

Eine Katze läuft über meine  
Tastatur!

—

Einstieg in Regex

Marie Mann – m.mann@pengutronix.de



# Marie Mann



- seit 2017 bei Pengutronix
- 4. CLT Vortrag



# Wie komme ich zu dem Vortrag?

Mach mal 'ne Shell auf!

```
▶ ls 202312* 2024* | sed 's/^.*-\  
(...)\-.*\/\1/' > ~/c.2024
```

```
▶ sort ~/c.2024 userlist | uniq -u
```



# Bearbeiten von Pengutronix-Kalendern

2025 Mar 21 | Fri |

[ ] ptx.buero.pm

( ) Anreise CLT

2025 Mar 22 + Sat +

( ) CLT

2025 Mar 23 + Sun +

[ ] 14:00–15:00 Vortrag



# Wann nutzt ihr Regular Expressions?



# Wann nutzt ihr Regular Expressions?

- **Input-Validierung**
- Suchen & Ersetzen
- Filtern von Informationen



# Wann nutzt ihr Regular Expressions?

- Input-Validierung
- **Suchen & Ersetzen**
- Filtern von Informationen



# Wann nutzt ihr Regular Expressions?

- Input-Validierung
- Suchen & Ersetzen
- **Filtern von Informationen**

# Linux- / Unix-Tools

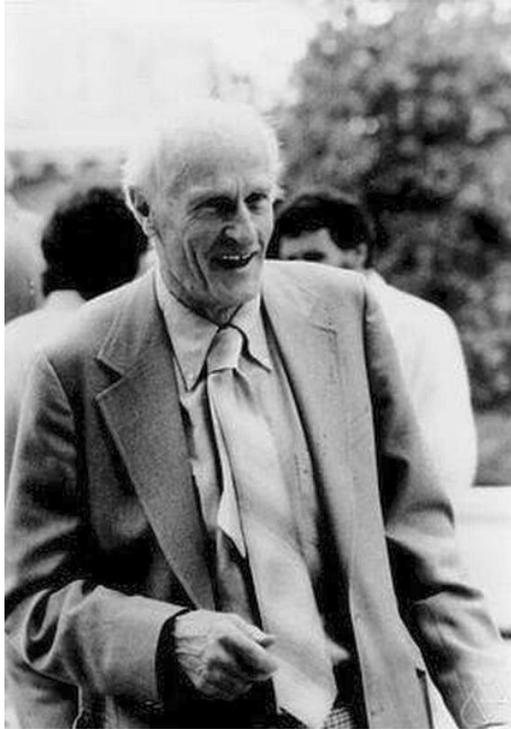
Python sed Vim  
grep  
tr perl awk

... und viel mehr



„[...] denn es gibt kein vernünftiges System, in dem reguläre Ausdrücke nicht vorkommen.“

# Exkurs: Personen hinter Regular Expressions



cc-by-sa 2.0 Quelle: <https://commons.wikimedia.org/wiki/File:Kleene.jpg>

**Stephen Cole Kleene**  
(1909 - 1994)



cc-by-sa 4.0 Quelle:  
[https://commons.wikimedia.org/wiki/File:Henry\\_Spencer\\_2023.jpg](https://commons.wikimedia.org/wiki/File:Henry_Spencer_2023.jpg)

**Henry Spencer**  
(\*1955)

Aktion

Bitte aufstehen!



# Aktion

‘ [cK]h? a+t(ze)? ‘

# Aktion

' [acektz]\* '

# Aktion

‘(cat | katze)’

# Wie funktionieren Regular Expressions?

# HOW TO REGEX

STEP 1: OPEN YOUR FAVORITE EDITOR



STEP 2: LET YOUR CAT PLAY ON YOUR KEYBOARD



05.07.20219, <https://x.com/garabatokid/status/1147063121678389253>

# Die nächste Hürde

Regex != Regex

# Geschmacksrichtung: grep

## Regular Expressions - Cheat Sheet

### Definiton: Ausdruck

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

- beliebiges Zeichen; Ausnahme: Zeilenumbruch
- Zeichen, die enthalten sein dürfen: `[abc]` oder `[a-c]` => Findet a,b,c,d, aber nicht f!
- Zeichen, die nicht enthalten sein dürfen: `[!abc]` oder `[!a-c]` => Findet f, aber nicht a,b,c,d.

### Gruppierungen

- `( )` Ausdrücke werden gruppiert und können über Backreference wieder verwendet werden.
- `\1` Backreference, die (hier) auf die erste Gruppe zugreift.
- Auf die Gruppe kann nicht per Backreference zugegriffen werden.

Das Ergebnis beginnt mit einem 'c' oder 'k'. Es folgt mind. ein 'h', dann ein 't'.

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

### Quantifier

Die Quantifier beziehen sich immer auf das vorherige Zeichen.  
`grep -P 'Linux?'` <- das '?' ist optional  
 => Mögliche Ergebnisse: Linu s oder Linux

- `?` Der vorh. Ausdruck ist optional. Wiederholung 0 oder 1
- `+` Der vorh. Ausdruck muss mind. 1x vorkommen. Wiederholung >= 1
- `*` Der vorh. Ausdruck muss nicht und kann beliebig oft gefunden werden. Wiederholung >= 0
- `{n}` Der vorh. Ausdruck wird mind. n-mal gefunden. Wiederholung >= n
- `{n,m}` Der vorh. Ausdruck wird mind. aber höchstens m-mal gefunden. Wiederholung >= n, <= m
- `{n}` Der vorh. Ausdruck wird genau n-mal gefunden. Wiederholung = n
- Alternative: `|`  
`grep -E 'c|k'` => sucht nach 'c|' oder 'k|'. `'c|k'` wird nicht gefunden.

### Zeichenklassen

- `\w` Findet Buchstaben, Zahlen oder Unterstriche.
- `\W` Findet alle Zeichen **außer** Buchstaben, Zahlen und Unterstriche.
- `\d` Findet Zahlen.
- `\D` Findet alle Zeichen **außer** Zahlen.
- `\s` Findet Whitespaces (Leerzeichen, Tabs).
- `\S` Findet **nicht** Whitespaces.

### Grenzen

- `^` Der Ausdruck steht am Zeilenanfang.
- `$` Der Ausdruck steht am Zeilenende.
- `^$` => Matched auf eine leere Zeile.
- `^.*$` => Matched auf alles.
- `(word boundary)` Findet Zeichenketten, an deren Anfang oder Ende Whitespace steht: `\b` oder `\B`  
`\b` => Sucht nach ' p' in ' penguin' => penguin matched nicht.
- `\b` Findet Zeichenketten, an deren Anfang oder Ende **kein** Whitespace steht: `\b` oder `\B`  
`echo "denken entdecken ende" | grep "\bde\b"` => denken entdecken ende

### Escaping

Escaping ist notwendig, wenn auf Zeichen geffert 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 "(zhoo)"` <- Beide Ausdrücke suchen nach (zhoo).
- `grep -E "(zhoo)"` <- Beide Ausdrücke suchen nach der Gruppe zh.

### Delimiter

In einigen Programmen (z.B. sed oder via) kann der Delimiter (Trenner) frei gesetzt werden. Ein Escaping des Delimiters kann so vermieden werden.  
`echo "cifs://server/path/file" | sed -e 's//R//g' <- '/' ist der Delimiter`

### Legende

Der Regex-Funktionsumfang unterscheidet sich je nach Tool und Programmiersprache.  
`\G` erfordert escaping in `grep -P` vorhanden `P` nur in `grep -P` vorhanden



- #### 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 (PCRE).
  - `-F` `--fixed-strings` Interpret PATTERNS as fixed strings, not regular expressions.

### Flags

- Flags (Schalter) sind ein optionaler Parameter, um das Verhalten des Ausdrucks zu verändern.
- `/B` **global matching** Stoppt nicht nach dem ersten Treffer, sondern findet alle Übereinstimmungen.
- `/i` **case-insensitive** Groß- oder Kleinschreibung wird ignoriert.
- `/n` **multiline mode** Jede Zeile wird separat behandelt.

### Lookaround

- Mit einem Lookaround kann ein Ausdruck gesucht werden, der (NICHT) vor oder hinter einem anderen Ausdruck steht.
- `\d(?!3)` => Sucht nach mind. einer Zahl, an deren Anfang oder Ende Whitespace steht: `\d(?!3)` => Sucht nach ' p' in ' penguin' => penguin matched nicht.
- `(?=)` positiver Lookahead
- `(?!)` negativer Lookahead
- `(?<=)` positiver Lookbehind
- `(?<!)` negativer Lookbehind



powered by CC-BY-SA: Marie Mann, Handout zum Vortrag "Eine Katze läuft über meine Tastatur" - Eintrag in RegEX! <https://git.pengutronix.de/cgit/mcm/regexcxs/plain/regex-spickzettel.pdf>

<https://git.pengutronix.de/cgit/mcm/regexcxs/plain/regex-spickzettel.pdf>



# Syntax

‘(cat|katze)’

# Syntax: Alternative

'(cat | katze)'

## Regular Expressions - Cheat Sheet

**Definition: Ausdruck**  
 Suchmuster, das eine bestimmte Menge von Zeichenketten beschreibt und erkennt.

**Gruppierungen**

- beliebiger Zeichen: `[ ]`
- Ausdrücke: `{ } | ( )`
- Zeichen, die enthalten sein dürfen: `[ ]` oder `[ ]` oder `[ ]` oder `[ ]`
- Zeichen, die nicht enthalten sein dürfen: `[ ]` oder `[ ]` oder `[ ]` oder `[ ]`

**Escaping**  
 Escaping ist notwendig, wenn auf Zeichen gefolgt 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 -E "(Z|z)h"` <- beide Ausdrücke suchen nach `(Z|z)h`.

`grep -E "\Z"` <- beide Ausdrücke suchen nach `\Z`.

`grep -E "\Z"` <- beide Ausdrücke suchen nach der `Zeichenkategorie`.

**Legende**  
 Die RegEx Funktionsumfänge unterscheiden sich je nach Tool und Programmiersprache.

`[ ]` erfordert escaping  
`[ ]` nur in `grep -P` in `grep -E`



Das Ergebnis beginnt mit einem `^` oder `^`.  
 Es folgt ein `.` für `^`, dann ein `^`.

`'[c]k[h]{0,}a+t(ze)?'`

Quantifier  
 Die Quantifier beziehen sich immer auf das vorherige Zeichen.

`grep -P '^Linux$' -o -E 'x'` ist optional  
 => Mögliche Ergebnisse: `Linux` oder `Linux`

**Delimiter**  
 In einigen Programmen (z.B. sed oder vim) kann der Delimiter (Trennzeichen) festgesetzt werden. Ein Escaping des Delimiters kann so vermeiden werden.

`echo "Life is never really life"`  
`sed -e 's/ / /g'` <- `/` ist der Delimiter

**Grenzen**  
 Der Ausdruck steht am Zeilenanfang.  
 Der Ausdruck steht am Zeilenende.  
`^` <- Matche auf eine leere Zeile.  
`$` <- Matche auf alles.

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

- `-i` global matching: Sieht nicht nach dem ersten Treffer, sondern findet alle Übereinstimmungen.
- `-s` caseinsensitive: Groß- oder Kleinschreibung wird ignoriert.
- `-m` multiline mode: Jede Zeile wird separat behandelt.

**Zeichenklassen**

- `[a-zA-Z]` Findet Buchstaben, Zahlen oder Unterstriche.
- `[0-9]` Findet Zahlen.
- `[^a-zA-Z]` Findet alle Zeichen außer Zahlen.
- `[^0-9]` Findet alle Zeichen außer Zahlen.
- `[^a-zA-Z0-9]` Findet alle Zeichen außer Buchstaben, Zahlen und Unterstrichen.
- `[^0-9a-zA-Z]` Findet alle Zeichen außer Buchstaben, Zahlen und Unterstrichen.
- `[^a-zA-Z0-9_]` Findet alle Zeichen außer Buchstaben, Zahlen, Unterstrichen und Whitespaces (Leerzeichen, Tab).
- `[^a-zA-Z0-9_ ]` Findet alle Zeichen außer Buchstaben, Zahlen, Unterstrichen und Whitespaces.

**Lookaround**  
 Mit einem Lookaround kann ein Ausdruck geprüft werden, der nicht vor oder hinter einem anderen Ausdruck steht.

`^<[ ]` <- Sucht nach mind. einer Zahl, der ein `[ ]` Symbol folgt; z.B. `34`.

`[ ]>` <- positiver Lookahead  
`[ ]<` <- negativer Lookahead  
`[ ]<=` <- positiver Lookbehind  
`[ ]>=` <- negativer Lookbehind





# Syntax

`'[acektz]*'`





# Syntax

‘ [cK]h?a+t(ze)? ’

# Syntax: Zeichenauswahl

' [cK]h?a+t(ze)? '

## Regular Expressions - Cheat Sheet

**Definition: Ausdruck**  
 Suchmuster, das eine bestimmte Menge von Zeichenketten beschreibt und erkennt.

**Gruppierungen**

- ⌈⌋: Ausdrücke werden gruppiert und können über  $\backslash$  Backreference wieder verwendet werden.
- 1: Backreference, die (hier) auf die erste Gruppe reagiert.
- 2: Auf die Gruppe kann nicht per Backreference zugegriffen werden.

Das Ergebnis beginnt mit einem 'c' oder 'k'.  
 Es folgt mind. ein 'h', dann ein 'a'.

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

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

**grep -P 'Linux'** <- die 'e' ist optional  
 => Mögliche Ergebnisse: Linux oder Linux

Der vorh. Ausdruck ist optional.  $\{0,1\}$  Wiederholung > 1

Der vorh. Ausdruck muss mind. 1 vorkommen.  $\{1,\}$  Wiederholung > 1

Der vorh. Ausdruck muss nicht und kann beliebig oft gefunden werden.  $\{0,\}$  Wiederholung > 0

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

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

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

Alternativ:  $\{n,k\}$  sucht nach '{1,}' oder '{1,1}' nicht gefunden.

**Escaping**  
 Escaping ist notwendig, wenn auf Zeichen gefolgt 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 -E '(Z|H|N|)'** <- beide Ausdrücke suchen nach '(Z|H|N)'.  
**grep -E '(Z|H|N|)'** <- beide Ausdrücke suchen nach der Gruppe Z|H|N.

**grep -s** kann mit dem Funktionsumfang von **grep -E** genutzt werden, indem einige Parameter escaped werden.  
**grep -E '(Z|H|N|)'** <- beide Ausdrücke suchen nach der Gruppe Z|H|N.

**Delimiter**  
 In einigen Programmen (z.B. sed oder vi) kann der Delimiter (Trennzeichen) festgesetzt werden. Ein Escaping des Delimiters kann so vermieden werden.

**echo "Linux//server/path/file"**  
**sed -e 's//a//g'** <- // ist der Delimiter

**Zeichenklassen**

- $\backslash w$  Findet Buchstaben, Zahlen oder Unterstriche.
- $\backslash W$  Findet alle Zeichen außer Buchstaben, Zahlen und Unterstriche.
- $\backslash d$  Findet Zahlen.
- $\backslash D$  Findet alle Zeichen außer Zahlen.
- $\backslash s$  Findet Whitepsace (Leerzeichen, Tab).
- $\backslash S$  Findet nicht-Whitepsace.

**Grenzen**

- $\wedge$  Der Ausdruck steht am Zeilenanfang.
- $\$$  Der Ausdruck steht am Zeilenende.
- $\$>$  >= Matche auf eine leere Zeile.
- $\$<$  <= Matche auf alles.
- $\b$  word boundary / findet Zeichenketten, an deren Anfang oder Ende whitespaces steht.
- $\B$  >= suche nach '\b' >= erfolgreich matched nicht.
- $\B$  Findet Zeichenketten, an deren Anfang oder Ende kein whitespaces steht.
- $\B$  <= denken an 'leeren ende' <= '\B' <= '\B'

**Legende**

Die RegEx Funktionsumfang unterscheidet sich je nach Tool und Programmiersprache.

$\{ \}$  erfordert escaping  $\{ \}$  nur in grep -P  
 $\{ \}$  nur in grep -P  
 $\{ \}$  in grep -E

**man grep**  
 searches for PATTERNS in each FILE  
**grep** -ball-c-egpr  
 Interpret PATTERNS as basic regular expressions.

**-E** --extended-regex  
 Interpret PATTERNS as extended regular expressions.

**-P** --perl-regex  
 Interpret PATTERNS as Perl-compatible regular expressions (PCRE).

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

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

- $-i$  global matching: Sucht nicht nach dem ersten Treffer, sondern findet alle Übereinstimmungen.
- $-s$  caseinsensitive: Groß- oder Kleinschreibung wird ignoriert.
- $-m$  multiline mode: Jede Zeile wird separat behandelt.

**Lookaround**

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

$\{ \}$  sucht nach mind. einer Zahl, der ein 'e' Symbol folgt; z.B. 3e.

$\{ \}$  positiver Lookahead  
 $\{ \}$  negativer Lookahead  
 $\{ \}$  positiver Lookbehind  
 $\{ \}$  negativer Lookbehind



# Syntax: Quantifier

' [cK]h?a+t(ze)? '

## Regular Expressions - Cheat Sheet

**Definition: Ausdruck**  
 Suchmuster, das eine bestimmte Menge von Zeichenketten beschreibt und erkennt.

**Gruppierungen**

- beliebiges Zeichen: `[ ]` oder `[...]` oder `[...]` oder `[...]`
- Zeichen, die enthalten sein dürfen: `[ ]` oder `[...]` oder `[...]` oder `[...]`
- Zeichen, die nicht enthalten sein dürfen: `[ ]` oder `[...]` oder `[...]` oder `[...]`

**Escaping**  
 Escaping ist notwendig, wenn auf Zeichen gefolgt 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 -E "(ZNEB)"` -> beide Ausdrücke suchen nach `(ZNEB)`.

`grep -E "\(ZNEB)"` -> beide Ausdrücke suchen nach `(ZNEB)`.

`grep -E "(ZNEB)"` -> beide Ausdrücke suchen nach `(ZNEB)`.

`grep -E "\(ZNEB)"` -> beide Ausdrücke suchen nach `(ZNEB)`.

**Legende**  
 Die RegEx-Funktionsumfang unterscheidet sich je nach Tool und Programmiersprache.

`[ ]` erfordert escaping `[ ]` nur in `grep -P` in `grep -E`

**man grep**  
 Search for PATTERNS in each FILE.  
`grep -s` search for PATTERNS in each FILE.  
`grep -i` ignore case distinctions in PATTERNS.  
`grep -E` interpret PATTERNS as extended regular expressions.  
`grep -F` interpret PATTERNS as fixed strings, not regular expressions.  
`grep -P` interpret PATTERNS as Perl-compatible regular expressions (PCREs).  
`grep -F -P` interpret PATTERNS as fixed strings, not regular expressions.

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

- `/i` global matching: Sucht nicht nur nach dem ersten Treffer, sondern findet alle Übereinstimmungen.
- `/i` caseinsensitive: Groß- oder Kleinschreibung wird ignoriert.
- `/m` multiline mode: Jede Zeile wird separat behandelt.

**Lookaround**  
 Mit einem Lookaround kann ein Ausdruck geprüft werden, der nicht vor oder hinter einem anderen Ausdruck steht.

`^(?=...)` -> Sucht nach mind. einer Zahl, der ein `E` Symbol folgt; z.B. `3E`.

`(?=...)` positiver Lookahead  
`(?!...)` negativer Lookahead  
`(<=...)` positiver Lookbehind  
`(<?!...)` negativer Lookbehind

**Quantifier**  
 Die Quantifier bezeichnen sich immer auf die vorherigen Zeichen.

Das Ergebnis beginnt mit einem `'` oder `"`.  
 Es folgt mind. `'` oder `"`, dann ein `'` oder `"`.

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

An der nächsten Stelle wird ein `'` oder `"` oder `'` oder `"` folgen.

**Delimiter**  
 In einigen Programmen (z.B. `sed` oder `awk`) kann der Delimiter (Trennzeichen) festgesetzt werden. Ein Escaping des Delimiters kann so verwendet werden.

`awk 'NR==1{print}/^([a-z])$/'` -> ist der Delimiter

**Größen**

- `^` Der Ausdruck steht am Zeilenanfang.
- `$` Der Ausdruck steht am Zeilenende.
- `^$` -> Matche auf eine leere Zeile.
- `^$` -> Matche auf alles.
- `^b` (word boundary) findet Zeichenketten, an deren Anfang oder Ende whitespaces steht.
- `^B` (word boundary) findet Zeichenketten, an deren Anfang oder Ende kein whitespaces steht.
- `^b` (word boundary) findet Zeichenketten, an deren Anfang oder Ende whitespaces steht.
- `^B` (word boundary) findet Zeichenketten, an deren Anfang oder Ende kein whitespaces steht.

CC-BY-SA Maria Maria, Handout zum Vortrag "Eine kleine Welt über meine Tastatur" - Eintrag in RegEx.  
<https://gitlab.com/gregor.v.d.lindt/notes/regular-expressions-cheat-sheet>





# Syntax: Escaping

## Regular Expressions - Cheat Sheet

### Definition: Ausdruck

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

- beliebiges Zeichen  
Ausnahme: Zeilenumbruch
- [abcd] Zeichen, die enthalten sein dürfen  
[a-b] oder [a-d] => Findet a,b,c,d,  
aber nicht f.
- [^abc] Zeichen, die nicht enthalten sein dürfen  
[!a-d] oder [!a-d] => Findet f, aber  
nicht a,b,c,d.

### Gruppierungen

- ( ) Ausdrücke werden gruppiert und können über \G Backreference weiter verwendet werden.
- | Backreference, die (hier) auf die erste Gruppe zugeht.  
s/(Lars)\(Fridolin Lars)\A\*/g  
=> Aus Lars Fridolin Lars wird Fridolin Lars T
- (?: ) Auf die Gruppe kann nicht per Backreference zugegriffen werden.

Das Ergebnis beginnt mit einem 'c' oder 'k'.  
Es folgt mind. ein 'a', dann ein 't'.  
`'[cK]h{0,}a+t(ze)?'`

### Quantifier

Die Quantifier beziehen sich immer auf das vorherige Zeichen.  
`grep -P 'Linux?'` <- das 'x' ist optional  
=> Mögliche Ergebnisse: Linux 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 'ct{1,3}'` => sucht nach 'ct', 'ctt' oder 'ctt', 'ctt' wird nicht gefunden.

An der nächsten Stelle wird ein 'b' @-mal oder mehrmals gefunden.  
Die Zeichen 'ze' sind optional.

### Zeichenklassen

- \w Findet Buchstaben, Zahlen oder Unterstriche
- \W Findet alle Zeichen außer Buchstaben, Zahlen und Unterstriche.
- \d Findet Zahlen. \G
- \D Findet alle Zeichen außer Zahlen. \G
- \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', penguin matched nicht.
- \B Findet Zeichenketten, an deren Anfang oder Ende kein Whitespace steht:  
echo "denken entdecken ende" | grep '\Bde\B'  
=> denken entdecken ende

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

### Delimitter

In einigen Programmen (z.B. sed oder vim) kann der Delimitter (Trenner) freigesetzt werden. Ein Escaping des Delimiters kann so vermieden werden.  
echo "cifs://server/path/file" | sed -e 's//#/#g' <- '#' ist der Delimitter

### Legende

Der Regex-Funktionsumfang unterscheidet sich je nach Tool und Programmiersprache.  
[G] erfordert escaping P nur in grep -P vorhanden



Linux-Tage  
23. und 24. März 2023

### man grep

```
grep searches for PATTERNS in each FILE  
-G, --basic-regex Interpret PATTERNS as basic regular expressions.  
-E, --extended-regex Interpret PATTERNS as extended regular expressions.  
-P, --perl-regex Interpret PATTERNS as Perl-compatible regular expressions (PCRES).  
-F, --fixed-strings Interpret PATTERNS as fixed strings, not regular expressions.
```

### 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

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. 36.

- (?=) positiver Lookahead
- (?!) negativer Lookahead
- (<=) positiver Lookbehind
- (<?) negativer Lookbehind



# Übungen

programm-clt-2025.txt

So, 14:00 Uhr, V3: "Einstieg  
in Regex" (Marie Mann)



# Aufgaben zu programm-clt-2025.txt

- Suche alle Vorträge mit Linux im Titel.
- Suche Vorträge, deren Referierende Tim oder Tom heißen.
- Suche Vorträge, bei dem im Titel ein Doppel-Vokal vorkommt.
- Suche Vorträge, in denen kein 'ai' oder 'ki' vorkommt.
- Suche Vorträge, in deren Titel ein Wort mit Bindestrich vorkommt (z.B. Bash-Historie).
- Suche Vorträge, in deren Titel ' - ' enthalten ist.
- Suche Vorträge, die zwischen 9 und 11 Uhr beginnen.
- Suche Vorträge, die ein Frage- oder Ausrufezeichen im Titel haben.
- Suche Vorträge, die Zahlen im Titel haben.
- Suche Vorträge, deren Titel nur aus einem Wort besteht.
- Suche Vorträge, die von mehr als einer Person gehalten werden.
- Suche Vorträge, die um 12 Uhr im Erdgeschoss (V4, V5, V6) stattfinden.
- Gib nur den Titel des Vortrags von "Senf" aus.
- Suche ungültige Einträge.
- Suche Vorträge, von denen der Nachname der Referierenden fünf Buchstaben hat.
- Suche die Vornamen der Referierenden (ein Name je Vortrag).

# Verwendung von Regular Expressions



# Verwendung von Regular Expressions

- **Nutzt Libraries**
- Ausführliche Kommentare
- Positive + Negative Ergebnisse

# Verwendung von Regular Expressions

- Nutzt Libraries
- **Ausführliche Kommentare**
- Positive + Negative Ergebnisse

# Verwendung von Regular Expressions

- Nutzt Libraries
- Ausführliche Kommentare
- **Positive + Negative Ergebnisse**

# Regular Expressions lernen

- <https://regexone.com/>
- <https://regexlearn.com/>

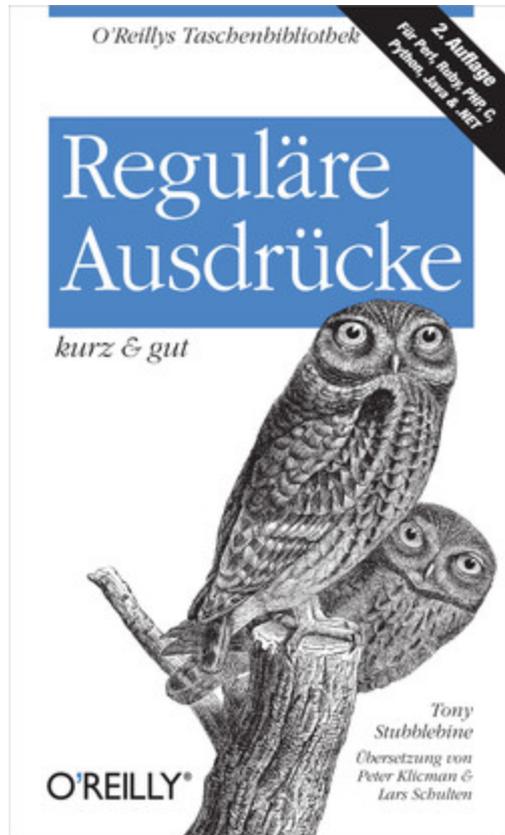
# Regular Expressions testen + debuggen

- <https://regex101.com/>
- [https://ivanzuzak.info/  
noam/webapps  
fsm\\_simulator/](https://ivanzuzak.info/noam/webapps/fsm_simulator/)

# Spaß mit Regular Expressions

- <https://regexcrossword.com/>
- <https://nicholas.carlini.com/writing/2025/regex-chess.html>

# Literatur-Tipps



Stubblebine: „Reguläre Ausdrücke – kurz & gut“, O'Reilly



Sommer, Kania, Wolf: „Shell-Programmierung“, Rheinwerk

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

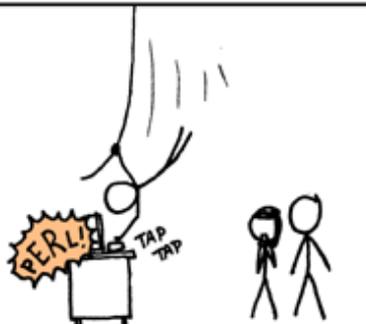


IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.



<https://xkcd.com/208/>

