**NAME**

expr – evaluate arguments as an expression

**SYNOPSIS**

**expr** *arguments*

**AVAILABILITY**

SUNWcsu

**DESCRIPTION**

The **expr** utility will evaluate the expression and write the result to standard output. The character **0** will be written to indicate a zero value and nothing will be written to indicate a **NULL** string.

**OPERANDS**

*arguments* are taken as an expression. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped (see **sh**(1)). Strings containing blanks or other special characters should be quoted. The length of the expression is limited to 512 characters.

The operators and keywords are listed below. 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** or **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 (see **NOTES**). Normally, the matching operator returns the number of bytes matched (**0** on failure).

**(** *expr* **)** pattern symbols; can be used to return a portion of the first argument.

*integer* An argument consisting only of an (optional) unary minus followed by digits.

*string* A string argument that cannot be identified as an *integer* argument or as one of the expression operator symbols.

**EXAMPLES**

Add 1 to the shell variable **a**:

```
example$ a=`expr $a + 1`
```

The following example emulates **basename**(1) — it returns the last segment of the path name **$a**. For **$a** equal to either **/usr/abc/file** or just **file**, the example returns **file**. (Watch out for **/** alone as an argument: **expr** takes it as the division operator; see **NOTES** below.)

**example$ expr $a : ´.*/\(.*\)´ \| $a**

Here is a better version of the previous example. The addition of the **//** characters eliminates any ambiguity about the division operator and simplifies the whole expression.

**example$ expr //$a : ´.*/\(.*\)´**

Return the number of characters in **$VAR**:

**example$ expr $VAR : ´.*´**

## ENVIRONMENT

See **environ**(5) for descriptions of the following environment variables that affect the execution of **expr**: **LC_COLLATE**, **LC_CTYPE**, **LC_MESSAGES**, and **NLSPATH**.

## EXIT STATUS

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 either **NULL** or **0**

**2** for invalid expressions.

**>2** an error occurred.

## FILES

**/usr/lib/locale/*locale*/LC_COLLATE/CollTable**
collation table generated by **localedef**

**/usr/lib/locale/*locale*/LC_COLLATE/coll.so**
shared object containing string transformation library routines

## SEE ALSO

**basename**(1), **ed**(1), **sh**(1), **environ**(5), **regex**(5), **regexp**(5)

## DIAGNOSTICS

**syntax error** for operator/operand errors

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

## NOTES

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:

**example$ expr $a = ´=´**

looks like:

**example$ expr = = =**

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

**example$ expr X$a = X=**

### Regular Expressions

Internationalized Regular Expressions are used in the POSIX and "C" locales. In other locales, Internationalized Regular Expressions are used if the following two conditions are met:

- **/usr/lib/locale/***locale***/LC_COLLATE/CollTable** is present
- **/usr/lib/locale/***locale***/LC_COLLATE/coll.so** is not present;

otherwise, Simple Regular Expressions are used. Note that all patterns are “anchored” (that is, begin with ^) and, therefore, ^ is not a special character in that context.

Internationalized Regular Expressions are explained on **regex**(5).
Simple Regular Expressions are explained on **regexp**(5).