

Reguläre Ausdrücke

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de
Version: 1.3.0.1

Folien: [HTML] <https://delors.github.io/lab-regexp/folien.de.md.html>
[PDF] <https://delors.github.io/lab-regexp/folien.de.md.html.pdf>

Fehler melden: <https://github.com/Delors/delors.github.io/issues>

KI Verwendung: Bei der Erstellung der Unterlagen wurden KI Assistenten (insbesondere Claude aber ggf. auch ChatGPT, Ollama mit Gemma/LLama/Qwen oder OpenCode mit Kimi/Qwen/Deepseek...) unterstützend eingesetzt. Dies erfolgte insbesondere zur Unterstützung bei der Generierung von Grafiken (d. h. SVG Dateien), oder um sich Übersichtstabellen generieren zu lassen. Weiterhin wurde KI zur allgemeinen Qualitätssicherung eingesetzt. Inhalte, die ggf. von der KI vorgeschlagen wurden, wurden im Falle der Übernahme explizit validiert und angepasst.

Reguläre Ausdrücke

- Reguläre Ausdrücke (🚩 *regular expressions*) sind ein Standardwerkzeug zum Patternmatching auf Textdaten.
 - Wir verwenden diese z. B. zum Suchen und Ersetzen in Textdateien,
 - im Rahmen der Entwicklung von Lexern und Parsern,
 - um Eingaben zu überprüfen (Sanitizing) oder
 - um Wörterbücher, Leaks und weitere Kontextinformationen zu Passwortkandidaten zu verarbeiten.
- Reguläre Ausdrücke beschreiben reguläre Sprachen und können durch einen endlichen Automaten erkannt werden.
- Reguläre Ausdrücke nehmen - im Normalfall - immer einen maximalen Musterabgleich vor (🚩 *greedy matching / eager matching*).

1. Schreiben von regulären Ausdrücken

Fokus ist hier auf grundlegende reguläre Ausdrücke.

Es gibt verschiedene Dialekte von regulären Ausdrücken. Hier wird der Dialekt verwendet, der von GNU grep mit der Option -E (Extended Regular Expressions) unterstützt wird. Einige Beispiele verwenden Perl-kompatible reguläre Ausdrücke (PCRE), die mit grep -P genutzt werden können.

Reguläre Ausdrücke - Zeichenklassen

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

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

findet: a

Der Punkt repräsentiert ein beliebiges Zeichen - außer den Zeilenumbruch.

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

findet: ab

Klassen von Zeichen können in eckigen Klammern angegeben werden "[]".

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

findet: a, c, g

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

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

findet: a, b, x, y

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

```
1 | LANG=de_DE.UTF-8; echo "ää" | grep -E "[a-z]"
```

findet: a, ä

aber

```
1 | LANG=C; echo "ää" | grep -E "[a-z]"
```

findet „nur“: a

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

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

findet: 1, 2, 3

Reguläre Ausdrücke - Escapezeichen

Der Backslash \ dient als Escapezeichen für Sonderzeichen.

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

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

Reguläre Ausdrücke - Anker

^: Steht für den Anfang einer Zeile.

\$: steht für das Ende einer Zeile.

```
1 $ echo "abcabcabc" | grep -Eo "abc"
2 abc
3 abc
4 abc
5 $ echo "abcY_abcZ" | grep -Eo "^abc."
6 abcY
7 $ echo "XbcYbc" | grep -Eo ".bc$"
8 Ybc
```

Reguläre Ausdrücke - 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.

```
1 $ echo "Sa--aa--aaaE" | grep -Eo "aa*"
2 a, aa, aaa
3 $ echo "Sa--aa--aaaE" | grep -Eo "aa+"
4 aa, aaa
5 $ echo "Sa--aa--aaaE" | grep -Eo "a?"
6 a, a, a, a, a, a
```

{ 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.

```
1 $ echo "Sa--aa--aaaE" | grep -Eo "a{2,2}"
2 aa
3 aa
4 $ echo "Sa--aa--aaaE" | grep -Eo "a{2,3}"
5 aa
6 aaa
```

Reguläre Ausdrücke - Alternativen

| trennt verschiedene Alternativen.

```
1 $ echo "HundMausAffe" | grep -Eo "Hund|Affe"
2 Hund
3 Affe
```

Aufgrund des „gierigem“ Musterabgleichs ist bei dem Abgleich von Alternativen im Allgemeinen darauf zu achten, dass zuerst auf längere bzw. spezifischere Muster geprüft wird.

```
1 $ echo "Schiffahrt" | grep -Eo "Schiff|Schiffahrt"
2 Schiffahrt
3
4 # Perl compatible (requires GNU grep)
5 echo "Schiffahrt" | grep -Po "Schiff|Schiffahrt"
6 Schiff
```

Reguläre Ausdrücke - Klammern

() dienen der Gruppierung von Teilausdrücken und der Referenzierbarkeit bzw. Rückreferenzen mittel `$<NR>` wobei NR die Nummer der Gruppe ist.

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

```
1 $ echo "XaaaaX" | grep -Eo "^(.)*\1$"
2 XaaaaX
3
4 $ echo "XaaaaY" | grep -Eo "^(.)*\1$"
```

Reguläre Ausdrücke - 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.

```
1 $ echo "HundKatzeHundMaus" | grep -Po 'Hund(?=Katze){1,2}'
2 HundKa
3
4 $ echo "HundKatzeHundMaus" | grep -Po 'Hund(?!Katze){1,2}'
5 HundMa
```

Reguläre Ausdrücke - *Non-capturing Groups*

(?:...): definiert eine Gruppe, die nicht für Rückreferenzen verwendet wird. (Primär für Effizienz.)

```
1 $ echo "abcabc" | grep -Po '(?:abc){2}'  
2 abcabc
```

Reguläre Ausdrücke - *Eager vs. Lazy Matching*

*?: match 0 oder mehr Vorkommen des vorherigen Zeichens oder Musters, aber so wenige wie möglich/nötig

+?: match 1 oder mehr Vorkommen des vorherigen Zeichens oder Musters, aber so wenige wie möglich/nötig

(??) : match 0 oder 1 Vorkommen des vorherigen Zeichens oder Musters, aber so wenige wie möglich/nötig

```
1 $ echo '<div>Hallo</div><div>Welt</div>' | grep -Eo '<.*?>'
2 <div>
3 </div>
4 <div>
5 </div>
6 $ echo '<div>Hallo</div><div>Welt</div>' | grep -Eo '<.*>'
7 <div>Hallo</div><div>Welt</div>
```



Fingerübungen

1.1. Schmetterling in Rockyou

Prüfen Sie ob der Begriff: schmetterling oder Schmetterling in der Datei `rockyou.txt` vorkommt.

1.2. Wiederholungen von Zeichen in Passwörtern

Finden Sie alle Passworte in denen ein Zeichen mind. 3 oder mehrmals wiederholt wird. z. B. "x0000!" oder "aaaabbbb".

1.3. Wiederholungen von Sequenzen in Passwörtern

Finden Sie alle Passworte, in denen eine Sequenz mit mindestens 3 Zeichen wiederholt wird, z. B. „TestTest“ oder „1AffeIstAffe#“.

1.4. Kein Sonderzeichen danach

Finden Sie alle Buchstabensequenzen mit ein bis drei Zeichen, bei denen das letzte Zeichen nicht gefolgt wird von einem Sonderzeichen (z. B. `ab` und `de` in „`abc_def_`“).

