HOWTO zum Kommando "expr"

(C) 2006-2025 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: shell-expr-HOWTO.txt,v 1.2 2025/02/23 20:14:55 tsbirn Exp $

INHALTSVERZEICHNIS

0) Einführung
1) "expr"-Operatoren
1.1) Arithmetik
1.2) Numerischer Vergleich
1.3) Text-Vergleiche
1.4) Inkrementieren / Dekrementieren
1.5) Logische Verknüpfung (es gibt KEINE logische Negation!)
1.6) Shortcut/Short Circuit-Evaluation von logischen Ausdrücken
1.7) Klammerung
1.8.1) String-Operationen
1.8.2) GNU-String-Operationen
1.9) Alternativen zum Rechnen ohne "expr" (nur in bash + ksh möglich)
1.9.1) Inkrement/Dekrement ohne "expr"
1.10) Fließkomma-Berechnungen (bc/dc = binary/decimal calculator)

---

0) Einführung

"expr" (expression) is a command which calculates the value of an expression
(numerical, string, comparison, logical, ...) and prints it out on the terminal:

```
  Numerical expression --> Result of calculation
  String expression    --> Result of string operation
  Comparison           --> Result "1" if true, "0" if false (opposite of exit status)
  Logical expression   --> Result "1" if true, "0" if false (opposite of exit status)
```

"expr" supports only INTEGER numbers. For FLOATING POINT numbers tools like
"bc" (binary calculator), "dc" (decimal calculator) or "awk" have to be used.

Furthermore "expr" produces an EXIT STATUS:

```
  0 = Expression syntactically correct and result is NOT 0 and NOT empty
  1 = Expression syntactically correct and result is     0 or      empty
  2 = Expression syntactically wrong
  3 = An error occurred
```

The elements of an expression have to be separated by SPACES.
Shell special characters (e.g. * & | ( ) < >) have to be QUOTED.

A lot of replacements for "expr" have been added to the shell over time
(faster, more features, don't need/allow spaces inside expression, ...):

```
  declare -i VAR   # Variable of type integer --> Assignment evaluated as expression
  let VAR=...      # Numeric expression
  VAR=$[ ... ]     # Numeric expression
  (( VAR = ... ))  # Numeric expression
  VAR=$(( ... ))   # Numeric expression
```

1) "expr"-Operatoren

1.1) Arithmetik

```
  expr 10 + 3      # --> 13  Addition
  expr 10 - 3      # --> 7   Subtraktion
  expr 10 \* 3     # --> 30  Multiplikation
  expr 10 / 3      # --> 3   Division
  expr 10 % 3      # --> 1   Modulo (Divisionsrest)
```

1.2) Numerischer Vergleich

```
  expr 10 = 3      # --> 1  Zahl gleich             (Exit status 0)
  expr 10 != 3     # --> 0  Zahl ungleich           (Exit status 1)
  expr 10 \< 3     # --> 1  Zahl kleiner            (Exit status 0)
  expr 10 \<= 3    # --> 1  Zahl kleiner gleich     (Exit status 0)
  expr 10 \> 3     # --> 0  Zahl größer             (Exit status 1)
  expr 10 \>= 3    # --> 0  Zahl größer gleich      (Exit status 1)
```

1.3) String-Vergleich

```
  expr abc =   def   # --> 0  String gleich         (Exit status 1)
  expr abc !=  def   # --> 1  String ungleich       (Exit status 0)
  expr abc \<  def   # --> 1  String kleiner        (Exit status 0)
  expr abc \<= def   # --> 1  String kleiner gleich (Exit status 0)
  expr abc \>  def   # --> 0  String größer         (Exit status 1)
  expr abc \>= def   # --> 0  String größer gleich  (Exit status 1)
```

1.4) Inkrementieren / Dekrementieren

```
  COUNT=`expr $COUNT + 1`    # COUNT um 1 erhöhen (Inkrement)
  COUNT=$(expr $COUNT - 1)   # COUNT um 1 verringern (Dekrement)
```

1.5) Logische Verknüpfung (es gibt KEINE logische Negation!)

```
  expr 10 \& 2      # --> 10 AND  (Exit status 0)
  expr 10 \& 0      # -->  0 AND  (Exit status 1)
  expr  0 \& 2      # -->  0 AND  (Exit status 1)
  expr  0 \& 0      # -->  0 AND  (Exit status 1)
  expr 10 \| 2      # --> 10 OR   (Exit status 0)
  expr 10 \| 0      # --> 10 OR   (Exit status 0)
  expr  0 \| 2      # -->  2 OR   (Exit status 0)
  expr  0 \| 0      # -->  0 OR   (Exit status 1)
```

1.6) Shortcut/Short Circuit-Evaluation von logischen Ausdrücken

```
  expr 5 \> 7 \| 3 + 2    # --> 5  (Exit status 0)
  expr 5 \> 7 \& 3 + 2    # --> 0  (Exit status 1)
```

1.7) Klammerung

```
  expr \( 1 + 2 \) \* 3    # --> 9
  expr    1 + 2    \* 3    # --> 7
```

1.8.1) String-Operationen

```
  expr "abcde" : ".*"              # --> 5    Match Länge
  expr "abcde" : "abc"             # --> 3    Match Länge
  expr "abcde" : '\(.*c\)'         # --> abc  Match Muster
```

1.8.2) GNU-String-Operationen

```
  expr length "abcde"              # --> 5    String Länge
  expr index "abcde" "cd"          # --> 3    Match start Index
  expr substr "abcde" 2 3          # --> bcd  Substring ab Index 2 mit Länge 3
  expr match "abcde" "abc"         # --> 3    Match Länge
  expr match "abcde" "\(.*c\)"     # --> abc  Match Muster
```

1.9) Alternativen zum Rechnen ohne "expr" (nur in bash + ksh möglich)

```
  declare -i SUM                  #          1a) Variable hat Typ Integer
  SUM=111+111                     # --> 222  1b) KEINE Leerzeichen erlaubt!
  unset SUM                       #          2+3+4) Variable hat keinen Typ
  let SUM=111+222                 # --> 333  2a) KEINE Leerzeichen erlaubt!
  let SUM=" 111 + 333 "           # --> 444  2b) Leerzeichen erlaubt!
  SUM=$(( 111 + 444))             # --> 555  3a) Leerzeichen erlaubt!
  ((SUM = 111 + 555))             # --> 666  3b) Leerzeichen erlaubt!
  SUM=$[ 111 + 666 ]              # --> 777  4) Leerzeichen erlaubt, veraltet!
```

1.9.1) Inkrement/Dekrement ohne "expr"

```
  N=5                             # --> 5
  ((++N))                         # --> 6   Inkrement (prefix)
  ((N++))                         # --> 7   Inkrement (postfix)
  ((--N))                         # --> 6   Dekrement (prefix)
  ((N--))                         # --> 5   Dekrement (postfix)
  ((N += 5))                      # --> 10  Zusammengesetzte Addition
  ((N -= 5))                      # --> 10  Zusammengesetzte Subtraktion
  ((N *= 5))                      # --> 25  Zusammengesetzte Multiplikation
  ((N /= 5))                      # --> 5   Zusammengesetzte Division
  ((N %= 5))                      # --> 0   Zusammengesetzte Modulo/Divisionsrest
```

1.10) Fließkomma-Berechnungen (bc/dc = binary/decimal calculator)

```
  RES=$(echo "1.2 + 3.4 - 5.6 * 7.8" | bc -S6)   # --> -39.08 (-S6 = 6 digits)
  RES=$(bc -S6 -e "1.2 + 3.4 - 5.6 * 7.8")       # --> -39.08 (-e = expression)
  RES=$(echo "1.2 3.4 + 5.6 7.8 * - p" | dc)     # --> -39.08
  RES=$(dc -S6 -e "1.2 3.4 + 5.6 7.8 * - p")     # --> -39.08 (p = print)
  RES=$(awk "BEGIN { print 1.2 + 3.4 - 5.6 * 7.8 }")              # --> -39.08
  RES=$(echo "BEGIN { print 1.2 + 3.4 - 5.6 * 7.8 }" | awk -f -)  # --> -39.08
```