

## ANHANG B

# Reguläre Ausdrücke

Reguläre Ausdrücke dienen dazu, Sprachen, also Mengen von Zeichen, zu beschreiben und zu manipulieren. Ein regulärer Ausdruck ist eine Zeichenfolge, die in abgekürzter Schreibweise eine Menge von Zeichenketten definiert (eine *reguläre Menge*). Eine Zeichenkette stimmt mit einem regulären Ausdruck überein, wenn sie ein Element der von dem regulären Ausdruck beschriebenen regulären Menge ist. Eine einfache Variante, die die meisten schon einmal verwendet haben, ist das Pattern Matching der Kommandozeile, zum Beispiel: `ls *.tex`, das uns alle vorhandenen LaTeX-Dateien anzeigt. Das Pattern Matching der Shell ist allerdings recht begrenzt. Wenn es darum geht, E-Mails auf bestimmte Muster zu untersuchen, müssen Pattern, die man verwenden kann, etwas komplizierter sein. Reguläre Ausdrücke für sich sind ein so komplexes Thema, dass man darüber eigene Bücher schreiben kann. Jedem, der sich damit näher befassen möchte, sei das Buch »Reguläre Ausdrücke« von Jeffrey E. Friedl ans Herz gelegt.

Der einfachste reguläre Ausdruck besteht nur aus einer einfachen Zeichenkette ohne Operatoren: `/foo/`. Das »/« stellt hierbei einen Begrenzer dar, je nach Sprache kann dieser Begrenzer auch variieren. Der Begrenzer legt den Beginn und das Ende des Ausdrucks fest. Das Beispiel trifft auf alle Vorkommen der Zeichenkette `foo` zu. Das gilt unabhängig davon, ob diese am Anfang einer Zeile oder in einem Wort vorkommt, also zum Beispiel auch auf die Zeichenkette `'foobar'`. Zusätzlich zu den Zeichenketten gibt es verschiedene Operatoren, die den Suchausdruck modifizieren.

Tabelle B-1: Reguläre Ausdrücke

Operator	Bedeutung	Beispiel
<code>^</code>	Repräsentiert den Zeilenanfang	<code>/^Anfang/</code> Findet »Anfang«, wenn es am Anfang der Zeile steht.
<code>\$</code>	Repräsentiert das Zeilenende	<code>/Ende\$/</code> Finde »Ende«, wenn es das letzte Wort in der Zeile ist.

Tabelle B-1: Reguläre Ausdrücke (Fortsetzung)

Operator	Bedeutung	Beispiel
*	Beliebig viele oder kein Vorkommen des vorangehenden Zeichens oder Ausdrucks	/1*2/ Findet 1 und 2, egal ob und wie viele Leerzeichen dazwischen liegen.
.	Trifft auf ein beliebiges Zeichen zu	/./o/ Trifft auf alles zu, das mit oo endet und ein beliebiges Zeichen davor hat.
+	Mindestens ein oder mehr Vorkommen des vorhergehenden Zeichens oder Ausdrucks	/a+/ Findet mindestens »a«, aber auch beliebig oft »a«.
?	Null oder ein Vorkommen des vorangehenden Zeichens oder Ausdrucks	/a?/ Findet ein oder kein »a«.
{min,}	Mindestens »min« Vorkommen des vorhergehenden Zeichens oder Ausdrucks	/1{4,}/ »1« muss mindestens viermal oder öfter vorkommen.
{min, max}	Mindestens »min« und maximal »max« Vorkommen des vorhergehenden Zeichens oder Ausdrucks	/1{2,4}/ »1« muss mindestens zweimal, aber darf maximal viermal vorkommen.
{n}	Genau »n« Vorkommen des vorhergehenden Zeichens oder Ausdrucks	/1{3}/ Trifft auf dreimaliges Vorkommen von »1« zu.
[ ]	Genau eines der eingeschlossenen Zeichen	/a[bc]/ Trifft auf »ab« oder »ac« zu.
[^ ]	Passt auf genau ein Zeichen, das verschieden von den Zeichen im Ausdruck ist	/a[^bc]/ Trifft nicht auf »ab« oder »ac« zu.
[A-Z], [0-9]	Steht für die Zeichen in dem angegebenen Bereich.	/[o-q]/ Trifft auf fo, fp und fq zu.
\	Sperrt die Sonderbedeutung des Zeichens	/a\\/ Macht aus »/« ein normales Zeichen.
<	Muster am Wortanfang suchen. Dieser Operator kann je nach verwendeter Software anders sein.	/<<foo/ Findet »foobar«, aber nicht »barfoo«.
>	Muster am Wortende suchen. Dieser Operator kann je nach verwendeter Software anders sein.	/foo>/ Findet »barfoo«, aber nicht »foobar«.
	Das »oder« erlaubt Verwendung von Alternativen.	/({foo} bar)/ Findet »foo« oder »bar«.
( )	Klammern werden zur Gruppierung und Referenzierung (siehe unten) von Ausdrücken eingesetzt.	/({foo}){2}/ Findet zweimaliges Vorkommen des gruppierten Ausdrucks (»foofoo«).

## Rückwärtsreferenzen

Manchmal kann es nützlich sein, auf Treffer in einem Ausdruck noch einmal zugreifen zu können, deshalb gibt es Rückwärtsreferenzen, die es ermöglichen, auf vorangegangene Treffer zuzugreifen. Einen Treffer speichert man, indem man ihn »gruppiert«, also in Klammern einschließt. Danach kann man auf die erste Gruppe mit \1, die zweite Gruppe mit \2 und so weiter zugreifen. Der folgende Ausdruck findet zum Beispiel verdoppelte Wörter:

```
/<([a-zA-Z]+) +\1>/
```

Zuerst findet man mit dem Operator »<« eine Wortgrenze, danach gruppiert man das Wort. Da nur Wörter und keine Zahlen oder Ähnliches von Interesse sind, beschränkt man sich mittels [a-zA-Z]+ auf Buchstabenketten beliebiger Länge. Danach dürfen beliebig viele Leerzeichen folgen. Anschließend greift man mit der \1 auf den vorherigen Treffer zu, und wenn das folgende Wort identisch mit dem vorangegangenen Wort ist, erzielt der Ausdruck einen Treffer.

## Geschmacksrichtungen

Leider gibt es je nach Implementierung sehr große Unterschiede in der Syntax der verschiedenen Programme, einige Features sind auch gar nicht implementiert worden. Einige Implementierungen erweitern die »klassischen« Ausdrücke, um eine größere Vielfalt an Ausdrücken zu ermöglichen. Es folgt eine kleine Tabelle, die die größten Unterschiede darstellt, diese Tabelle wurde Friedls Buch »Mastering Regular Expressions« entnommen. Weitere Informationen finden sich in der Dokumentation der verwendeten Software/Sprache.

Tabelle B-2: Unterschiede bei regulären Ausdrücken

Feature	Grep	Egrep	awk	Perl
*, ^, \$, [...]	✓	✓	✓	✓
? +	\? \+ \	? +	? +	? +
Gruppierung	\(...\)	(...)	(...)	(...)
Wortgrenzen	nicht unterstützt	\< \>	nicht unterstützt	\b \B
Rückwärtsreferenzen	✓	nicht unterstützt	nicht unterstützt	✓
\w, \W	✓	nicht unterstützt	nicht unterstützt	✓

