.
3. Klicke im Dialog auf **Umgebungsvariablen**.
4. Wähle unter *Benutzervariablen* oder *Systemvariablen* den Eintrag **Path**.
5. Klicke auf **Bearbeiten**.
6. Klicke auf **Neu**.
7. Trage den Git-Pfad ein:

   ```
   C:\Program Files\Git\cmd
   ```

8. Bestätige alle Fenster mit **OK**.
9. Schließe alle geöffneten Terminals und öffne ein neues Terminalfenster.

Prüfe anschließend erneut:

```
git --version
(Get-Command git).Source
```

> **Hinweis:** Für einen einzelnen Windows-Benutzer genügt in der Regel ein Eintrag unter den *Benutzervariablen*. Ein Eintrag unter den *Systemvariablen* ist sinnvoll, wenn Git für alle Benutzerkonten des Computers verfügbar sein soll.

Warum muss das Terminal neu gestartet werden?

Umgebungsvariablen werden beim Start eines Prozesses übernommen. Ein bereits geöffnetes PowerShell-, CMD- oder Windows-Terminal-Fenster kennt Änderungen an PATH daher noch nicht automatisch.

Nach einer Installation oder PATH-Änderung gilt:

1. Alle Terminalfenster schließen.
2. Windows Terminal gegebenenfalls vollständig beenden.
3. Ein neues Terminal öffnen.
4. `git --version` erneut ausführen.

Auch PhpStorm sollte neu gestartet werden, wenn es während der Git-Installation bereits geöffnet war. Die IDE übernimmt die Umgebung ebenfalls beim Start.

Prüfung aus PhpStorm heraus

Nachdem Git in PowerShell oder CMD gefunden wird, prüfst du die IDE-Anbindung separat:

1. Öffne in PhpStorm **File → Settings**.
2. Navigiere zu **Version Control → Git**.
3. Kontrolliere das Feld **Path to Git executable**.
4. Klicke auf **Test**.

PhpStorm sollte eine Meldung ähnlich dieser anzeigen:

```
Git version is 2.47.1.windows.1
```

Falls PhpStorm Git nicht findet, obwohl `git --version` im Terminal funktioniert, kannst du den Pfad direkt eintragen:

```
C:\Program Files\Git\cmd\git.exe
```

Die explizite Angabe ist besonders hilfreich, wenn mehrere Git-Versionen installiert sind oder PhpStorm mit einer abweichenden Umgebung gestartet wurde.

Kompakte Diagnose-Checkliste

Führe diese Befehle nacheinander in PowerShell aus:

```
git --version
Get-Command git
(Get-Command git).Source
$env:Path -split ';' | Where-Object { $_ -match 'Git' }
```

Eine funktionierende Konfiguration erfüllt diese Kriterien:

- `git --version` gibt eine Git-Version aus.
- `Get-Command git` zeigt eine Anwendung namens `git.exe`.
- Der aufgelöste Pfad verweist auf die gewünschte Git-for-Windows-Installation.
- PATH enthält typischerweise `C:\Program Files\Git\cmd`.
- PhpStorm bestätigt die Git-Version im VCS-Testdialog.

Merksatz

> **Git Bash kann Git kennen, obwohl Windows Git noch nicht über PATH kennt.**
> Entscheidend für eine zuverlässige Nutzung in PowerShell, CMD, Windows Terminal und PhpStorm ist der erfolgreiche Test mit `git --version` außerhalb von Git Bash.

Revision #1
Created 2026-07-25 11:13:44 UTC by art10m
Updated 2026-07-25 11:13:44 UTC by art10m