NAME

expr, test, [− evaluate expressions

SYNOPSIS

expr *expression*
test *expression*
[*expression* **]**

DESCRIPTION

Expr evaluates the expression and prints the result. **Test** evaluates the expression without printing the result. The ‘‘[’’ command is a synonym for **test**; when invoked under this name the last argument to **expr** must be a ‘‘]’’, which is deleted and not considered part of the expression.

Three data types may occur in the *expression*: string, integer, and boolean. The rules for conversion are as follows:

string−>integer	Done via **atoi**(3).
integer−>string	Convert to decimal representation.
string−>boolean	"" −> false, everything else to true.
boolean−>string	false −> "", true −> "true".
integer−>boolean	0 −> false, everything else to true.
boolean−>integer	false −> 0, true −> 1.

Any argument to **expr** which is not a legal operator is treated as a string operand of type **string**.

As a special case, if *expression* is omitted, the result is false.

We now list the operators. The syntax

integer **op** *integer* −> *boolean* **(3)**

means that **op** is a binary operator which takes operands of type *integer* and produces a result of type *boolean*. The ‘‘(3)’’ means that the priority of **op** is 3. Operands are automatically converted to the appropriate type. The type *any* is used for operator that take operands of any type.

any **−o** *any* −> *any* (1)
Returns the value of the left hand operand if the left hand operand would yield **true** if converted to type **boolean**, and the value of the right hand operand otherwise. The right hand operand is evaluated only if necessary. ‘‘|’’ is a synonym for ‘‘−o’’.

any **-a** *any* −> *any* (2)
Returns the value of the left hand operand if the left hand operand would yield **false** if converted to type **boolean**, and the value of the right hand operand otherwise. The right hand operand is evaluated only if necessary. ‘‘&’’ is a synonym for ‘‘−a’’.

! *boolean* −> *boolean* (3)
Returns true if the operand is false, and false if the operand is true.

string **=** *string* −> *boolean* (4)
True if the two strings are equal.

string **!=** *string* −> *boolean* (4)
True if the two strings are not equal.

integer **−eq** *integer* −> *boolean* (4)
True if the two operands are equal.

integer **−ne** *integer* −> *boolean* (4)
True if the two operands are not equal.

integer **−gt** *integer* −> *boolean* (4)
True if the first operand is greater than the second one.

integer **−lt** *integer* −> *boolean* (4)
: True if the first operand is less than the second one.

integer **−ge** *integer* −> *boolean* (4)
: True if the first operand is greater than or equal to the second one.

integer **−le** *integer* −> *boolean* (4)
: True if the first operand is less than or equal to the second one.

integer **+** *integer* −> *integer* (5)
: Add two integers.

integer **−** *integer* −> *integer* (5)
: Subtract two integers.

integer ***** *integer* −> *integer* (6)
: Multiply two integers. ‘‘*’’ is special to the shell, so you generally have to write this operator as ‘‘*’’.

integer **/** *integer* −> *integer* (6)
: Divide two integers.

integer **%** *integer* −> *integer* (6)
: Returns the remainder when the first operand is divided by the second one.

string **:** *string* −> *integer or string* (7)
: The second operand is interpreted as a regular expression (as in the System V **ed** program). This operator attempts to match part (or all) of the first operand with the regular expression. The match must start at the beginning of the first operand. If the regular expression contains \(\) pairs, then the result of this operator is the string which is matched by the regular expression between these pairs, or the null string if no match occurred. Otherwise, the result is the number of characters matched by the regular expression, or zero if no no match occurred.

−n *string* −> *integer* (8)
: Returns the number of characters in the string.

−z *string* −> *boolean* (8)
: Returns true if the string contains zero characters.

−t *integer* −> *boolean* (8)
: Returns true if the specified file descriptor is associated with a tty.

The remaining operators all deal with files. Except as noted, they return false if the specified file does not exist. The ones dealing with permission use the effective user and group ids of the shell.

−r *string* −> *boolean* (8)
: True if you have read permission on the file.

−w *string* −> *boolean* (8)
: True if you have write permission on the file.

−x *string* −> *boolean* (8)
: True if you have execute permission on the file.

−f *string* −> *boolean* (8)
: True if the file is a regular file.

−d *string* −> *boolean* (8)
: True if the file is a directory.

−c *string* −> *boolean* (8)
: True if the file is a character special file.

−b *string* −> *boolean* (8)
: True if the file is a block special file.

−p *string* −> *boolean* (8)
True if the file is a named pipe (i.e. a fifo).

−u *string* −> *boolean* (8)
True if the file is setuid.

−g *string* −> *boolean* (8)
True if the file is setgid.

−k *string* −> *boolean* (8)
True if the file has the sticky bit set.

−s *string* −> *integer or boolean* (8)
Returns the size of the file, or 0 if the file does not exist.

−h *string* −> *boolean* (8)
True if the file is a symlink. This is the only file test operator that does not follow symlinks, all others do. So ‘‘−d’’ and ‘‘−h’’ are both true on a symlink pointing to a directory. ‘‘−L’’ is a synonym for ‘‘−h’’.

EXIT CODE

0 if the result of *expression* would be **true** if the result were converted to **boolean**.
1 if the result of *expression* would be **false** if the result were converted to **boolean**.
2 if *expression* is syntactically incorrect.

EXAMPLES

filesize=‘expr −s file‘
Sets the shell variable *filesize* to the size of *file*.

if [−s file]; then command; fi
Execute *command* if *file* exists and is not empty.

x=‘expr "$x" : ’.\{4\}\(.\{0,3\}\)’‘
Sets *x* to the substring of *x* beginning after the fourth character of *x* and continuing for three characters or until the end of the string, whichever comes first.

x=‘expr X"$x" : X’.\{4\}\(.\{0,3\}\)’‘
This example is the same as the previous one, but it uses a leading ‘‘X’’ to make things work when the value of *x* looks like an operator.

BUGS

The relational operators of the System V **expr** command are not implemented.

Certain features of this version of **expr** are not present in System V, so care should be used when writing portable code.

COPYRIGHT

Kenneth Almquist.