

Reguläre Ausdrücke

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw-mannheim.de
Version: 2024-02-26



Reguläre Ausdrücke (Wiederholung)

Reguläre Ausdrücke (🇺🇸 *regular expressions*) sind ein Standardwerkzeug. Wir verwenden diese insbesondere um Wörterbücher, Leaks und weitere Kontextinformationen zu verarbeiten.

Reguläre Ausdrücke können insbesondere zum Patternmatching auf Textdaten genutzt werden.

Reguläre Ausdrücke beschreiben reguläre Sprachen und können durch einen endlichen Automaten erkannt werden.

Reguläre Ausdrücke - in der Standardeinstellung - nehmen immer einen maximalen Musterabgleich vor (🇺🇸 *greedy matching / eager matching*).

Reguläre Ausdrücke (Grundlagen)

1. Zeichenklassen:

Buchstaben und Zahlen können direkt in einem regulären Ausdruck verwenden, um entsprechenden Text zu finden. Zum Beispiel steht **"a"** für das Zeichen **a**.

```
echo -n "abc" | grep -E "a"
```

findet: a

Der Punkt repräsentiert ein beliebiges Zeichen - außer den Zeilenumbruch.

```
echo -n "abc" | grep -E "a."
```

findet: ab

Reguläre Ausdrücke (Grundlagen)

1. Zeichenklassen (fortgesetzt):

Eine ganze Klasse von Zeichen kann in eckigen Klammern angegeben werden "[]".

```
echo -n "abcdefg" | grep -E "[acg]"
```

findet: a, c, g

Klassen können auch durch Bereiche beschrieben werden (a-z, A-Z, 0-9):

```
echo -n "ab12xy" | grep -Eo "[a-z]"
```

findet: a, b, x, y

Reguläre Ausdrücke (Grundlagen)

1. Zeichenklassen (fortgesetzt):

Welche Buchstaben genau durch eine Klasse repräsentiert werden hängt von den Spracheinstellungen ab!

```
LANG=de_DE.UTF-8; echo "aä" | grep -Eo "[a-z]"
```

findet: a, ä

aber

```
LANG=C; echo "aä" | grep -Eo "[a-z]"
```

findet „nur“: a

Reguläre Ausdrücke (Grundlagen)

1. Zeichenklassen (fortgesetzt):

Die Negation einer Klasse wird durch an ein `^` direkt am Anfang der Klasse erzwungen.

```
echo "abc123" | grep -Eo "[^a-z]"
```

findet: 1, 2, 3

2. Escapezeichen:

Der Backslash `\` dient als Escapezeichen für Sonderzeichen.

```
echo "abc-123[a-z]" | grep -Eo "[a-z]"
```

findet: `[a-z]` (aber nicht "abc")

Reguläre Ausdrücke (Grundlagen)

3. Anker

^: Steht für den Anfang einer Zeile.

\$: steht für das Ende einer Zeile.

```
$ echo "abcabcabc" | grep -Eo "abc"  
abc  
abc  
abc  
$ echo "abcY_abcZ" | grep -Eo "^abc."  
abcY  
$ echo "XbcYbc" | grep -Eo ".bc$"  
Ybc
```

Reguläre Ausdrücke (Grundlagen)

3. Quantifizierung

*: "null oder mehr" Vorkommen des vorherigen Zeichens oder Musters.

+: "ein oder mehr" Vorkommen des vorherigen Zeichens oder Musters.

?: "null oder ein" Vorkommen des vorherigen Zeichens oder Musters.

```
$ echo "Sa--aa--aaaE" | grep -Eo "aa*"
a, aa, aaa
$ echo "Sa--aa--aaaE" | grep -Eo "aa+"
aa, aaa
$ echo "Sa--aa--aaaE" | grep -Eo "a?"
a, a, a, a, a, a
```


Reguläre Ausdrücke (Grundlagen)

3. Quantifizierung (fortgesetzt)

`{X,Y}`: zwischen X und Y Vorkommen des vorherigen Zeichens oder Musters. Die obere Grenze ist optional, um zum Beispiel X und mehr Vorkommen zu finden.

```
$ echo "Sa--aa--aaaE" | grep -Eo "a{2,2}"  
aa  
aa  
$ echo "Sa--aa--aaaE" | grep -Eo "a{2,3}"  
aa  
aaa
```

Reguläre Ausdrücke (Grundlagen)

4. Alternativen: | trennt verschiedene Alternativen.

```
$ echo "HundMausAffe" | grep -Eo "Hund|Affe"  
Hund  
Affe
```

Aufgrund des „gierigem“ Musterabgleichs ist bei dem Abgleich von Alternativen generell darauf zu achten, dass zuerst auf den letzten Abgleich geprüft wird.

```
$ echo "Schiffahrt" | grep -Eo "Schiff|Schiffahrt"  
Schiffahrt  
  
# Perl compatible  
echo "Schiffahrt" | grep -Po "Schiff|Schiffahrt"  
Schiff
```

Reguläre Ausdrücke (Grundlagen)

5. Klammern: () dienen der Gruppierung von Teilausdrücken und der Referenzierbarkeit bzw. Rückreferenzen.

Beispiel: der folgende Ausdruck findet Zeichenketten, die mit dem Zeichen aufhören mit dem sie begonnen haben.

```
$ echo "XaaaaX" | grep -Eo "^(.)*\1$"
XaaaaX
```

```
$ echo "XaaaaY" | grep -Eo "^(.)*\1$"

```

Reguläre Ausdrücke (Fortgeschrittene Grundlagen)

6. Lookahead

(?=...): ist ein positiver Lookahead und stellt sicher, dass ein bestimmtes Muster im Text folgt, ohne es selbst in das Ergebnis aufzunehmen.

(?!...): ist ein negativer Lookahead und stellt sicher, dass ein bestimmtes Muster im Text *nicht* folgt.

```
$ echo "HundKatzeHundMaus" | grep -Po 'Hund(?=Katze){1,2}'  
HundKa
```

```
$ echo "HundKatzeHundMaus" | grep -Po 'Hund(?!Katze){1,2}'  
HundMa
```

1. Prüfen Sie ob der Begriff: schmetterling oder Schmetterling in rockyou vorkommt.
2. Finden Sie alle Passworte in denen ein Zeichen mind. 3 oder mehrmals wiederholt wird. z.B. "x0000!" oder "aaaabbbb".
3. Finden Sie alle Passworte, in denen eine Sequenz mit mindestens 3 Zeichen wiederholt wird. z.B. "TestTest" oder "1AffelstAffe#".