Zeilenenden mit core.autocrlf verstehen

Warum Zeilenenden wichtig sind

Textdateien bestehen aus einzelnen Zeilen. Damit ein Programm erkennt, wo eine Zeile endet, verwendet das Betriebssystem ein *Zeilenende*.

Die zwei wichtigsten Varianten sind:

- **LF** (*Line Feed*, `\n`) – üblich unter Linux und macOS
- **CRLF** (*Carriage Return* + *Line Feed*, `\r\n`) – traditionell unter Windows

Eine PHP-Datei kann fachlich identisch sein, obwohl sie unterschiedliche Zeilenenden verwendet. Git erkennt diese Unterschiede jedoch auf Dateiebene. Ohne klare Regeln können dadurch Diffs entstehen, die scheinbar jede Zeile einer Datei geändert zeigen.

> **Beispiel:** Eine Datei enthält 300 Zeilen PHP-Code. Ein Teammitglied speichert sie mit LF statt CRLF. Inhaltlich wurde nichts verändert, Git zeigt aber möglicherweise 300 entfernte und 300 hinzugefügte Zeilen an.

Das erschwert Reviews, erhöht Konfliktrisiken und verdeckt echte Änderungen.

Das Grundprinzip von `core.autocrlf`

Die Git-Konfiguration `core.autocrlf` steuert, ob Git Zeilenenden beim **Auschecken** und **Einchecken** automatisch umwandelt.

Git unterscheidet dabei zwei Richtungen:

1. **Checkout**: Git schreibt Dateien aus dem Repository in dein Arbeitsverzeichnis.
2. **Commit**: Git übernimmt Inhalte aus dem Arbeitsverzeichnis in die Repository-Historie.

Das gewünschte Ziel in vielen plattformübergreifenden Projekten lautet:

- Im Repository liegen Textdateien einheitlich mit **LF** vor.
- Unter Windows dürfen Arbeitskopien bei Bedarf **CRLF** verwenden.

Der typische Ablauf sieht so aus:

```
Repository mit LF
        ↓ Checkout
Arbeitsverzeichnis unter Windows mit CRLF
        ↓ git add und Commit
Repository wieder mit LF
```

Damit bleibt die Historie auf allen Plattformen konsistent, während lokale Werkzeuge im vertrauten Zeilenende arbeiten können.

Die drei Werte von `core.autocrlf`

Prüfe zunächst die aktuell wirksame Einstellung:

```
git config --show-origin --get core.autocrlf
```

Die Option `--show-origin` zeigt zusätzlich, *aus welcher Konfigurationsdatei* der Wert stammt. Das ist hilfreich, wenn globale und lokale Einstellungen voneinander abweichen.

`core.autocrlf=true`

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

Diese Einstellung ist für viele klassische Windows-Setups geeignet.

Git verhält sich dann grundsätzlich so:

- Beim **Checkout**: LF wird zu CRLF umgewandelt.
- Beim **Commit**: CRLF wird wieder zu LF normalisiert.

Die Repository-Historie bleibt dadurch bei Textdateien typischerweise LF-basiert.

Empfehlung für Einsteiger unter Windows 11: Wenn ein Projekt noch keine verbindliche `.gitattributes`-Datei besitzt und du mit klassischen Windows-Werkzeugen arbeitest, ist `true` ein sinnvoller Ausgangspunkt.

`core.autocrlf=input`

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

Diese Variante normalisiert Zeilenenden nur beim Hinzufügen zum Repository:

- Beim **Checkout**: Git verändert die Zeilenenden nicht.
- Beim **Commit**: CRLF wird zu LF normalisiert.

Das bedeutet: Dateien bleiben im Arbeitsverzeichnis so, wie dein Editor sie speichert. CRLF wird beim Commit dennoch als LF in Git abgelegt.

Diese Einstellung wird häufig unter Linux und macOS verwendet. Sie kann unter Windows ebenfalls sinnvoll sein, wenn PhpStorm und das Team konsequent LF verwenden.

`core.autocrlf=false`

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

Git führt dann keine automatische Konvertierung über diese Einstellung durch.

Das ist nicht automatisch falsch. Es verlangt aber, dass ein Projekt seine Zeilenenden auf andere Weise zuverlässig regelt – idealerweise mit einer versionierten `.gitattributes`-Datei. Ohne Projektregeln kann diese Einstellung dazu führen, dass LF und CRLF unkontrolliert in die Historie gelangen.

Empfehlung für dieses Windows-11-Setup

Für einen sicheren Einstieg kannst du global Folgendes konfigurieren:

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

Kontrolliere anschließend den gespeicherten Wert:

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

Die erwartete Ausgabe lautet:

```
true
```

Diese Einstellung gilt für alle Repositories deines Windows-Benutzerkontos, sofern ein Repository sie nicht lokal überschreibt.

> ⚠ **Wichtig:** `core.autocrlf` ist eine lokale Arbeitsumgebungs-Einstellung. Sie wird normalerweise nicht mit dem Repository geteilt. Teamweit reproduzierbare Regeln gehören später in eine versionierte `.gitattributes`-Datei.

Globale, lokale und wirksame Konfiguration unterscheiden

Git kann denselben Konfigurationswert auf mehreren Ebenen speichern:

- **Systemweit**: für alle Benutzer des Computers
- **Global**: für dein Windows-Benutzerkonto
- **Lokal**: nur für ein bestimmtes Repository

Prüfe alle gefundenen Werte:

```
git config --show-origin --get-all core.autocrlf
```

Ein Beispiel könnte so aussehen:

```
file:C:/Users/alex/.gitconfig  true
file:.git/config               false
```

In diesem Fall überschreibt die lokale Einstellung `false` den globalen Wert `true`.

Den lokalen Wert prüfst du innerhalb eines Repositorys mit:

```
git config --local --get core.autocrlf
```

Du kannst ihn gezielt setzen:

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

Oder wieder entfernen, damit die globale Einstellung greift:

```
git config --local --unset core.autocrlf
```

Warnungen zu Zeilenenden richtig verstehen

Beim Hinzufügen von Dateien kann Git Warnungen ausgeben, etwa:

```
warning: in the working copy of 'src/Calculator.php', LF will be replaced by CRLF the next time Git touches it
```

Diese Meldung bedeutet nicht zwingend, dass ein Fehler vorliegt. Git informiert dich darüber, dass die Datei in deiner Windows-Arbeitskopie künftig mit CRLF erscheinen wird.

Bei `core.autocrlf=true` ist das erwartbares Verhalten:

- Git speichert die Datei intern beziehungsweise im Commit als LF.
- Git kann sie bei einem späteren Checkout als CRLF in das Arbeitsverzeichnis schreiben.

Problematisch wird es erst, wenn ein Commit ausschließlich aus ungewollten Zeilenende-Änderungen besteht oder wenn das Team keine einheitliche Regel verfolgt.

Den Zustand einer Datei prüfen

Mit Git kannst du analysieren, welche Zeilenenden Git für eine Datei erkennt:

```
git ls-files --eol src/Calculator.php
```

Eine mögliche Ausgabe lautet:

```
i/lf    w/crlf  attr/                src/Calculator.php
```

Die Teile haben folgende Bedeutung:

- `i/lf`: Im **Index** liegt die Datei mit LF vor.
- `w/crlf`: Im **Working Tree**, also im Arbeitsverzeichnis, liegt sie mit CRLF vor.
- `attr/`: Für diese Datei greift derzeit kein besonderes Attribut.

Diese Kombination ist unter Windows mit `core.autocrlf=true` normal.

Wenn im Arbeitsverzeichnis LF verwendet wird, kann die Ausgabe stattdessen so aussehen:

```
i/lf    w/lf    attr/                src/Calculator.php
```

Prüfe alle verfolgten Dateien mit:

```
git ls-files --eol
```

Das ist besonders nützlich, wenn du vor einem Pull Request unerwartet große Diffs bemerkst.

Zeilenenden in PhpStorm einstellen

PhpStorm kann Zeilenenden unabhängig von Git speichern. Deshalb sollten IDE und Git nicht gegeneinander arbeiten.

Den Zeilenend-Typ der aktuell geöffneten Datei erkennst und änderst du in PhpStorm normalerweise über die Statusleiste am unteren Fensterrand. Dort steht beispielsweise:

```
CRLF
```

oder:

```
LF
```

Ein Klick darauf erlaubt, das Zeilenende für die aktuelle Datei zu ändern.

Für neue Dateien kannst du die Standardvorgabe in den Einstellungen festlegen:

```
File → Settings → Editor → Code Style → Line separator
```

Für moderne PHP-Projekte ist **LF** oft die robusteste teamweite Wahl – auch unter Windows. Wenn dein Projekt bereits `.gitattributes` verwendet, hat dessen Regelwerk Vorrang für die Git-Normalisierung.

> **Praxisregel:** Ändere Zeilenenden nicht beiläufig während fachlicher Änderungen. Wenn eine Umstellung notwendig ist, führe sie als separaten, klar benannten Commit durch.

Warum `.gitattributes` langfristig besser ist

`core.autocrlf` ist benutzerabhängig: Jedes Teammitglied kann einen anderen Wert gesetzt haben. Eine Datei namens `.gitattributes` liegt dagegen im Repository und wird gemeinsam versioniert.

Eine einfache, robuste Grundlage lautet:

```
* text=auto
```

Damit erkennt Git Textdateien automatisch und normalisiert sie beim Einchecken grundsätzlich auf LF.

Für ein PHP-Projekt kann eine präzisere Variante sinnvoll sein:

```
*.php  text eol=lf
*.xml  text eol=lf
*.yml  text eol=lf
*.yaml text eol=lf
*.json text eol=lf
*.md   text eol=lf
*.sh   text eol=lf

*.bat  text eol=crlf
*.cmd  text eol=crlf
```

```
*.png  binary
*.jpg  binary
*.jpeg binary
*.gif  binary
*.zip  binary
*.pdf  binary
```

Diese Regeln bedeuten:

- PHP-, Konfigurations-, Markdown- und Shell-Dateien werden mit LF behandelt.
- Windows-Batchdateien werden gezielt mit CRLF ausgecheckt.
- Binärdateien werden nicht als Text interpretiert und nicht umgewandelt.

Die detaillierte Arbeit mit `.gitattributes` folgt später im Kurs. Für die erste Einrichtung genügt die Erkenntnis:

> **`core.autocrlf` regelt deine lokale Git-Umgebung; `.gitattributes` definiert verbindliche Regeln für das Projekt.**

Bereits vorhandene Projekte nicht unbedacht umwandeln

Wenn du `core.autocrlf` in einem bestehenden Repository erstmals setzt, sollte Git nicht automatisch einen riesigen Änderungsblock erzeugen. Prüfe deshalb immer zuerst:

```
git status
```

Untersuche auffällige Änderungen:

```
git diff --check
```

Und sieh dir den tatsächlichen Diff an:

```
git diff
```

Wenn Git plötzlich sehr viele geänderte Zeilen anzeigt, obwohl du keinen Code bearbeitet hast, könnten Zeilenenden die Ursache sein.

In einem Teamprojekt solltest du dann **nicht sofort committen**. Kläre stattdessen:

1. Gibt es bereits eine `.gitattributes`-Datei?
2. Welche Zeilenenden verwendet der Hauptbranch?
3. Welche Teamregel gilt für PHP-, Shell- und Batchdateien?
4. Soll eine einmalige, abgestimmte Normalisierung erfolgen?

Eine absichtliche Normalisierung aller Textdateien ist möglich, sollte aber als eigener Teamentscheid behandelt werden. Sie verändert potenziell viele Dateien und erschwert sonst die parallele Arbeit an Feature-Branches.

Typische Situationen und passende Reaktionen

Jede Zeile einer Datei erscheint geändert

Wahrscheinliche Ursache: Die Datei wurde mit einem anderen Zeilenende gespeichert.

Prüfe:

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

Verschwindet der Großteil des Diffs mit dieser Option, handelt es sich wahrscheinlich um Zeilenende-Unterschiede.

Prüfe zusätzlich:

```
git ls-files --eol pfad/zur/datei.php
```

Reaktion: Verwirf die unbeabsichtigte Änderung oder stelle das richtige Zeilenende in PhpStorm wieder her. Committe keine reine Zeilenende-Änderung zusammen mit einer fachlichen Änderung.

Eine Shell-Datei funktioniert auf Linux nicht

Wahrscheinliche Ursache: Die Datei hat CRLF statt LF. Shells können das unsichtbare Carriage-Return-Zeichen als Teil eines Befehls interpretieren.

Reaktion: Speichere `.sh`-Dateien mit LF und sichere dies mit `.gitattributes` ab:

```
*.sh text eol=lf
```

Eine Batchdatei funktioniert unter Windows nicht wie erwartet

Wahrscheinliche Ursache: Die Datei wurde mit LF gespeichert, obwohl das verwendete Windows-Werkzeug CRLF erwartet.

Reaktion: Verwende für `.bat`- und `.cmd`-Dateien eine explizite Regel:

```
*.bat text eol=crlf
*.cmd text eol=crlf
```

Git warnt bei jedem `git add`

Wahrscheinliche Ursache: Git kündigt lediglich eine erwartete Umwandlung zwischen LF im Repository und CRLF im Arbeitsverzeichnis an.

Reaktion: Kontrolliere mit `git ls-files --eol`, ob die Datei erwartungsgemäß behandelt wird. Wenn die Teamregeln stimmen und der Diff fachlich korrekt ist, ist die Warnung meist harmlos.

Empfohlene Basiskonfiguration

Für die Übungen dieses Kurses unter Windows 11:

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

`core.safecrlf=warn` weist dich auf Umwandlungen hin, bei denen Git einen möglichen Datenverlust oder eine nicht eindeutig umkehrbare Konvertierung erkennt. Es blockiert den Vorgang nicht, macht dich aber aufmerksam.

Prüfe die Konfiguration anschließend:

```
git config --global --get-regexp "core\.(autocrlf|safecrlf)"
```

Eine passende Ausgabe wäre:

```
core.autocrlf true
core.safecrlf warn
```

Checkliste vor dem ersten Commit

- ☐ `core.autocrlf` ist mit `git config --global --get core.autocrlf` geprüft.
- ☐ PhpStorm speichert neue Dateien mit dem zum Projekt passenden Zeilenende.
- ☐ `git status` zeigt nur beabsichtigte Änderungen.
- ☐ `git diff` enthält keine unerwarteten Änderungen an jeder einzelnen Zeile.
- ☐ Vorhandene `.gitattributes`-Regeln wurden beachtet.
- ☐ Shell-Skripte, PHP-Dateien und Konfigurationsdateien verwenden im Team konsistente Regeln.
- ☐ Batchdateien werden bei Bedarf gezielt als CRLF behandelt.

Mit diesem Verständnis vermeidest du einen der häufigsten plattformübergreifenden Git-Fehler: große, irreführende Diffs, die ausschließlich aus unsichtbaren Zeilenende-Unterschieden bestehen.

Revision #1
Created 2026-07-25 11:13:46 UTC by art10m
Updated 2026-07-25 11:13:46 UTC by art10m