

Dateien mit PowerShell prüfen

Eine Übung zu SHA-256, Zeilenenden und Ergebnisobjekten

Öffentliche Leseausgabe · Version 1.0 · 18. September 2026

Webfassung: raphaellugmayr.at/gedanken/powershell-dateien-pruefen

Vom Befehl zu einer nachvollziehbaren Beobachtung

Ein Befehl ist im Unterricht dann hilfreich, wenn die erwartete Beobachtung vorher klar ist. Diese Übung greift einen Teil meines Skripts „PowerShell verstehen“ auf: Wir erzeugen eine kleine bekannte Datei, betrachten ihren Hash und ändern anschließend ausschließlich das Zeilenende. So lässt sich eine Prüfsumme mit den tatsächlich gespeicherten Bytes verbinden.

Die Beispiele richten sich an PowerShell 7. Sie benötigen keine Administratorrechte, keinen Download fremder Skripte und keine Änderung der Ausführungsrichtlinie. Angelegt wird ein eigener, zufällig benannter Übungsordner im temporären Verzeichnis. Die Dateien darin bleiben zur Nachkontrolle erhalten.

1. Eine eindeutig definierte Testdatei anlegen

Wir schreiben genau die drei ASCII-Bytes für abc. Dadurch hängt die Ausgangsdatei nicht von den Textvorgaben eines Editors ab.

```
$demoRoot = Join-Path ([IO.Path]::GetTempPath()) (
    'hash-demo-' + [Guid]::NewGuid().ToString('N')
)
[IO.Directory]::CreateDirectory($demoRoot) | Out-Null
$demoFile = Join-Path $demoRoot 'abc.txt'
[IO.File]::WriteAllBytes($demoFile, [byte[]](0x61, 0x62, 0x63))

Get-Item -LiteralPath $demoFile |
    Select-Object Name, Length, FullName
```

Die Länge muss drei Bytes betragen. Der Ordnername ist bei jedem Durchlauf anders. `-LiteralPath` behandelt den Dateinamen wörtlich; Zeichen wie eckige Klammern werden nicht als Suchmuster ausgewertet.

2. Den Hashwert berechnen und vergleichen

`Get-FileHash` liefert unter anderem Algorithmus, Hash und Pfad. Die Datei wird dafür gelesen. Der verwendete Algorithmus steht im Beispiel ausdrücklich im Aufruf. Die Einzelheiten beschreibt [Microsoft zu Get-FileHash](#).

```
$actual = Get-FileHash -LiteralPath $demoFile -Algorithm SHA256
$expected = 'BA7816BF8F01CFEA414140DE5DAE2223' +
            'B00361A396177A9CB410FF61F20015AD'

$actual | Select-Object Algorithm, Hash, Path
$actual.Hash -eq $expected
```

Der Vergleich muss True liefern. Der erwartete Wert passt genau zu der in Schritt 1 festgelegten Eingabe. Bei einem echten Download würde ich den erwarteten Wert aus einer vertrauenswürdigen Quelle beziehen. Ein selbst neu berechneter Wert ist noch keine unabhängige Referenz für die Herkunft.

3. Nur ein Zeilenende ergänzen

Nun entstehen zwei weitere Dateien. Ihr sichtbarer Text ist weiterhin abc, ihre Bytefolgen unterscheiden sich aber.

```
$lfFile = Join-Path $demoRoot 'abc-lf.txt'
$CrLfFile = Join-Path $demoRoot 'abc-crlf.txt'
[IO.File]::WriteAllBytes($lfFile, [byte[]](0x61, 0x62, 0x63, 0x0A))
[IO.File]::WriteAllBytes(
    $CrLfFile, [byte[]](0x61, 0x62, 0x63, 0x0D, 0x0A)
)

Get-ChildItem -LiteralPath $demoRoot -File |
    Get-FileHash -Algorithm SHA256 |
    Select-Object Hash, Path
```

Es müssen drei verschiedene Hashwerte erscheinen. Die Dateilängen sind drei, vier und fünf Bytes. Das LF-Zeilenende fügt ein Byte an, CRLF zwei. Ein anderer Hash allein bedeutet also noch nicht, dass sich der für einen Menschen erkennbare Inhalt wesentlich geändert hat.

Für eigene Textdateien spielt zusätzlich die Kodierung eine Rolle. Windows PowerShell und neuere PowerShell-Versionen haben unterschiedliche Standardvorgaben bei verschiedenen Textbefehlen. Explizite Bytes vermeiden diese Unterschiede in der Übung. Bei einem Textworkflow sollten Kodierung und gewünschte Zeilenenden bewusst gewählt werden. [Microsoft: Zeichenkodierung](#)

4. Das Ergebnis als Objekt behandeln

`$actual.Hash` greift auf die Eigenschaft des Ergebnisobjekts zu. Die sichtbare Tabelle ist eine Darstellung dieses Objekts. Man muss die Bildschirmzeile nicht anhand von Leerzeichen zerlegen, um den Wert zu vergleichen. Genau darin liegt für diese Aufgabe ein praktischer Vorteil der [PowerShell-Pipeline](#).

```
$actual | Get-Member
$actual.Hash.Length
```

Die Länge des Hexwerts muss 64 sein. Erst am Ende eines Ablaufs würde ich für die Bildschirmausgabe formatieren. Wer vorher mit Formatierungsanweisungen weiterarbeitet, hat nicht mehr dieselben fachlichen Eigenschaften wie im ursprünglichen Ergebnisobjekt.

Wie ich Abweichungen untersuchen würde

Zuerst würde ich den vollständigen Pfad kontrollieren. Wurde wirklich die erwartete Datei gelesen? Danach folgen Dateigröße, Zeilenenden, Kodierung und gegebenenfalls ein erneuter Export. Bei einem Arbeitsablauf kann sich eine Datei auch zwischen zwei Prüfschritten verändern. Deshalb gehören Zeitpunkt und Dateiversion zur Dokumentation.

Die Übung untersucht bekannte Testdateien. Sie durchsucht keine Arbeitsverzeichnisse und verändert keine bestehenden Dateien. Für den Unterricht lässt sich vor jedem Schritt zuerst die erwartete Länge oder Eigenschaft notieren und anschließend mit dem Ergebnis vergleichen. Ein Irrtum in der Erwartung ist dabei nützlich: Er zeigt, welche Annahme noch unklar war.

Überprüfung dieser Fassung

Die vier Codeblöcke wurden am 18. September 2026 zusammen mit PowerShell 7.6.6 unter macOS auf einem Apple-Silicon-Rechner ausgeführt. Der Hashvergleich für abc lieferte True; die Dateien hatten drei, vier und fünf Bytes sowie drei unterschiedliche SHA-256-Werte. Der Hexwert war 64 Zeichen lang. Das dokumentiert diesen konkreten Durchlauf; andere Betriebssysteme wurden für diese Fassung nicht separat getestet.

Welche Aussage bleibt am Ende?

Wir haben für eine festgelegte Bytefolge den erwarteten Hash erhalten und beobachtet, dass zusätzliche Zeilenendbytes andere Werte erzeugen. Das ist eine konkrete, wiederholbare Beobachtung. Eine Aussage über Schadsoftware, Dateiautoren oder fachlich richtige Dokumente wurde damit nicht geprüft.

Den Hintergrund erklärt [SHA-256 verstehen](#). Für Vertraulichkeit und Herkunft führt die nächste Frage zu [Verschlüsselung und digitalen Signaturen](#).

Quellen und Verwendung

Microsoft: Get-FileHash

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash>

Microsoft: Zeichenkodierung in PowerShell

https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_character_encoding

Microsoft: Objekt-Pipelines in PowerShell

https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_pipelines

Autor: Raphael Lugmayr. Diese Leseausgabe entwickelt ausgewählte Inhalte eigener Lehrunterlagen weiter. Sie ist eine Einführung mit Beispielen; die verlinkten Standards und Dokumentationen beschreiben die fachlichen Einzelheiten. Quellenstand: 18. September 2026.

Für Unterricht und Weiterempfehlungen bitte auf die Webfassung verweisen. Sie enthält den aktuellen Stand und zugehörige Materialien. Zitiervorschlag: Raphael Lugmayr (2026), Dateien mit PowerShell prüfen, Version 1.0, raphaellugmayr.at. Rückmeldungen: raphael.lugmayr@stoicera.com.