Git Bash, PowerShell und CMD vergleichen

Drei Konsolen, drei Aufgabenbereiche

Unter Windows 11 kannst du Git aus verschiedenen Kommandozeilen heraus verwenden. Die wichtigsten sind:

- **Git Bash** – eine Unix-ähnliche Shell, die mit Git for Windows installiert wird
- **PowerShell** – die moderne Windows-Shell für Administration und Automatisierung
- **Eingabeaufforderung** bzw. **CMD** – die klassische Windows-Konsole

Alle drei können denselben Git-Befehl ausführen, sofern Git korrekt installiert und über die PATH-Variable erreichbar ist:

```
git --version
git status
git commit -m "Dokumentation ergänzen"
```

Der Unterschied liegt also nicht in Git selbst, sondern in der Umgebung: Pfade, Dateibefehle, Skripte, Variablen und die Art, wie Ausgaben verarbeitet werden, unterscheiden sich deutlich.

> **Grundregel:** Git-Kommandos bleiben weitgehend gleich. Shell-spezifisch sind vor allem Befehle *um Git herum*.

Git Bash

Git Bash wird zusammen mit **Git for Windows** installiert. Sie stellt eine Bash-Umgebung bereit, die sich an Linux und macOS orientiert. Viele in Git-Dokumentationen, Tutorials und Open-Source-Projekten verwendete Befehle funktionieren dort direkt.

Typische Eigenschaften

- Unix-ähnliche Befehle wie `ls`, `cd`, `cat`, `grep` und `rm`
- Pfadangaben im Unix-Stil
- Gute Kompatibilität mit vielen Git-Anleitungen
- Enthält üblicherweise SSH, OpenSSH-Agent, Git und weitere Hilfsprogramme
- Besonders geeignet für Git, PHP-Projekte, Composer und Shell-Skripte

Ein typischer Startbildschirm könnte so aussehen:

```
alex@PC MINGW64 ~
$
```

`~` steht für dein persönliches Benutzerverzeichnis.

Pfade in Git Bash

Windows-Pfade werden in Git Bash normalerweise anders geschrieben:

Windows	Git Bash
`C:\Users\Alex`	`/c/Users/Alex`
`D:\Projekte\shop`	`/d/Projekte/shop`
`C:\Program Files\Git`	`/c/Program Files/Git`

In einen Projektordner wechselst du beispielsweise so:

```
cd /c/Users/Alex/Projekte/mein-projekt
```

Enthält ein Pfad Leerzeichen, muss er in Anführungszeichen stehen:

```
cd "/c/Users/Alex/Meine Projekte/mein-projekt"
```

Häufige Dateibefehle in Git Bash

```
pwd
ls
```

```
ls -la
mkdir mein-projekt
touch README.md
cp quelle.txt ziel.txt
mv alt.txt neu.txt
rm datei.txt
```

Dabei bedeuten die wichtigsten Befehle:

- `pwd` zeigt den aktuellen Ordner an.
- `ls` listet Dateien und Ordner auf.
- `mkdir` erstellt einen Ordner.
- `touch` erstellt eine leere Datei oder aktualisiert ihren Zeitstempel.
- `cp` kopiert Dateien.
- `mv` verschiebt oder benennt Dateien um.
- `rm` löscht Dateien.

⚠ `rm` besitzt keinen Papierkorb. Eine damit gelöschte Datei ist in der Regel sofort entfernt. Verwende insbesondere `rm -rf` nur mit äußerster Vorsicht.

Wann Git Bash besonders sinnvoll ist

Git Bash ist für diesen Kurs meist die beste Wahl, wenn du:

- Git anhand von Dokumentation und Tutorials lernen möchtest,
- Git-Befehle unter Windows möglichst ähnlich wie unter Linux oder macOS verwenden willst,
- mit SSH-Schlüsseln arbeitest,
- Shell-Skripte aus Open-Source-Projekten ausführen musst,
- Befehle wie `grep`, `find`, `sed` oder `cat` nutzen möchtest.

Für Einsteiger ist Git Bash oft angenehmer, weil viele Git-Beispiele im Internet genau diese oder eine sehr ähnliche Umgebung voraussetzen.

PowerShell

PowerShell ist die moderne Kommandozeile von Windows. Sie ist auf Systemverwaltung, Automatisierung und strukturierte Datenverarbeitung ausgelegt. Unter Windows 11 ist sie standardmäßig verfügbar.

Je nach Installation begegnen dir zwei Varianten:

- **Windows PowerShell** – die ältere, vorinstallierte Variante
- **PowerShell** bzw. PowerShell 7+ – die moderne, plattformübergreifende Weiterentwicklung

Für Git funktionieren beide Varianten grundsätzlich gleich.

Typische Eigenschaften

- Native Integration in Windows
- Zugriff auf Windows-Dienste, Registry, Prozesse und Dateisystem
- Leistungsfähige Skriptsprache
- Verarbeitung von Objekten statt nur Textzeilen
- Git kann direkt verwendet werden, wenn es im PATH liegt

Ein typischer Prompt sieht etwa so aus:

```
PS C:\Users\Alex>
```

Pfade in PowerShell

PowerShell verwendet normalerweise klassische Windows-Pfade:

```
cd C:\Users\Alex\Projekte\mein-projekt
```

Alternativ akzeptiert PowerShell in vielen Fällen auch Schrägstriche:

```
cd C:/Users/Alex/Projekte/mein-projekt
```

Bei Leerzeichen setzt du den gesamten Pfad in Anführungszeichen:

```
cd "C:\Users\Alex\Meine Projekte\mein-projekt"
```

Häufige Dateibefehle in PowerShell

```
Get-Location
Get-ChildItem
New-Item -ItemType Directory mein-projekt
New-Item -ItemType File README.md
Copy-Item quelle.txt ziel.txt
Move-Item alt.txt neu.txt
```

```
Remove-Item datei.txt
```

PowerShell kennt zusätzlich viele kurze Aliase:

```
pwd
ls
mkdir mein-projekt
cp quelle.txt ziel.txt
mv alt.txt neu.txt
rm datei.txt
```

Diese Kurzformen sehen Git-Bash-Befehlen ähnlich, sind aber nicht immer exakt gleich. Hinter `ls` steckt in PowerShell beispielsweise `Get-ChildItem`, nicht der Unix-Befehl `ls`.

> Verwende bei PowerShell-Skripten möglichst die vollständigen Befehlsnamen wie `Get-ChildItem` oder `Remove-Item`. Das macht Skripte verständlicher und vermeidet Missverständnisse zwischen Shells.

Besonderheit: PowerShell verarbeitet Objekte

Git Bash und CMD behandeln Kommandoausgaben hauptsächlich als Text. PowerShell verarbeitet dagegen oft strukturierte .NET-Objekte.

Ein Beispiel: Alle laufenden Prozesse anzeigen und nach einem Namen filtern:

```
Get-Process | Where-Object ProcessName -like "*php*"
```

Das ist für Windows-Automatisierung sehr nützlich, aber für einfache Git-Aufgaben zunächst nicht entscheidend.

Wann PowerShell besonders sinnvoll ist

PowerShell ist eine gute Wahl, wenn du:

- ohnehin häufig mit Windows arbeitest,
- Windows-spezifische Aufgaben automatisierst,
- Skripte für Dateien, Dienste, Prozesse oder Netzwerkverwaltung schreibst,
- Git und Windows-Administration in einer einheitlichen Umgebung kombinieren willst,

- das integrierte Terminal von PhpStorm oder Windows Terminal mit einem Windows-Profil nutzen möchtest.

Git-Befehle selbst funktionieren in PowerShell zuverlässig:

```
git status
git add .
git commit -m "Startprojekt anlegen"
git log --oneline
```

Die Eingabeaufforderung CMD

CMD, offiziell `cmd.exe`, ist die klassische Eingabeaufforderung von Windows. Sie ist seit vielen Windows-Versionen vorhanden und besonders ressourcenschonend. Für moderne Automatisierung bietet sie jedoch deutlich weniger Möglichkeiten als PowerShell.

Ein CMD-Prompt sieht typischerweise so aus:

```
C:\Users\Alex>
```

Typische Eigenschaften

- Auf praktisch jedem Windows-System verfügbar
- Einfach und schnell zu starten
- Kompatibel mit älteren Batch-Dateien
- Weniger komfortabel für Automatisierung als PowerShell
- Keine Unix-typischen Werkzeuge wie `grep`, `cat` oder `touch` ohne zusätzliche Installation

Pfade und Dateibefehle in CMD

CMD verwendet Windows-Pfade:

```
cd C:\Users\Alex\Projekte\mein-projekt
```

Bei einem Wechsel auf ein anderes Laufwerk brauchst du entweder zwei Schritte:

```
D:
cd \Projekte\mein-projekt
```

Oder du verwendest `cd /d`, das Laufwerk und Ordner gemeinsam wechselt:

```
cd /d D:\Projekte\mein-projekt
```

Häufige Dateibefehle:

```
cd
dir
mkdir mein-projekt
type nul > README.md
copy quelle.txt ziel.txt
move alt.txt neu.txt
del datei.txt
rmdir mein-ordner
```

Auch hier gilt: `del` und `rmdir` verschieben nichts in den Papierkorb.

Git in CMD verwenden

Wenn Git im PATH eingerichtet wurde, funktionieren Git-Befehle wie gewohnt:

```
git --version
git init
git status
git add README.md
git commit -m "Ersten Commit erstellen"
```

CMD ist deshalb nicht ungeeignet für Git. Es ist lediglich weniger komfortabel, wenn du viele Dateisystem-, Such- oder Automatisierungsbefehle benötigst.

Wann CMD noch sinnvoll ist

CMD ist passend, wenn du:

- ein altes Batch-Skript mit `.bat` oder `.cmd` ausführen musst,
- auf einem eingeschränkten Windows-System arbeitest,
- eine einfache, überall verfügbare Konsole brauchst,
- gezielt ältere Windows-Werkzeuge verwendest.

Für neue persönliche Git-Workflows ist Git Bash oder PowerShell normalerweise die bessere Wahl.

Direkter Vergleich

Kriterium	Git Bash	PowerShell	CMD
Primärer Zweck	Git und Unix-nahe Entwicklung	Windows-Administration und Automatisierung	Klassische Windows-Kommandos
Pfadstil	`/c/Users/Alex`	`C:\Users\Alex`	`C:\Users\Alex`
Unix-Befehle	Sehr gut verfügbar	Teilweise als Aliase vorhanden	Meist nicht verfügbar
Git-Dokumentationen nachvollziehen	Sehr gut	Gut	Gut
Windows-Systemverwaltung	Eingeschränkt	Sehr gut	Eingeschränkt
Moderne Skripte	Bash-Skripte	PowerShell-Skripte	Batch-Skripte
Einstieg für Git	Sehr empfehlenswert	Empfehlenswert	Möglich, aber weniger komfortabel
Typische Skriptdateien	`.sh`	`.ps1`	`.bat`, `.cmd`

Dieselbe Git-Aufgabe in allen drei Konsolen

Angenommen, dein Projekt liegt unter `C:\Users\Alex\Projekte\demo-app`.

Git Bash

```
cd /c/Users/Alex/Projekte/demo-app
git status
git add README.md
git commit -m "README ergänzen"
```

PowerShell

```
cd C:\Users\Alex\Projekte\demo-app
git status
git add README.md
git commit -m "README ergänzen"
```

CMD

```
cd /d C:\Users\Alex\Projekte\demo-app
git status
git add README.md
git commit -m "README ergänzen"
```

Ab `git status` sind die Befehle identisch. Du musst also nicht für jede Shell eine neue Git-Sprache lernen.

Unterschiede bei Variablen und Umgebungsvariablen

Shells unterscheiden sich deutlich darin, wie sie Variablen schreiben und lesen. Das wird wichtig, sobald du Konfigurationen prüfst, Tokens temporär setzt oder Skripte verwendest.

Git Bash

```
name="Alex"
echo "$name"

echo "$HOME"
echo "$PATH"
```

Eine Umgebungsvariable wird für einen einzelnen Befehl gesetzt:

```
GIT_EDITOR=vim git commit
```

PowerShell

```
$name = "Alex"
Write-Output $name

$HOME
$env:PATH
```

Eine Umgebungsvariable wird beispielsweise so gesetzt:

```
$env:GIT_EDITOR = "notepad"
git commit
```

CMD

```
set name=Alex
echo %name%

echo %USERPROFILE%
echo %PATH%
```

Für eine einzelne CMD-Sitzung:

```
set GIT_EDITOR=notepad
git commit
```

> **Wichtig:** Beispielbefehle für Bash, PowerShell und CMD solltest du nicht unbesehen mischen. Besonders Variablen, Anführungszeichen, Schleifen und Pipes verhalten sich unterschiedlich.

Anführungszeichen richtig einsetzen

Datei- und Ordnernamen mit Leerzeichen sind unter Windows häufig. Beispiele sind `C:\Program Files` oder ein Projektordner wie `Meine Projekte`.

In allen drei Konsolen sind doppelte Anführungszeichen für solche Pfade die sichere Standardwahl:

```
cd "/c/Users/Alex/Meine Projekte/demo-app"
```

```
cd "C:\Users\Alex\Meine Projekte\demo-app"
```

```
cd /d "C:\Users\Alex\Meine Projekte\demo-app"
```

Für Git-Commit-Nachrichten verwendest du ebenfalls meist doppelte Anführungszeichen:

```
git commit -m "Fehler beim Login beheben"
```

Bei komplexen Sonderzeichen, Variablen oder mehrzeiligen Skripten unterscheiden sich die Regeln der Shells. Für normale Git-Kommandos reichen doppelte Anführungszeichen in der Regel aus.

Welche Shell solltest du verwenden?

Für diesen Kurs bietet sich folgende praktische Entscheidung an:

Git Bash als Standard für Git lernen

Nutze **Git Bash**, wenn du die Git-Kommandozeile systematisch lernen möchtest. Die meisten Beispiele in diesem Kurs lassen sich dort direkt übernehmen. Außerdem ähneln die Befehle vielen Server-, CI- und Open-Source-Umgebungen.

PowerShell für Windows-nahe Arbeit

Nutze **PowerShell**, wenn du Windows administrierst, Dateien und Prozesse automatisierst oder bereits mit PowerShell vertraut bist. Git funktioniert dort vollständig; achte nur bei zusätzlichen Shell-Befehlen auf die abweichende Syntax.

CMD nur bei konkretem Bedarf

Nutze **CMD**, wenn ein älteres Skript oder eine bestimmte Anleitung dies verlangt. Für einen neuen Git-Workflow ist CMD normalerweise nicht die erste Wahl.

> **Empfehlung für den Einstieg:** Arbeite in den nächsten Kapiteln überwiegend mit **Git Bash**. Öffne PowerShell bewusst gelegentlich mit, damit du die Pfad- und Syntaxunterschiede kennenlernst.

Windows Terminal als gemeinsame Oberfläche

Windows Terminal ist keine eigene Shell. Es ist eine moderne Terminal-Anwendung, in der du unterschiedliche Shells als Tabs oder Profile öffnen kannst:

- Git Bash
- PowerShell
- Eingabeaufforderung
- Windows Subsystem for Linux, falls installiert
- SSH-Verbindungen zu Servern

Damit musst du nicht zwischen verschiedenen Anwendungen wechseln. Du kannst etwa einen Git-Bash-Tab für Git-Kommandos und einen PowerShell-Tab für Windows-Aufgaben verwenden.

Die Shell erkennst du am Prompt:

Prompt-Beispiel	Aktive Shell
`alex@PC MINGW64 ~`	Git Bash
`PS C:\Users\Alex>`	PowerShell
`C:\Users\Alex>`	CMD

Diese Unterscheidung ist wichtig, wenn ein kopierter Befehl einen Fehler ausgibt. Prüfe dann zuerst, für welche Shell der Befehl geschrieben wurde.

Git in PhpStorm und in der Konsole

PhpStorm verwendet Git nicht über eine eigene, abweichende Git-Variante. Die IDE ruft die installierte Git-Anwendung im Hintergrund auf. Deshalb gelten dieselben Konzepte:

- Repository

- Branch
- Staging Area
- Commit
- Remote
- Fetch, Pull und Push

Die Konsole bleibt trotzdem wertvoll:

- Sie zeigt Git-Ausgaben vollständig und unmittelbar.
- Sie eignet sich für seltene oder fortgeschrittene Optionen.
- Viele Fehler lassen sich in der Konsole präziser untersuchen.
- Git-Dokumentationen enthalten fast immer CLI-Beispiele.
- Du verstehst besser, welche Aktion eine IDE-Schaltfläche tatsächlich ausführt.

PhpStorm kann ein integriertes Terminal mit Git Bash, PowerShell oder CMD öffnen. Welche Shell dort startet, legst du später in den Terminal-Einstellungen fest.

Kurzer Selbsttest

Öffne nacheinander Git Bash, PowerShell und CMD. Führe in jeder Konsole diese Befehle aus:

```
git --version
git config --global user.name
git config --global user.email
```

Prüfe danach den aktuellen Ordner:

```
pwd
```

```
Get-Location
```

```
cd
```

Wenn `git --version` in einer Shell funktioniert, in einer anderen aber nicht, ist Git wahrscheinlich nicht in allen Sitzungen über den PATH verfügbar. Schließe zunächst alle geöffneten Terminals und starte sie neu. Besteht das Problem weiter, wird die PATH-Konfiguration im nächsten Abschnitt gezielt geprüft.

Merksätze

- **Git ist in allen Shells dasselbe; die Umgebung darum herum ist unterschiedlich.**
- **Git Bash** ist die beste Standardwahl, um Git anhand vieler Anleitungen zu lernen.
- **PowerShell** ist die stärkste Wahl für Windows-Automatisierung.
- **CMD** ist vor allem für Kompatibilität mit älteren Windows-Skripten relevant.
- Pfade, Variablen und Dateibefehle sind shellabhängig.
- Bei Fehlermeldungen solltest du immer prüfen, *welche Shell* gerade aktiv ist.

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