NAME

expr - evaluate expression

SYNOPSIS

expr [**-e**] *expression*

DESCRIPTION

The **expr** utility evaluates *expression* and writes the result on standard output.

All operators and operands must be passed as separate arguments. Several of the operators have special meaning to command interpreters and must therefore be quoted appropriately. All integer operands are interpreted in base 10 and must consist of only an optional leading minus sign followed by one or more digits (unless less strict parsing has been enabled for backwards compatibility with prior versions of **expr** in FreeBSD).

Arithmetic operations are performed using signed integer math with a range according to the C *intmax_t* data type (the largest signed integral type available). All conversions and operations are checked for overflow. Overflow results in program termination with an error message on stdout and with an error status.

The **-e** option enables backwards compatible behaviour as detailed below.

Operators are listed below in order of increasing precedence; all are left-associative. Operators with equal precedence are grouped within symbols ‘{’ and ‘}’.

expr1 | *expr2*
: Return the evaluation of *expr1* if it is neither an empty string nor zero; otherwise, returns the evaluation of *expr2* if it is not an empty string; otherwise, returns zero.

expr1 & *expr2*
: Return the evaluation of *expr1* if neither expression evaluates to an empty string or zero; otherwise, returns zero.

expr1 {=, >, >=, <, <=, !=} *expr2*
: Return the results of integer comparison if both arguments are integers; otherwise, returns the results of string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relation is true, or 0 if the relation is false.

expr1 {+, -} *expr2*
: Return the results of addition or subtraction of integer-valued arguments.

expr1 {*, /, %} *expr2*
Return the results of multiplication, integer division, or remainder of integer-valued arguments.

expr1 : *expr2*
The ":" operator matches *expr1* against *expr2*, which must be a basic regular expression. The regular expression is anchored to the beginning of the string with an implicit "^".

If the match succeeds and the pattern contains at least one regular expression subexpression "\(...\)", the string corresponding to "\1" is returned; otherwise the matching operator returns the number of characters matched. If the match fails and the pattern contains a regular expression subexpression the null string is returned; otherwise 0.

Parentheses are used for grouping in the usual manner.

The **expr** utility makes no lexical distinction between arguments which may be operators and arguments which may be operands. An operand which is lexically identical to an operator will be considered a syntax error. See the examples below for a work-around.

The syntax of the **expr** command in general is historic and inconvenient. New applications are advised to use shell arithmetic rather than **expr**.

Compatibility with previous implementations

Unless FreeBSD 4.x compatibility is enabled, this version of **expr** adheres to the POSIX Utility Syntax Guidelines, which require that a leading argument beginning with a minus sign be considered an option to the program. The standard **--** syntax may be used to prevent this interpretation. However, many historic implementations of **expr**, including the one in previous versions of FreeBSD, will not permit this syntax. See the examples below for portable ways to guarantee the correct interpretation. The check_utility_compat(3) function (with a *utility* argument of "expr") is used to determine whether backwards compatibility mode should be enabled. This feature is intended for use as a transition and debugging aid, when **expr** is used in complex scripts which cannot easily be recast to avoid the non-portable usage. Enabling backwards compatibility mode also implicitly enables the **-e** option, since this matches the historic behavior of **expr** in FreeBSD. This option makes number parsing less strict and permits leading white space and an optional leading plus sign. In addition, empty operands have an implied value of zero in numeric context. For historical reasons, defining the environment variable EXPR_COMPAT also enables backwards compatibility mode.

ENVIRONMENT

EXPR_COMPAT If set, enables backwards compatibility mode.

EXIT STATUS

The **expr** utility exits with one of the following values:

0 the expression is neither an empty string nor 0.

1 the expression is an empty string or 0.

2 the expression is invalid.

EXAMPLES

- The following example (in sh(1) syntax) adds one to the variable *a*:

  ```
  a=$(expr $a + 1)
  ```

- This will fail if the value of *a* is a negative number. To protect negative values of *a* from being interpreted as options to the **expr** command, one might rearrange the expression:

  ```
  a=$(expr 1 + $a)
  ```

- More generally, parenthesize possibly-negative values:

  ```
  a=$(expr \( $a \) + 1)
  ```

- With shell arithmetic, no escaping is required:

  ```
  a=$((a + 1))
  ```

- This example prints the filename portion of a pathname stored in variable *a*. Since *a* might represent the path */*, it is necessary to prevent it from being interpreted as the division operator. The // characters resolve this ambiguity.

  ```
  expr "//$a" : '.*/\(.*\)'
  ```

- With modern sh(1) syntax,

  ```
  "${a##*/}"
  ```

  expands to the same value.

The following examples output the number of characters in variable *a*. Again, if *a* might begin with a hyphen, it is necessary to prevent it from being interpreted as an option to **expr**, and *a* might be interpreted as an operator.

- To deal with all of this, a complicated command is required:

  ```
  expr \( "X$a" : ".*" \) - 1
  ```

- With modern sh(1) syntax, this can be done much more easily:

  ```
  ${#a}
  ```

  expands to the required number.

SEE ALSO

sh(1), test(1), check_utility_compat(3)

STANDARDS

The **expr** utility conforms to IEEE Std 1003.1-2008 ("POSIX.1"), provided that backwards compatibility mode is not enabled.

Backwards compatibility mode performs less strict checks of numeric arguments:

- An empty operand string is interpreted as 0.
- Leading white space and/or a plus sign before an otherwise valid positive numeric operand are allowed and will be ignored.

The extended arithmetic range and overflow checks do not conflict with POSIX's requirement that arithmetic be done using signed longs, since they only make a difference to the result in cases where using signed longs would give undefined behavior.

According to the POSIX standard, the use of string arguments *length*, *substr*, *index*, or *match* produces undefined results. In this version of **expr**, these arguments are treated just as their respective string values.

The **-e** flag is an extension.

HISTORY

An **expr** utility first appeared in the Programmer's Workbench (PWB/UNIX). A public domain version of **expr** written by Pace Willisson <*pace@blitz.com*> appeared in 386BSD-0.1.

AUTHORS

Initial implementation by Pace Willisson <*pace@blitz.com*> was largely rewritten by J.T. Conklin <*jtc@FreeBSD.org*>.