Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'expr.1p' command

$ man expr.1p

EXPR(1P) POSIX Programmer's Manual EXPR(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

expr ? evaluate arguments as an expression

SYNOPSIS

expr operand...

DESCRIPTION

The expr utility shall evaluate an expression and write the result to standard output.

OPTIONS

None.

OPERANDS

The single expression evaluated by expr shall be formed from the oper? and operands, as described in the EXTENDED DESCRIPTION section. The ap? plication shall ensure that each of the expression operator symbols:

( ) | & = > >= < <= != + - * / % :

and the symbols integer and string in the table are provided as sepa? rate arguments to expr.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of expr:

LANG Provide a default value for the internationalization vari? ables that are unset or null. (See the Base Definitions vol? ume of POSIX.1?2017, Section 8.2, Internationalization Vari? ables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regu? lar expressions and by the string comparison operators.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behav? ior of character classes within regular expressions.

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard er? ror.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The expr utility shall evaluate the expression and write the result, followed by a <newline>, to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The formation of the expression to be evaluated is shown in the follow? ing table. The symbols expr, expr1, and expr2 represent expressions formed from integer and string symbols and the expression operator sym? bols (all separate arguments) by recursive application of the con? structs described in the table. The expressions are listed in order of decreasing precedence, with equal-precedence operators grouped between horizontal lines. All of the operators shall be left-associative.

Expression	Description
integer	An argument consisting only of an (op? tional) unary minus followed by digits.
string	A string argument; see below.
(expr)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
expr1 : expr2	Matching expression; see below.
expr1 * expr2	Multiplication of decimal integer-valued arguments.
expr1 / expr2	Integer division of decimal integer-val? ued arguments, producing an integer re? sult.
expr1 % expr2	Remainder of integer division of decimal integer-valued arguments.
expr1 + expr2	Addition of decimal integer-valued argu?

? ? ments. ?

?expr1 - expr2 ? Subtraction of decimal integer-valued ?

? ? arguments. ?

??

? ? Returns the result of a decimal integer ?

? ? comparison if both arguments are inte? ?

? ? gers; otherwise, returns the result of a ?

? ? string comparison using the locale-spe? ?

? ? cific collation sequence. The result of ?

? ? each comparison is 1 if the specified ?

? ? relationship is true, or 0 if the rela? ?

? ? tionship is false. ?

?expr1 = expr2 ? Equal. ?

?expr1 > expr2 ? Greater than. ?

?expr1 >= expr2 ? Greater than or equal. ?

?expr1 < expr2 ? Less than. ?

?expr1 <= expr2 ? Less than or equal. ?

?expr1 != expr2 ? Not equal. ?

??

?expr1 & expr2 ? Returns the evaluation of expr1 if nei? ?

? ? ther expression evaluates to null or ?

? ? zero; otherwise, returns zero. ?

??

?expr1 | expr2 ? Returns the evaluation of expr1 if it is ?

? ? neither null nor zero; otherwise, re? ?

? ? turns the evaluation of expr2 if it is ?

? ? not null; otherwise, zero. ?

??

Matching Expression

The ':' matching operator shall compare the string resulting from the evaluation of expr1 with the regular expression pattern resulting from the evaluation of expr2. Regular expression syntax shall be that de? fined in the Base Definitions volume of POSIX.1?2017, Section 9.3, Ba?

sic Regular Expressions, except that all patterns are anchored to the beginning of the string (that is, only sequences starting at the first character of a string are matched by the regular expression) and, therefore, it is unspecified whether '^' is a special character in that context. Usually, the matching operator shall return a string repre? senting the number of characters matched ('0' on failure). Alterna? tively, if the pattern contains at least one regular expression subex? pression "[\(...\)]", the string matched by the back-reference expres? sion "\1" shall be returned. If the back-reference expression "\1" does not match, then the null string shall be returned.

Identification as Integer or String

An argument or the value of a subexpression that consists only of an optional unary minus followed by digits is a candidate for treatment as an integer if it is used as the left argument to the | operator or as either argument to any of the following operators: & = > >= < <= != + - * / %. Otherwise, the argument or subexpression value shall be treated as a string.

The use of string arguments length, substr, index, or match produces unspecified results.

EXIT STATUS

The following exit values shall be returned:

0 The expression evaluates to neither null nor zero.

1 The expression evaluates to null or zero.

2 Invalid expression.

>2 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

The expr utility has a rather difficult syntax:

* Many of the operators are also shell control operators or reserved words, so they have to be escaped on the command line.

* Each part of the expression is composed of separate arguments, so

liberal usage of <blank> characters is required. For example:

```
??????????????????????????????????????????
?  Invalid    ?       Valid       ?
??????????????????????????????????????????
?expr 1+2       ? expr 1 + 2          ?
?expr "1 + 2"   ? expr 1 + 2          ?
?expr 1 + (2 * 3) ? expr 1 + \( 2 \* 3 \) ?
??????????????????????????????????????????
```

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in expr. Newly written scripts should avoid expr in favor of the new fea? tures within the shell; see Section 2.5, Parameters and Variables and Section 2.6.4, Arithmetic Expansion.

After argument processing by the shell, expr is not required to be able to tell the difference between an operator and an operand except by the value. If "$a" is '=', the command:

```
expr "$a" = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to expr (and they all may be taken as the '=' operator). The following works reliably:

```
expr "X$a" = X=
```

Also note that this volume of POSIX.1?2017 permits implementations to extend utilities. The expr utility permits the integer arguments to be preceded with a unary minus. This means that an integer argument could look like an option. Therefore, the conforming application must employ the "--" construct of Guideline 10 of the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines to protect its operands if there is any chance the first operand might be a negative integer (or any string with a leading minus).

For testing string equality the test utility is preferred over expr, as it is usually implemented as a shell built-in. However, the functional? ity is not quite the same because the expr = and != operators check

whether strings collate equally, whereas test checks whether they are identical. Therefore, they can produce different results in locales where the collation sequence does not have a total ordering of all characters (see the Base Definitions volume of POSIX.1?2017, Section 7.3.2, LC_COLLATE).

EXAMPLES

The following command:

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

adds 1 to the variable a.

The following command, for "$a" equal to either /usr/abc/file or just file:

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

returns the last segment of a pathname (that is, file). Applications should avoid the character '/' used alone as an argument; expr may in? terpret it as the division operator.

The following command:

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

is a better representation of the previous example. The addition of the "//" characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that pathnames may con? tain characters contained in the IFS variable and should be quoted to avoid having "$a" expand into multiple arguments.

The following command:

```
expr "X$VAR" : '.*' - 1
```

returns the number of characters in VAR.

RATIONALE

In an early proposal, EREs were used in the matching expression syntax. This was changed to BREs to avoid breaking historical applications.

The use of a leading <circumflex> in the BRE is unspecified because many historical implementations have treated it as a special character, despite their system documentation. For example:

```
expr foo : ^foo     expr ^foo : ^foo
```

return 3 and 0, respectively, on those systems; their documentation

would imply the reverse. Thus, the anchoring condition is left unspeci?
fied to avoid breaking historical scripts relying on this undocumented
feature.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, Parameters and Variables, Section 2.6.4, Arithmetic Expan?
sion

The Base Definitions volume of POSIX.1?2017, Section 7.3.2, LC_COLLATE,
Chapter 8, Environment Variables, Section 9.3, Basic Regular Expres?
sions, Section 12.2, Utility Syntax Guidelines

COPYRIGHT