Suchpotenzial

Reguläre Ausdrücke in der PowerShell benutzen

Jede Skriptsprache, die etwas auf sich hält, bietet reguläre Ausdrücke an: Damit lassen sich beliebige Texte nach Mustern durchsuchen, in Bestandteile zerlegen oder Ersetzungen durchführen. Auch die PowerShell kann mit regulären Ausdrücken umgehen, beim Umgang damit ist aber einiges zu beachten.

Von Hajo Schulz

Um reguläre Ausdrücke – auf Englisch Regular Expressions oder kurz RegEx – ranken sich zahlreiche Gerüchte: Einerseits gelten sie als Universalwerkzeug für so ziemlich jede Aufgabe in der Datenverarbeitung. Andererseits sind sie als schwer beherrsch- und wartbar verschrien und stehen in dem Ruf, praktisch nicht zu lesen und nachzuvollziehen zu sein. Wie so oft liegt die Wahrheit irgendwo in der Mitte.

Zunächst einmal sind reguläre Ausdrücke dazu da, Suchmuster zu formulieren, mit denen sich dann beliebige Texte nach Fundstellen durchforsten lassen. Wahr ist auch, dass diese Muster sehr kompliziert und kryptisch werden können. Müssen sie aber nicht: Schon ein einfaches Suchwort kann ein regulärer Ausdruck sein.

PowerShell

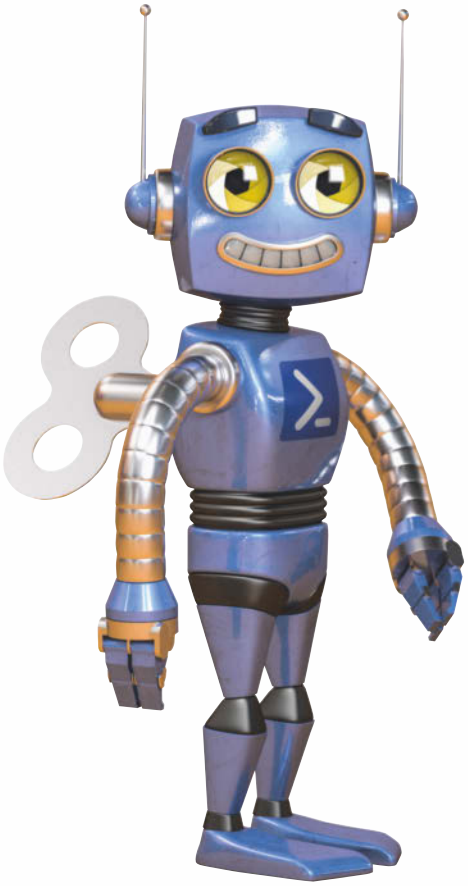
In der PowerShell begegnen dem Anwender reguläre Ausdrücke an verschiedenen Stellen. Die am häufigsten benutzte dürfte das Cmdlet `Select-String` sein. Es dient dazu, Textdateien oder Listen von Zeichenketten zu durchsuchen. Damit ähnelt es dem Unix-/Linux-Programm `grep` oder dem aus der Windows-Eingabeaufforderung bekannten Befehl `findstr`. In seiner einfachsten Form sieht sein Aufruf so aus:

```
Select-String test.log 'Error'
```

Der Befehl liest die Datei test.log Zeile für Zeile ein und gibt nur diejenigen aus, in denen das Wort „Error“ vorkommt. Bei diesem Wort handelt es sich bereits um einen regulären Ausdruck – Folgen normaler Buchstaben und Ziffern ohne Sonderzeichen passen genau auf alle Zeichenketten, in denen sie exakt wie angegeben vorkommen. Man kann `Select-String` also durchaus benutzen, ohne sich jemals mit regulären Ausdrücken beschäftigt zu haben.

Eine erste „echte“ Regex wäre das Suchmuster `'Error|Warning'`: Es findet alle Zeilen, in denen entweder „Error“ oder „Warning“ vorkommt (oder beides). Das Pipe-Symbol `|` steht in der RegEx-Syntax für „oder“.

Übrigens lässt sich `Select-String` auch mit einer Dateimaske wie `*.txt` statt einem einzigen Dateinamen aufrufen. Durchsucht werden dann alle Dateien, zu denen diese Maske passt. Bei einer solchen Dateimaske handelt es sich aber nicht um eine RegEx, sondern um die Windows-typischen Wildcards.



Matchmaker

Um Listen von Dateinamen mithilfe regulärer Ausdrücke zu filtern, kann man in der PowerShell den Vergleichsoperator `-match` einspannen. Alle Textdateien liefert beispielweise der Befehl

```
dir | ? Name -match '\.txt$'
```

Die Ausgabe von `dir` landet hier über die Objekt-Pipeline der PowerShell bei dem Cmdlet `Where-Object`, abgekürzt durch das Fragezeichen. Das verwendet den Operator `-match`, der genau dann „Wahr“ liefert, wenn das, was vor ihm steht, zu der RegEx passt, die ihm folgt.

Der im Beispiel verwendete reguläre Ausdruck setzt sich wie folgt zusammen: Er beginnt mit einem Punkt. Der steht aber in der RegEx-Syntax normalerweise für ein beliebiges Zeichen, ähnlich dem Fragezeichen in Dateimasken. Deshalb ist ihm hier ein \ vorangestellt: Das ist in regulären Ausdrücken das sogenannte Escape-Zeichen. Es sorgt dafür, dass das folgende Zeichen seine besondere Bedeutung verliert und für sich selbst steht. Das abschließende $-Zeichen pinnt den Such-String an das Ende des durchsuchten Textes. Anders gesagt: Der Match ist nur dann erfolgreich, wenn die zu untersuchende Zeichenkette hier endet. Zusammengefasst bedeutet der gezeigte reguläre Ausdruck also, dass der übergebene Dateiname einen Punkt gefolgt von den Zeichen t, x und t enthalten und dann zu Ende sein muss, damit der Vergleich erfolgreich ist.

Als Pendant zum $-Zeichen kennt die RegEx-Syntax noch den Zirkumflex: Beginnt eine RegEx mit dem ^-Zeichen, muss das folgende Muster am Anfang der untersuchten Zeichenkette stehen, damit es passt.

Anders als die meisten anderen RegEx-Implementierungen sucht die PowerShell übrigens standardmäßig, ohne die Groß-/Kleinschreibung zu beachten. Wer groß- von kleingeschriebenen Buchstaben unterscheiden will, muss statt `-match` den Operator `-cmatch` verwenden. `Select-String` kennt zum Umschalten die Option `-CaseSensitive`.

Zeichenmengen

Der oben schon erwähnte Oder-Operator (`|`) ist praktisch, wenn man einige wenige Alternativen erlauben will. Ist aber so etwas wie „eine beliebige Ziffer“ oder „ein Vokal“ gesucht, würde seine Verwendung schnell unübersichtlich. Deshalb kennt die RegEx-Syntax sogenannte Zeichenklassen. Man schreibt sie in das Suchmuster, indem man die erlaubten Alternativen in eckige Klammern einschließt: `'h[aiu]t'` findet „hat“, „Hit“ und „Hut“ (aber beispielsweise nicht „Haut“ – es ist ja nur eines der Zeichen gefragt). Auch Zeichenbereiche sind erlaubt: `'[0-9]'` steht für eine beliebige Ziffer. Beide Schreibweisen lassen sich kombinieren: `'[a-z0-9_]'` erlaubt Buchstaben von „a“ bis „z“, Ziffern und den Unterstrich. Beginnt die Zeichenklasse mit einem ^-Zeichen, bedeutet das, dass hier alles außer den aufgelisteten Zeichen passt: `'[^a-z]'` findet alle Nicht-Buchstaben.

Einige Zeichenklassen sind bereits vordefiniert, darunter `\d` für Dezimalziffern, `\w` für „Wortzeichen“, also Ziffern und Buchstaben, und `\s` für Wortzwischenräume wie Leer- und Tabulatorzeichen oder Umbrüche. Groß geschrieben kehrt sich die jeweilige Bedeutung um: `'\D'` passt zu allen Zeichen außer Ziffern. Außer der kompakten Schreibweise spricht für die Verwendung dieser Klassen, dass sie auch mit Unicode-Sonderzeichen korrekt umgehen: `\w` enthält beispielsweise die deutschen Umlaute, `[a-z]` nicht.

Bei der Suche nach Textmustern kommt es häufig vor, dass ein bestimmtes Element optional ist oder auch mehrfach vorkommen darf. So etwas bilden in regulären Ausdrücken die sogenannten Quantifizierer ab. Sie stehen grundsätzlich hinter dem Element, auf das sie sich beziehen. So findet `'\d*'` null bis beliebig viele Ziffern, `'\d+'` mindestens eine und `'\d?'` keine oder eine. Eine genaue Anzahl oder ein definierter Bereich erlaubter Zeichen lässt sich mit geschweiften Klammern festlegen: `'.{3}'` findet genau drei beliebige Zeichen, `'\s{2,5}'` zwei bis fünf Zwischenräume und `'\w{4,}'` mindestens vier Buchstaben.

Folgende RegEx prüft beispielsweise, ob die Eingabe ein Datum in normaler deutscher Schreibweise enthält, mit oder ohne führende Nullen vor einstelligen Tages- und Monatszahlen und mit optionalen Leerzeichen hinter den beiden Punkten:

```
'\d{1,2}\. ?\d{1,2}\. ?\d{4}'
```

Die ersten beiden Zifferngruppen dürfen ein- oder zweistellig sein, die letzte (das Jahr) muss vierstellig ausgeschrieben sein. Vor den Fragezeichen steht jeweils ein Leerzeichen, sodass dieses in der Eingabe vorhanden sein kann, aber nicht muss. Wie fast immer könnte man den Aufwand für die Erkennung eines gültigen Datums noch weiter treiben, etwa als erste Ziffer der Monatsnummer nur 0 oder 1 zulassen. Sie können ja mal zur Übung eine peniblere RegEx ausarbeiten.

Gruppierung

Um Oder-Alternativen zu formulieren oder sicherzustellen, dass sich ein Quantifizierer auf den richtigen Teil der RegEx bezieht, ist es manchmal nötig, innerhalb des Suchmusters Klammern zu setzen. Ein Beispiel mag das verdeutlichen: Angenommen, Sie wollen prüfen, ob eine Internet-Adresse zu einer von Ihnen erlaubten Top-Level-Domain gehört. Dann könnte die Prüfung so aussehen:

```
$addr -match '[^.]+\.(com|de|org)'
```

Gemeint ist damit eine Zeichenfolge aus mindestens einem, aber beliebig vielen Nicht-Punkten (`[^.]+`), gefolgt von einem Punkt und einer der erlaubten Domains. Ohne die Klammern würde beispielsweise auch die Adresse „duden.at“ die Prüfung überstehen, weil sie „de“ enthält und das erste |-Zeichen alles davor als einen einzigen Unterausdruck ansieht.

Klammern sind außerdem wichtig, um dem `-match`-Operator einen ganz besonderen Trick zu entlocken: das Extrahieren von Daten aus einem Text. Dazu muss man wissen, dass `-match` nicht nur den Erfolg oder Misserfolg seiner Arbeit zurückliefert, sondern im Falle eines positiven Ergebnisses auch die Variable `$matches` befüllt: Sie enthält dann eine Liste, deren erstes Element (mit dem Index 0) den Teil der Eingabe enthält, der dem gesamten regulären Ausdruck entsprochen hat, und in weiteren Einträgen die Bestandteile, die zu geklammerten Ausdrücken gepasst haben. Ein Beispiel mag das verdeutlichen:

```
if($line -match
  '(\d{1,2})\. ?(\d{1,2})\. ?(\d{4})')
{
  $day = [int]$matches[1]
  $month = [int]$matches[2]
  $year = [int]$matches[3]
}
```

Der reguläre Ausdruck ist hier derselbe, der oben schon Datumsangaben erkannt hat, nur ergänzt um drei Klammerpaare. Die speichern die drei erkannten Zahlen in einer Liste und verstauen diese in der Variablen `$matches`, von wo aus sie weiterverarbeitet werden können.

Einen ähnlichen Mechanismus benutzt auch der `-replace`-Operator. So lassen sich zum Beispiel mit der Zeile

```
$line -replace '(.+), (.+)', '$2 $1'
```

die Einträge in einer Namensliste vom Format „Nachname, Vorname“ auf „Vorname Nachname“ umstellen: Im Ersetzungs-Ausdruck kann man geklammerte Ausdrücke aus der Such-RegEx einfach mit `$1`, `$2` und so weiter referenzieren.

Wie weiter?

Die PowerShell kennt noch einige andere, eher exotische Einsatzzwecke für reguläre Ausdrücke, darunter den Operator `-Split`. Was Microsoft dazu an Referenz-Informationen veröffentlicht hat, haben wir für Sie unter ct.de/yhwq verlinkt. Dort finden Sie auch einen Link in die Dokumentation zum .NET Framework, das ja der PowerShell und damit auch ihrer RegEx-Implementierung zugrunde liegt. *(hos@ct.de)* c't

Referenz-Informationen: ***ct.de/yhwq***

Elemente regulärer Ausdrücke (Auswahl)

Syntax	Bedeutung
abc|def	abc oder def
^abc	abc am Anfang der Eingabe
abc$	abc am Ende der Eingabe
.	beliebiges Zeichen
[abc]	eines der angegebenen Zeichen
[a-z]	eines der Zeichen im Bereich
[^abc]	jedes außer den angegebenen Zeichen
a?	null oder ein a
a+	mindestens ein a
a*	beliebig viele (auch null) a
a{3}	genau 3 a
a{2,4}	2 bis 4 a
a{4,}	mindestens 4 a
(abc)	fasst abc zu einem Ausdruck zusammen