NAME

expr — evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **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, 2'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*
returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr*
returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
returns 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.

expr : *expr*
The matching operator : compares the first argument with the second argument which must be a regular expression; regular expression syntax is the same as that of *ed*(1), except that all patterns are 'anchored' (i.e., 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.

ARCHAIC FORMS

The following operators are supported by the current version of **expr**, but have been made obsolete by the : operator. These should not be used in new applications.

substr *expra exprb exprc*
returns on standard output that portion of *expra* (possibily null) which is defined by the numerical offset (*exprb*, starting at **1**) and the numerical span (*exprc*). A large span value may be given to obtain the remainder of the string.
substr abcd 2 2 is equivalent to **abcd : '..\(..\)'**

length *expr*
returns the length in characters of the expression that follows.
length *expr* **is equivalent to** *expr* **: '.*'**

index *expra exprb*
searches the first expression for the first character that matches a character from the second expression. It returns the character position number if is succeds, or **0** if it fails.
index abcd d is equivalent to **abcd : d**

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 (i.e., file). 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.**

SEE ALSO

ed(1), sh(1).

EXIT CODE

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

DIAGNOSTICS

syntax error − for operator/operand errors
non-numeric argument − if arithmetic is attempted on such a string

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=