Windows Terminal konfigurieren

Windows Terminal bündelt PowerShell, Eingabeaufforderung, Git Bash und weitere Shells in einer modernen Oberfläche. Für die Git-Arbeit unter Windows 11 ist es besonders praktisch: Du kannst mehrere Terminals in Tabs öffnen, Arbeitsverzeichnisse gezielt starten und ein einheitliches Profil für Git Bash anlegen.

> **Ziel:** Nach diesem Abschnitt startest du Git Bash bequem im Windows Terminal, öffnest Repositories direkt im passenden Ordner und kennst die wichtigsten Einstellungen für eine zuverlässige tägliche Arbeit.

Windows Terminal installieren und aktualisieren

Auf aktuellen Windows-11-Systemen ist Windows Terminal meist bereits installiert. Öffne das Startmenü und suche nach **Windows Terminal**.

Falls es fehlt oder aktualisiert werden soll:

1. Öffne den **Microsoft Store**.
2. Suche nach **Windows Terminal**.
3. Wähle **Installieren** oder **Aktualisieren**.

Alternativ lässt sich Windows Terminal über `winget` installieren:

```
winget install --id Microsoft.WindowsTerminal
```

Prüfe anschließend die installierte Version:

```
wt --version
```

Die Ausgabe enthält beispielsweise eine Versionsnummer wie `1.22.x`. Entscheidend ist nicht die konkrete Nummer, sondern dass der Befehl ohne Fehlermeldung funktioniert.

Die Oberfläche verstehen

Windows Terminal verwaltet jede Kommandozeilenumgebung als **Profil**. Ein Profil definiert unter anderem:

- welche Anwendung gestartet wird,
- in welchem Ordner sie startet,
- welches Symbol, Farbschema und Schriftbild verwendet wird,
- ob das Profil als Administrator geöffnet wird,
- welche Umgebung beim Öffnen eines neuen Tabs standardmäßig erscheint.

Typische automatisch erkannte Profile sind:

Profil	Typischer Zweck
Windows PowerShell	Klassische Windows-Verwaltung und bestehende Skripte
PowerShell	Moderne PowerShell-Version, falls installiert
Eingabeaufforderung	Kompatibilität mit älteren `cmd.exe`-Befehlen
Git Bash	Git-Befehle und Unix-ähnliche Werkzeuge aus Git for Windows
Azure Cloud Shell	Arbeit mit Azure, falls eingerichtet
Ubuntu oder andere WSL-Distributionen	Linux-Umgebung über Windows Subsystem for Linux

Für diesen Kurs sind vor allem **Git Bash** und **PowerShell** relevant:

- **Git Bash** eignet sich hervorragend für Git-Tutorials und viele plattformübergreifende Befehle.
- **PowerShell** ist nützlich für Windows-spezifische Aufgaben, etwa Dateisystem-, Netzwerk- oder Verwaltungsbefehle.

Git Bash als Terminalprofil prüfen

Wenn Git for Windows installiert wurde, erkennt Windows Terminal Git Bash in der Regel automatisch. Öffne Windows Terminal und klicke auf den Pfeil neben dem Pluszeichen **+**. In der Liste sollte **Git Bash** erscheinen.

Wähle den Eintrag aus. Ein Git-Bash-Tab zeigt typischerweise einen Prompt ähnlich diesem:

```
max@PC MINGW64 ~
$
```

Prüfe darin Git:

```
git --version
```

Erwartet wird eine Ausgabe ähnlich wie:

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

Wenn Git Bash nicht in der Profilliste erscheint, schließe Windows Terminal vollständig und öffne es erneut. Die automatische Profilerkennung erfolgt beim Start.

Falls das Profil weiterhin fehlt, kannst du es manuell anlegen.

Ein Git-Bash-Profil manuell erstellen

Öffne die Einstellungen über eine der folgenden Möglichkeiten:

- Klicke auf den Pfeil neben **+** und wähle **Einstellungen**.
- Verwende die Tastenkombination `Strg` + `,`.

Wähle links **Neues Profil hinzufügen** und dann **Neue leere Konfiguration**. Trage anschließend diese Werte ein:

Einstellung	Empfohlener Wert
Name	`Git Bash`
Befehlszeile	`C:\Program Files\Git\bin\bash.exe -l -i`
Startverzeichnis	`%USERPROFILE%`
Symbol	`C:\Program Files\Git\mingw64\share\git\git-for-windows.ico`

Die Optionen `-l -i` haben einen Zweck:

- `-l` startet Bash als *Login-Shell*. Dadurch werden die üblichen Git-for-Windows-Startdateien geladen.
- `-i` startet eine interaktive Shell, die für die tägliche Arbeit mit Eingaben vorgesehen ist.

Der Installationspfad kann abweichen. Prüfe im Explorer, ob Git tatsächlich unter `C:\Program Files\Git` installiert ist. Bei einer benutzerbezogenen Installation liegt Git häufig unter folgendem Pfad:

```
C:\Users\<Benutzername>\AppData\Local\Programs\Git
```

Passe den Pfad im Profil entsprechend an.

Profil über die JSON-Datei konfigurieren

Die grafischen Einstellungen reichen für die meisten Fälle aus. Fortgeschrittene können jedoch direkt die JSON-Konfiguration bearbeiten. Öffne dazu in den Einstellungen unten links **JSON-Datei öffnen**.

Ein Git-Bash-Profil innerhalb von `profiles` → `list` kann beispielsweise so aussehen:

```
{
    "name": "Git Bash",
    "commandline": "C:\\Program Files\\Git\\bin\\bash.exe -l -i",
    "startingDirectory": "%USERPROFILE%",
    "icon": "C:\\Program Files\\Git\\mingw64\\share\\git\\git-for-windows.ico",
    "hidden": false
}
```

Beachte dabei zwei wichtige Details:

1. In JSON müssen Backslashes als `\\` geschrieben werden.
2. Nach dem Speichern lädt Windows Terminal die Änderungen normalerweise automatisch. Andernfalls öffnest du ein neues Fenster.

> **Hinweis:** Bearbeite die JSON-Datei nur gezielt. Ein fehlendes Komma oder eine zusätzliche Klammer kann verhindern, dass Windows Terminal die Konfiguration lädt.

Ein sinnvolles Startverzeichnis festlegen

Ein Terminal startet standardmäßig oft im Benutzerprofil, also etwa hier:

```
C:\Users\max
```

Für Git ist das eine gute allgemeine Ausgangsbasis. Du kannst dort beispielsweise einen Ordner für alle Entwicklungsprojekte anlegen:

```
mkdir $HOME\Projekte
```

In Git Bash lautet der vergleichbare Befehl:

```
mkdir -p ~/Projekte
```

Wenn du deine Repositories immer in diesem Ordner speicherst, setze für dein Git-Bash-Profil als **Startverzeichnis**:

```
%USERPROFILE%\Projekte
```

Damit öffnet jeder neue Git-Bash-Tab direkt im Projektbereich.

Ein Startverzeichnis sollte nicht auf ein einzelnes Repository festgelegt werden, sofern du parallel an mehreren Projekten arbeitest. Ein zentraler Ordner wie `Projekte`, `Code` oder `Repositories` ist flexibler.

Git Bash als Standardprofil festlegen

Wenn Git Bash deine primäre Arbeitsumgebung für Git sein soll, kannst du sie als Standardprofil konfigurieren.

1. Öffne die **Einstellungen** von Windows Terminal.
2. Wähle links **Start**.
3. Setze bei **Standardprofil** den Eintrag **Git Bash**.
4. Speichere die Einstellung.

Ab jetzt startet Windows Terminal beim Öffnen direkt mit Git Bash.

Das ist eine Komfortentscheidung, keine technische Notwendigkeit. Git funktioniert ebenso in PowerShell, sofern Git korrekt im `PATH` verfügbar ist. Die Wahl des Standardprofils sollte zu deinem Arbeitsstil passen:

- Wähle **Git Bash**, wenn du überwiegend Kursbefehle, Bash-Skripte oder Unix-Programme verwendest.
- Wähle **PowerShell**, wenn du häufig Windows-Automatisierung und PowerShell-Skripte nutzt.

Repository direkt im Windows Terminal öffnen

Besonders effizient ist das Öffnen eines Terminals direkt im Ordner eines Projekts.

Über den Explorer

Navigiere im Datei-Explorer zu deinem Repository. Klicke mit der rechten Maustaste auf einen freien Bereich im Ordner und wähle:

```
In Terminal öffnen
```

Windows Terminal startet im aktuellen Verzeichnis. Welches Profil erscheint, hängt vom festgelegten Standardprofil ab.

Mit der Adressleiste des Explorers

Klicke in die Adressleiste des Explorers, gib Folgendes ein und bestätige mit `Enter`:

```
wt
```

Dadurch öffnet sich Windows Terminal im aktuell angezeigten Ordner.

Mit dem Befehl `wt`

Der Befehl `wt` steuert Windows Terminal aus anderen Konsolen oder Skripten. Um Git Bash in einem bestimmten Projektordner zu öffnen, verwende beispielsweise:

```
wt -p "Git Bash" -d C:\Users\max\Projekte\mein-projekt
```

Die Optionen bedeuten:

Option	Bedeutung
`-p "Git Bash"`	Startet das Profil mit dem Namen „Git Bash“
`-d <Pfad>`	Legt das Startverzeichnis fest

Wenn der Profilname Leerzeichen enthält, muss er in Anführungszeichen stehen.

In Git Bash kannst du das aktuelle Repository in einem neuen Git-Bash-Tab öffnen:

```
wt -p "Git Bash" -d .
```

Der Punkt `.` steht für das aktuelle Verzeichnis.

Tabs, Bereiche und Arbeitsabläufe nutzen

Windows Terminal kann mehrere Shells gleichzeitig verwalten. Das ist für Git besonders nützlich, weil du verschiedene Aufgaben trennen kannst.

Ein sinnvoller Ablauf könnte so aussehen:

Tab oder Bereich	Aufgabe
Git Bash im Feature-Branch	Änderungen bearbeiten, `git status`, `git add`, `git commit`
Zweiter Git-Bash-Tab im selben Repository	Logs, Diffs oder Vergleiche prüfen
PowerShell	Windows-spezifische Befehle und Skripte ausführen
PhpStorm-Terminal	Befehle direkt im Kontext der IDE ausführen

Erstelle einen neuen Tab mit:

```
Strg + Umschalt + T
```

Ein Bereich teilt das aktuelle Terminalfenster in mehrere nebeneinanderliegende Konsolen. Das Kontextmenü des Tabs bietet dafür je nach Version Optionen wie **Bereich teilen**. Alternativ lassen sich Bereiche über die Befehlspalette anlegen.

> **Praxisregel:** Verwende getrennte Tabs oder Bereiche nicht als Ersatz für getrennte Arbeitsverzeichnisse. Wenn du gleichzeitig an zwei Branches arbeiten

> möchtest, nutze später `git worktree` oder klone das Repository getrennt. Zwei Terminals im selben Arbeitsverzeichnis sehen immer denselben Git-Zustand.

Darstellung für Git-Ausgaben verbessern

Git-Diffs, Logs und Konfliktmarker profitieren von einer gut lesbaren Schrift und ausreichendem Kontrast.

Öffne die Einstellungen und wähle dein **Git-Bash-Profil**. Unter **Darstellung** sind diese Werte empfehlenswert:

Einstellung	Empfehlung	Begründung
Schriftart	Cascadia Mono, Consolas oder JetBrains Mono	Zeichen und Einrückungen bleiben klar unterscheidbar
Schriftgröße	11 bis 14	Gut lesbar bei Diffs und längeren Logs
Zeilenhöhe	Standard oder leicht erhöht	Erleichtert das Lesen dichter Ausgaben
Farbschema	Kontrastreiches dunkles oder helles Schema	Konfliktmarker und Diff-Farben bleiben erkennbar
Transparenz	Dezent oder deaktiviert	Text sollte stets gut lesbar bleiben

Eine **Monospace-Schriftart** ist wichtig, weil jedes Zeichen dieselbe Breite besitzt. Nur so bleiben Einrückungen, Tabellenähnliche Ausgaben und Diff-Strukturen zuverlässig ausgerichtet.

Vermeide sehr kleine Schrift und starke Transparenz. Beides erschwert insbesondere die Arbeit mit langen Commit-Hashes, Dateipfaden und Konfliktmarkern.

Kopieren und Einfügen sicher verwenden

In Windows Terminal gelten üblicherweise moderne Tastenkombinationen:

Aktion	Tastenkombination

Kopieren	`Strg` + `C`, wenn kein Prozess läuft oder Text markiert ist
Einfügen	`Strg` + `V`
Alles auswählen	`Strg` + `A`
Suchen	`Strg` + `Umschalt` + `F`
Neuer Tab	`Strg` + `Umschalt` + `T`

Bei Konsolenprogrammen hat `Strg` + `C` eine zweite Bedeutung: Es kann einen laufenden Prozess abbrechen. Das ist beispielsweise bei einem versehentlich gestarteten, lange laufenden Befehl sinnvoll, kann aber auch einen gerade aktiven Git-Vorgang unterbrechen.

Markiere daher zum Kopieren zuerst den gewünschten Text. Windows Terminal erkennt die Auswahl und kopiert dann statt den Prozess abzubrechen.

> ⚠ **Sicherheitsregel:** Füge Git-Befehle aus Webseiten, Chats oder Tickets nicht blind ein. Lies jeden Befehl vollständig, bevor du ihn ausführst — besonders Befehle mit `reset --hard`, `clean`, `push --force`, `rm` oder Pipe-Weiterleitungen.

Nützliche Profileinstellungen für Git Bash

Für die tägliche Arbeit reichen wenige, bewusst gewählte Einstellungen.

Administratorrechte nicht standardmäßig aktivieren

Git benötigt normalerweise **keine Administratorrechte**. Starte Git Bash deshalb nicht dauerhaft als Administrator.

Ein erhöhtes Terminal kann zu unerwünschten Dateibesitzern oder Berechtigungen führen und erhöht bei versehentlichen Befehlen den möglichen Schaden. Öffne ein Administrator-Terminal nur für konkrete Systemaufgaben, etwa bei bestimmten Installationen oder globalen Konfigurationen.

Terminaltitel sinnvoll setzen

Windows Terminal zeigt den Titel eines Tabs an. Git Bash setzt oft automatisch den aktuellen Ordner oder Hostnamen. Das ist normalerweise ausreichend.

Bei vielen offenen Tabs hilft es, Tabs eindeutig zu benennen. Nutze dafür im Tab-Kontextmenü die Funktion zum Umbenennen oder lege unterschiedliche Profile für verschiedene Aufgaben an, beispielsweise:

- `Git Bash`
- `Git Bash – Projekte`
- `PowerShell – Verwaltung`

Vermeide jedoch zu viele fast identische Profile. Eine übersichtliche Profilliste ist wertvoller als maximale Anpassbarkeit.

Quake-Modus als optionale Schnellkonsole

Windows Terminal bietet einen optionalen *Quake-Modus*: Ein Terminal gleitet vom oberen Bildschirmrand ein und aus. Das kann nützlich sein, wenn du sehr häufig kurze Git-Abfragen wie `git status` oder `git log --oneline` ausführst.

Die Funktion ist rein optional. Für Einsteiger ist ein normales Terminalfenster meist übersichtlicher, weil mehrere Tabs und längere Ausgaben leichter nachvollziehbar bleiben.

Git-Farben im Terminal prüfen

Git verwendet in Git Bash standardmäßig Farben für Statusmeldungen, Branches und Diffs. Prüfe dies in einem Repository:

```
git status
```

Typische Farbbedeutungen sind:

- **Rot:** Änderungen sind noch nicht für den Commit vorgemerkt.
- **Grün:** Änderungen liegen bereits in der Staging Area.
- **Weitere Farben:** Abhängig vom Git-Theme und vom jeweiligen Befehl.

Falls Git keine Farben zeigt, prüfe zunächst die Konfiguration:

```
git config --global --get color.ui
```

Eine sinnvolle Einstellung ist:

```
git config --global color.ui auto
```

`auto` bedeutet: Git verwendet Farben, wenn die Ausgabe in einem interaktiven Terminal erscheint, aber nicht zwingend bei einer Umleitung in eine Datei.

Grundlegende Funktionsprüfung

Führe nach der Konfiguration diese Prüfung in einem Git-Bash-Tab aus:

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

Die Befehle prüfen nacheinander:

1. Git ist erreichbar.
2. Dein Git-Autorenname ist gesetzt.
3. Deine Git-E-Mail-Adresse ist gesetzt.
4. Du befindest dich im erwarteten Arbeitsverzeichnis.

Erstelle anschließend testweise einen Projektordner:

```
mkdir -p ~/Projekte/terminal-test
cd ~/Projekte/terminal-test
git init
git status
```

Eine erfolgreiche Ausgabe enthält einen Hinweis auf ein neu initialisiertes Repository und zeigt einen sauberen Arbeitsstand an. Lösche den Testordner später nur dann, wenn du ihn nicht für die folgenden Übungen verwenden möchtest:

```
cd ..
rm -rf terminal-test
```

> ⚠ `rm -rf` löscht Dateien ohne Papierkorb. Prüfe mit `pwd` und `ls` immer zuerst, in welchem Ordner du dich befindest. Auf Windows ist Löschen über den Explorer für Einsteiger oft sicherer.

Häufige Probleme und Lösungen

Git Bash fehlt in der Profilliste

Mögliche Ursache: Git for Windows wurde nach Windows Terminal installiert oder die automatische Erkennung ist fehlgeschlagen.

Lösung:

1. Schließe alle Windows-Terminal-Fenster.
2. Starte Windows Terminal erneut.
3. Prüfe den Installationspfad von Git.
4. Lege bei Bedarf das Profil manuell mit `bash.exe -l -i` an.

`git` wird in Git Bash nicht gefunden

Mögliche Ursache: Die Git-for-Windows-Installation ist unvollständig oder das falsche Programm wird als Profil gestartet.

Lösung: Prüfe den Pfad in der Profilkonfiguration. Verwende bevorzugt:

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

Starte danach einen neuen Tab und teste erneut:

```
git --version
```

Sonderzeichen oder Umlaute werden falsch dargestellt

Mögliche Ursache: Eine ungeeignete Schriftart, ältere Konsoleneinstellungen oder eine problematische Zeichencodierung.

Lösung:

1. Wähle im Profil eine moderne Schrift wie **Cascadia Mono**.
2. Aktualisiere Windows Terminal und Git for Windows.
3. Öffne ein neues Git-Bash-Fenster.
4. Prüfe zunächst, ob Dateinamen mit Umlauten im Explorer korrekt gespeichert sind.

Windows Terminal verwendet standardmäßig Unicode-fähige Einstellungen. Vermeide deshalb veraltete Konsolenfenster, wenn moderne Git-Ausgaben und Dateinamen korrekt dargestellt werden sollen.

Terminal startet im falschen Ordner

Mögliche Ursache: Das Profil besitzt ein fest eingetragenes Startverzeichnis oder die Option „In Terminal öffnen“ wurde aus einem anderen Ordner aufgerufen.

Lösung: Prüfe im Git-Bash-Profil die Einstellung **Startverzeichnis**. Für einen neutralen Startpunkt eignet sich:

```
%USERPROFILE%\Projekte
```

PowerShell findet Git, Git Bash aber nicht — oder umgekehrt

Mögliche Ursache: Unterschiedliche `PATH`-Konfigurationen oder eine unvollständige Git-Installation.

Lösung: Vergleiche die Befehle in beiden Umgebungen:

```
git --version
```

```
git --version
```

Für den Kurs genügt es zunächst, wenn Git Bash zuverlässig funktioniert. Die systemweite `PATH`-Konfiguration wird im nächsten Abschnitt gesondert geprüft.

Empfehlungen für den weiteren Kurs

Richte Windows Terminal bewusst schlicht und stabil ein:

- Verwende **Git Bash** als gut erreichbares Profil.
- Starte standardmäßig in einem zentralen Projektordner.
- Nutze eine gut lesbare Monospace-Schrift.

- Öffne Repositories über **„In Terminal öffnen“** direkt im jeweiligen Ordner.
- Arbeite normalerweise **ohne Administratorrechte**.
- Prüfe vor riskanten Befehlen stets `git status` und das aktuelle Verzeichnis.

Damit steht eine verlässliche Kommandozeilenumgebung bereit. Im nächsten Schritt prüfst du, ob Git systemweit über die `PATH`-Variable erreichbar ist und wie sich unterschiedliche Shells dabei verhalten.

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