HOWTO zur Perl-Bit-Operatoren und Booleschen Operatoren

(C) 2008-2023 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: perl-bit-boole-op-HOWTO.txt,v 1.9 2025/02/23 20:14:55 tsbirn Exp $

Dieses Dokument beschreibt die in Perl verfügbaren Operatoren für Bit-Operationen & | ^ ~ << >> und vergleicht sie mit Boolesche Operationen && || ! sowie and or not xor.

INHALTSVERZEICHNIS

1) Bit-Operatoren
2) Boolesche Operatoren
3) Beispiele für die Bit-Verknüpfungen
4) Beispiele für die Booleschen Verknüpfungen

---

1) Bit-Operatoren

Die Bit-Operatoren verarbeiten ihre Operanden bitweise, d.h. jede Bitposition wird für sich verarbeitet (NOT, LEFTSHIFT, RIGHTSHIFT) oder einzeln mit der korrespondierenden Bitposition des anderen Operators verknüpft (AND, OR, XOR).

Operator	Name	Bedeutung
A & B	AND	Resultat-Bit "1" wenn beide Bits "1"
A | B	OR	Resultat-Bit "1" wenn eines der Bits "1"
A ^ B	XOR	Resultat-Bit "1" wenn ein Bit "1" und eines "0"
~ A	NOT	Resultat-Bit "1" wenn Bit "0" und umgekehrt
A << N	LEFTSHIFT	Bits um N Stellen nach links schieben (0 nach)
A >> N	RIGHTSHIFT	Bits um N Stellen nach rechts schieben (0 nach)

Bitte nicht verwechseln mit den logischen (Booleschen) Operatoren (&& || ! and or not, xor), die ihre Operanden nicht bitweise, sondern insgesamt als TRUE oder FALSE verarbeiten.

2) Boolesche Operatoren

Verarbeiten ihre Operanden als Wert TRUE (alle anderen Werte) oder Wert FALSE (0 "" "0" () undef) und liefern als Ergebnis wieder TRUE (meist "1") oder FALSE (meist "") (genauer: einen der beiden Operanden mit dieser Bedeutung):

Operator	Alternativ	Bedeutung
A && B	A and B	Resultat TRUE wenn beide Operanden TRUE
A || B	A or B	Resultat TRUE wenn einer der Operanden TRUE
! A	not A	Resultat TRUE wenn Operand FALSE (negieren)
	A xor B	Resultat TRUE wenn ein Operand TRUE und einer FALSE

Die Operatoren "&&" und "and", "||" und "or" sowie "!" und "not" haben exakt die gleiche Funktion BIS AUF DEN VORRANG. Während "&&", "||" und "!" VOR den Zuweisungs- (= -= += ...) und Listen-Operatoren (, => print sort ...) angesiedelt sind, sind "and", "or" und "not" NACH ALLEN Operatoren angesiedelt.

Dies ist einerseits in gewissen Situationen von Vorteil, kann andererseits
aber zu subtilen Fehlern führen. Ein einfacher Austausch von "&&" <-> "and"
bzw. "||" <-> "or" bzw. "!" <-> "not" in einem Programm führt normalerweise
zu einem anderen Programmverhalten. Durch Klammerung können diese Unterschiede
aber behoben werden. Beispiel:

```
  &fields("create", $cont1,  $cont2 or $cont3,  $empty);   # PENG!
  &fields("create", $cont1, ($cont2 or $cont3), $empty);   # OK!
  &fields("create", $cont1,  $cont2 || $cont3,  $empty);   # OK!

  $readonly =  $type ne "copy" and $type ne "rename";   # PENG!
  $readonly = ($type ne "copy" and $type ne "rename");  # OK!
  $readonly =  $type ne "copy" &&  $type ne "rename";   # OK!
```

3) Beispiele für die Bit-Verknüpfungen
---

* "&" (Bitweise AND)

```
                      +-----+---+---+
                      | AND | 0 | 1 |
     11001100         +-----+---+---+    0 & 0 --> 0
     10101010         |  0  | 0 | 0 |    0 & 1 --> 0
     --------         +-----+---+---+    1 & 0 --> 0
     10001000         |  1  | 0 | 1 |    1 & 1 --> 1
                      +-----+---+---+
```

* "|" (Bitweise OR)

```
                      +-----+---+---+
                      |  OR | 0 | 1 |
     11001100         +-----+---+---+    0 | 0 --> 0
     10101010         |  0  | 0 | 1 |    0 | 1 --> 1
     --------         +-----+---+---+    1 | 0 --> 1
     11101110         |  1  | 1 | 1 |    1 | 1 --> 1
                      +-----+---+---+
```

* "^" (Bitweise XOR = entweder-oder, exclusive or)

```
                      +-----+---+---+
                      | XOR | 0 | 1 |
     11001100         +-----+---+---+    0 ^ 0 --> 0
     10101010         |  0  | 0 | 1 |    0 ^ 1 --> 1
     --------         +-----+---+---+    1 ^ 0 --> 1
     01100110         |  1  | 1 | 0 |    1 ^ 1 --> 0
                      +-----+---+---+
```

* "~" (Bitweise NOT = invertieren)

```
                      +-----+---+---+
                      | NOT | 0 | 1 |
     11001100         +-----+---+---+    ~ 0 --> 1
     --------         |     | 1 | 0 |    ~ 1 --> 0
     00110011         +-----+---+---+
```

* "<<" (Bitweise LEFTSHIFT = Multiplikation mit 2)

```
     11001100  (204)      10101010111  (1367)
    ---------            ------------
    110011000  (408)     101010101110  (2734)
```

* ">>" (Bitweise RIGHTSHIFT = Division durch 2)

```
     11001100  (204)      10101010111  (1367)
    ---------            ------------
     01100110  (102)      01010101011   (683)
```

4) Beispiele für die Booleschen Verknüpfungen
---

* F = FALSE = 5 Werte in Perl: 0 "" "0" () undef
  T = TRUE = ALLE anderen Werte in Perl

* "&&" bzw. "and" (logisches AND)

```
+-----+---+---+
| AND | F | T |
+-----+---+---+    T and T --> T
|  F  | F | F |    T and F --> F
+-----+---+---+    F and T --> F
|  T  | F | T |    F and F --> F
+-----+---+---+
```

* "||" bzw. "or" (logisches OR)

```
+-----+---+---+
|  OR | T | F |
+-----+---+---+    T or T --> T
|  T  | T | F |    T or F --> T
+-----+---+---+    F or T --> T
|  F  | F | F |    F or F --> F
+-----+---+---+
```

* "xor" (logisches XOR = entweder-oder, exclusive or)

```
+-----+---+---+
| XOR | T | F |
+-----+---+---+    T xor T --> F
|  T  | T | F |    T xor F --> T
+-----+---+---+    F xor T --> T
|  F  | F | T |    F xor F --> F
+-----+---+---+
```

* "!" oder "not" (logisches NOT = negieren)

```
+-----+---+---+
| NOT | T | F |
+-----+---+---+    not T --> F
|     | F | T |    not F --> T
+-----+---+---+
```