Regular Expressions - Cheat Sheet

Definiton: Ausdruck

Suchmuster, das eine bestimmte Menge von Zeichenketten beschreibt und erkennt.

`.` beliebiges Zeichen;
Ausnahme: Zeilenumbruch

`[?]` Zeichen, die enthalten sein dürfen
`[abcd]` oder `[a-d]` => Findet a,b,c,d, aber nicht f.

`[^?]` Zeichen, die nicht enthalten sein dürfen
`[^abcd]` oder `[^a-d]` => Findet f, aber nicht a,b,c,d.

Gruppierungen

`(?)` Ausdrücke werden gruppiert und können über \G 'Backreference' weiter verwendet werden.

`\1` Backreference, die (hier) auf die erste Gruppe zugreift.

```
s/\(Tux\) \(Fridolin Lars\)/\2 \1/g
=> Aus Tux Fridolin Lars wird Fridolin Lars Tux.
```

`(?:?)` Auf die Gruppe kann nicht per Backreference (P) zugegriffen werden.

Escaping

Escaping ist notwendig, wenn auf Zeichen gefiltert wird, die in der Regex-Syntax eine Bedeutung haben.

Escaping erfolgt, indem ein '\' vor das Zeichen gesetzt wird. Das Zeichen kann auch in '[ ]' gesetzt werden.

```
grep '(2h00)'
grep -E '\(2h00\)' <- Beide Ausdrücke suchen nach (2h00).
```

\G `grep -G` kann mit dem Funktionsumfang von `grep -E` genutzt werden, indem einige Parameter escaped werden.

```
grep '\(ze\)'  <- Beide Ausdrücke suchen
grep -E '(ze)'    nach der Gruppe ze.
```

Delimiter

In einigen Programmen (z.B. `sed` oder `vim`) kann der Delimiter (Trenner) frei gesetzt werden. Ein Escaping des Delimiters kann so vermieden werden.

```
echo "cifs://server/path\file" |
  sed -e 's#//#/#g' <- '#' ist der Delimiter
```

Legende

Der Regex-Funktionsumfang unterscheidet sich je nach Tool und Programmiersprache.

\G erfordert escaping in `grep -G`

(P) nur in `grep -P` vorhanden

Chemnitzer Linux-Tage
22. und 23. März 2025

man grep

`grep` searches for PATTERNS in each FILE

`-G, --basic-regexp`
Interpret PATTERNS as basic regular expressions.

`-E, --extended-regexp`
Interpret PATTERNS as extended regular expressions.

`-P, --perl-regexp`
Interpret PATTERNS as Perl-compatible regular expressions (PCREs).

`-F, --fixed-strings`
Interpret PATTERNS as fixed strings, not regular expressions.

```
Das Ergebnis beginnt mit einem 'c' oder 'K'.        Es folgt mind. ein 'a', dann ein 't'.

'[cK]h{0,}a+t(ze)?'

An der nächsten Stelle wird ein 'h' 0-mal oder mehrmals gefunden.     Die Zeichen 'ze' sind optional.
```

Quantifier

Die Quantifier beziehen sich immer auf das **vorherige** Zeichen.

```
grep -P 'Linux?' <- das 'x' ist optional
=> Mögliche Ergebnisse: Linus oder Linux
```

`?` Der **vorh.** Ausdruck ist optional. \G
Wiederholung 0 oder 1

`+` Der vorh. Ausdruck muss mind. 1x vorkommen. \G
Wiederholung >= 1

`*` Der vorh. Ausdruck muss nicht und kann beliebig oft gefunden werden.
Wiederholung >= 0

`{n,}` Der vorh. Ausdruck wird mind. n-mal gefunden. \G
Wiederholung >= n

`{n,m}` Der vorh. Ausdruck wird mind. n-mal, \G
aber höchstens m-mal gefunden.
Wiederholung >= n, <= m

`{n}` Der vorh. Ausdruck wird genau n-mal gefunden. \G
Wiederholung = n

`|` Alternative: \G
`grep -E 'clt|CLT'` => sucht nach 'clt' oder 'CLT', 'Clt' wird nicht gefunden.

Zeichenklassen

`\w` Findet Buchstaben, Zahlen oder Unterstriche-

`\W` Findet alle Zeichen **außer** Buchstaben, Zahlen und Unterstriche.

`\d` Findet Zahlen. (P)

`\D` Findet alle Zeichen **außer** Zahlen. (P)

`\s` Findet Whitespace (Leerzeichen, Tabs).

`\S` Findet **nicht**-Whitespace.

Grenzen

`^` Der Ausdruck steht am Zeilenanfang.

`$` Der Ausdruck steht am Zeilenende.

```
^$ => Matched auf eine leere Zeile.
^.*$ => Matched auf alles.
```

`\b` (=word boundary) findet Zeichenketten, an deren Anfang oder Ende whitespace steht:

```
\bpin\b => Sucht nach ' pin ', pinguin matched nicht.
```

`\B` Findet Zeichenketten, an deren Anfang oder Ende **kein** Whitespace steht:

```
echo 'denken entdecken ende' | grep '\Bde\B'
=> denken entdecken ende
```

Flags

Flags (Schalter) sind ein optionaler Parameter, um das Verhalten des Ausdrucks zu verändern.

`/g` **global matching**
Stoppt nicht nach dem ersten Treffer, sondern findet alle Übereinstimmungen.

`/i` **case-insensitive**
Groß- oder Kleinschreibung wird ignoriert.

`/m` **multiline mode**
Jede Zeile wird separat behandelt.

Lookaround (P)

Mit einem Lookaround kann ein Ausdruck gesucht werden, der (nicht) vor oder hinter einem anderen Ausdruck steht.

```
\d+(?=€) => Sucht nach mind. einer Zahl, der ein € Symbol folgt; z.B. 3€.
```

`(?=)` positiver Lookahead

`(?!)` negativer Lookahead

`(?<=)` positiver Lookbehind

`(?<!)` negativer Lookbehind