

Escape

Das Metazeichen, dem sich dieses Kapitel widmet, ist in diesem Manual bereits an zwei Stellen aufgetreten: Der Backslash. Wir haben ihn kennengelernt im Zusammenhang mit den Positionsankern, unter denen er in der Kombination mit < oder > (d.h. < bzw. >) den Wortanfang bzw. das Wortende kennzeichnet, und, kombiniert mit einem Buchstaben (z.B. `lw`) im Rahmen der Shorthand Expressions.

Zu Beginn des Manuals wurde darauf hingewiesen, dass ein regulärer Ausdruck meist nicht nur aus Metazeichen oder konkreten Zeichen besteht, sondern diese kombiniert. Zur Erinnerung:

Zwischen "abstrakt" und "konkret"

Wie funktioniert das nun mit den Regulären Ausdrücken? Wenn wir in einem Regex-kompatiblen Texteditor unsere Suchanfrage formulieren, dann haben wir zum einen die Möglichkeit, dies mithilfe einer konkreten Zeichenkette zu tun (z.B. *Autobahnen*). So weit nichts Neues! Reguläre Ausdrücke erlauben es uns zudem, völlig abstrakte Suchanfragen zu formulieren, in denen kein einziger Buchstabe vorkommt (z.B. das letzte Zeichen am Ende einer jeden Zeile). Meist bewegt man sich jedoch dazwischen. Die Syntax regulärer Ausdrücke ermöglicht es, konkrete Zeichen (z.B. *A, b, 1, !*, etc.) und sogenannte Metazeichen (z.B. Platzhalter, Wiederholung, Alternativen, Position) zu kombinieren, um so eine möglichst präzise und nötigenfalls komplexe Suchanfrage zusammenzustellen. Auch Metazeichen müssen in dem Suchstring auf irgendeine Weise ausgedrückt werden, weshalb einige konkrete Zeichen zu Metazeichen umgewidmet wurden (z.B. `.`, `{}`, `[]`, `|`, etc.). Ein häufiger Anfängerfehler besteht in der Tat darin, dass bestimmte Zeichen eigentlich konkret gemeint sind, bei der Durchführung der Suche jedoch als Metazeichen erkannt werden.

Nachdem in den Vorausgehenden Kapiteln nun die durch einige Metazeichen gegebenen Möglichkeiten beleuchtet worden sind, liegt für den Leser möglicherweise folgende Frage nah: Und was, wenn ich nach einem konkreten Zeichen suchen muss, das im Regex-Standard jedoch als Metazeichen fungiert? Für diesen Zweck stellt das Regex-Inventar den Backslash bereit. Der Backslash macht aus konkreten Zeichen Metazeichen und aus Metazeichen konkrete Zeichen.

Folgende Zeichen werden von allen Regex-fähigen Programmen als Metazeichen erkannt:

```
. [] \ * ? + - ^ $ /
```

Will ich nun zum Beispiel, dass ein Ausdruck Fragezeichen als konkrete Zeichen matcht, so muss ich dieses durch einen vorausgehenden Backslash kennzeichnen. Der Ausdruck zum Auffinden von Fragezeichen lautet somit `?`, einer zum Auffinden von Punkten `.` und einer zum Auffinden von Backslashes `\`. Es existieren jedoch auch Zeichenkombinationen, die sich erst durch einen Backslash zu einem Metazeichen werden, z.B. `lw` oder `l<`. Hier bewirkt der Backslash genau das Gegenteil. Während `w` und `<` konkrete Zeichen sind, wird ihnen in `lw` und `l<` eine Bedeutung zugeordnet (nämlich respektive: ein Wortzeichen, Beginn des Wortes).

Für wiederum andere Zeichen variiert der Zeichenstatus je nach Programm. Dazu zählen:

```
() {}
```

Sollte ein Ausdruck, von dessen Richtigkeit man vollends überzeugt ist, dennoch nicht das markieren, was er soll, kann es helfen, wenn man vor die eben genannten Zeichen einen Backslash zu platzieren. So auch im Falle des behandelten Quantifikators `{min,max}`, welcher unter Umständen auch die Form `{min,max}` haben kann.

Für die meisten Metazeichen gilt: Wenn sie außerhalb von eckigen Klammern über eine Metazeichenfunktion verfügen, dann ist dies auch innerhalb der eckigen Klammern der Fall. Es sei aber auch auf zwei Zeichen eingegangen, die innerhalb eckiger Klammern ihre Bedeutung ändern:

```
- ^
```

Während der Positionsanker `^` außerhalb eckiger Klammern einen Zeilenanfang markiert, stellt `-` gar kein Metazeichen dar, sondern sucht nach einem konkreten Bindestrich.

Befindet sich `^` direkt auf die öffnende Klammer eines Zeichensatzes folgend, ist es, wie bereits dargelegt, gleichbedeutend mit "außer". In jeglicher anderer Position innerhalb des Sets wird es seiner Metabedeutung enthoben und bildet ein konkretes Zeichen. Hierzu folgender Überblick:

Ausdruck	matcht...
<code>^a</code>	alle <i>a</i> am Zeilenanfang
<code>[^a]</code>	alle Zeichen außer <i>a</i> (in beliebiger Position)
<code>[a^]</code>	alle <i>a</i> und <code>^</code>

Ähnlich verhält es sich mit dem Bindestrich. Wir haben `-` bereits bei der Definition von Intervallen in Zeichensets kennengelernt und gesehen, dass es dort durchaus als Metazeichen fungiert. Dies funktioniert jedoch nur zwischen zwei Zeichen. Befindet sich der Bindestrich direkt nach der öffnenden Klammer eines Sets oder direkt vor der schließenden, wird er als konkretes Zeichen interpretiert. Und was passiert, wenn man den Bindestrich zwischen zwei Zeichen platziert, die nicht dergleichen Zeichenklasse angehören? Zum Beispiel `[4-a]`? Dies ist dem Rechner ziemlich egal, da er Zeichen über einen (ASCII-)Code definiert. Zwischen `4` (ASCII Dec 52) und `a` (ASCII Dec 97) liegen ein paar Ziffern, Sonderzeichen, alle Großbuchstaben und schließlich wieder ein paar Sonderzeichen. All dies bildet samt `4` und `a` das Zeichenset. Eine umgekehrte Reihenfolge führt übrigens zu einer Fehlermeldung.

Ausdruck	matcht...
----------	-----------

-a	- gefolgt von einem a
[a-e]	alle Buchstaben im Intervall a bis e
[-a]	alle - und alle a
[a-]	alle - und alle a
[4-a]	alle Zeichen im Intervall 4 (ASCII Dec 52) bis a (ASCII Dec 97)



Insbesondere bei Sonderzeichen sollte ich mir stets klar machen,

- bei welchen es sich um Metazeichen und bei welchen um konkrete Zeichen handelt.
- dass mich mithilfe eines Backslashes Metazeichen zu konkreten Zeichen und konkrete Zeichen zu Metazeichen ändern kann.
- ob die geschweiften und runden Klammern als Metazeichen oder als konkrete Zeichen definiert sind. Dazu probiere ich entweder einen einfachen Ausdruck aus oder lese das im Hilfe-Bereich meines Programms nach.
- dass innerhalb von Zeichensetdefinitionen für ^ und - abweichende Regeln gelten. Falls ein Ausdruck nicht funktioniert, überprüfe ich die Position der beiden Zeichen innerhalb der eckigen Klammern.



Unbekanntes Makro: 'scrollbar'