NAME

expr – evaluate arguments as a logical, arithmetic, or string expression

SYNOPSIS

/usr/ucb/expr *argument*. . .

AVAILABILITY

SUNWscpu

DESCRIPTION

expr evaluates expressions as specified by its arguments. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument, so terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note: **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, two's-complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by '\'. The list is in order of increasing precedence, with equal precedence operators grouped within **{ }** symbols.

expr \| *expr*
: Return the first *expr* if it is neither **NULL** nor **0**, otherwise returns the second *expr*.

expr **\&** *expr*
: Return the first *expr* if neither *expr* is **NULL** or **0**, otherwise returns **0**.

expr { **=**, \>, \>= , \<, \<=, **!=** } *expr*
: Return the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { **+**, – } *expr*
: Addition or subtraction of integer-valued arguments.

expr { *, **/**, **%** } *expr*
: Multiplication, division, or remainder of the integer-valued arguments.

string **:** *regular-expression*
match *string regular-expression*
: The two forms of the matching operator above are synonymous. The matching operators **:** and **match** compare the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **regexp**(5), except that all patterns are "anchored" (treated as if they begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the **\(. . . \)** pattern symbols can be used to return a portion of the first argument.

substr *string integer-1 integer-2*
Extract the substring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1* has a value greater than the length of *string*, **expr** returns a null string. If you try to extract more characters than there are in *string*, **expr** returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as **expr** tends to run forever in these cases.

index *string character-list*
Report the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

length *string*
Return the length (that is, the number of characters) of *string*.

(expr) Parentheses may be used for grouping.

EXAMPLES

1. **a=‘expr $a + 1‘**

   Adds 1 to the shell variable **a**.

2. **# ’For $a equal to either "/usr/abc/file" or just "file"’**
   expr $a : ’.*/\(.*\)’ \| $a

   Returns the last segment of a path name (that is, the filename part). Watch out for **/** alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. **# A better representation of example 2.**
   expr //$a : ’.*/\(.*\)’

   The addition of the **//** characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. **expr $VAR : ’.*’**

   Returns the number of characters in **$VAR**.

EXIT STATUS

expr returns the following exit codes:

0 if the expression is neither **NULL** nor **0**

1 if the expression *is* **NULL** or **0**

2 for invalid expressions.

SEE ALSO

sh(1), **test**(1), **regexp**(5)

DIAGNOSTICS

syntax error for operator/operand errors

non-numeric argument
if arithmetic is attempted on such a string

division by zero
if an attempt to divide by zero is made

BUGS

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an **=**, the command:

expr $a = '='

looks like:

expr = = =

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

expr X$a = X=

Note: the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

```
example% expr index expurgatorious length
syntax error
example%
```

generates the '**syntax error**' message as shown instead of the value **1** as you might expect.