expr

expr evaluates arguments as an expression.

syntax

```
expr EXPRESSION
expr OPTION
```

Options

--help	Display a help message and exit.
--version	Display version information and exit.

Expressions

expr prints the value of *EXPRESSION* to standard output. A blank line below separates increasing precedence groups.

EXPRESSION may be:

ARG1 | *ARG2*	*ARG1* if it is neither null nor 0, otherwise *ARG2*.
ARG1 **&** *ARG2*	*ARG1* if neither argument is null or 0, otherwise 0.
ARG1 < *ARG2*	*ARG1* is less than *ARG2*.
ARG1 <= *ARG2*	*ARG1* is less than or equal to *ARG2*.
ARG1 = *ARG2*	*ARG1* is equal to *ARG2*.
ARG1 **!=** *ARG2*	*ARG1* is unequal to *ARG2*.
ARG1 >= *ARG2*	*ARG1* is greater than or equal to *ARG2*.
ARG1 > *ARG2*	*ARG1* is greater than *ARG2*.
ARG1 **+** *ARG2*	arithmetic sum of *ARG1* and *ARG2*.
ARG1 **-** *ARG2*	arithmetic difference of *ARG1* and *ARG2*.
ARG1 ***** *ARG2*	arithmetic product of *ARG1* and *ARG2*.
ARG1 / *ARG2*	arithmetic quotient of *ARG1* divided by *ARG2*.
ARG1 **%** *ARG2*	arithmetic remainder of *ARG1* divided by *ARG2*.
STRING **:** *REGEXP*	anchored pattern match of regular expression *REGEXP* in *STRING*.
match *STRING REGEXP*	same as *STRING* **:** *REGEXP*.
substr *STRING POS LENGTH*	substring of *STRING*, *POS* counted from 1.
index *STRING CHARS*	index in *STRING* where any *CHARS* is found, or **0**.
length *STRING*	length of *STRING*.

+ *TOKEN*	interpret *TOKEN* as a string, even if it is a keyword like '**match**' or an operator like '/'.
(*EXPRESSION* **)**	value of *EXPRESSION*.

Notes About Usage

- Be aware that many operators need to be escaped or quoted to be interpreted correctly by shells.
- Comparisons are arithmetic if both ARGs are numbers, otherwise the comparisons are lexicographical.
- Pattern matches return the string matched between \**(** and \**)** or null; if \**(** and \**)** are not used, they return the number of characters matched or 0.
- Exit status is **0** if EXPRESSION is neither null nor 0, **1** if EXPRESSION is null or 0, **2** if EXPRESSION is syntactically invalid, and 3 if an error occurred.

expr examples

```
expr text : '.*'
```

Performs a regular expression match. The regular expression after the colon is matched to the text before the colon. The returned output is the number of characters that matched. Here, the regular expression **'.*'** represents "any number of any character", therefore the result is:

```
4
expr text : tex
```

Returns the number of characters from the regular expression after the colon which appear in the text before the colon. Here, the regular expression **'tex'** represents "exactly the consecutive characters **t**, **e**, and **x**", so the output would be:

```
3
expr text : '\(.*\)'
```

Here, the regular expression **'\\(.*\\)'** represents "The actual text (whatever appears in between the parentheses, which are escaped with backslashes) which matches the pattern **.***, which itself represents any number of any character." Matched against the text **text**, this returns the string exactly:

```
text
expr 5 = 5
```

Returns **1** (true) if the expressions are equivalent, or **0** (false) if they are not. Here, the values **5** and **5** are equal, and therefore *equivalent*, so the output will be:

```
1
expr '5' = '5'
```

Here, two strings are being compared for equivalence. If the strings match exactly, character-for-character, the result will be **1** (true). Otherwise, the result will be **0**. Here, the result is:

```
1
expr 5 \> 10
```

Here, the result is **1** (true) if **5** is less than **10**, otherwise the result is **0**. The "less than" symbol ("<") is preceded by a backslash ("\") to protect it from the shell, which would otherwise interpret it as a redirection operator. In this example, **5** is *not* greater than **10**, so the output is:

```
0
expr 5 \!= 5
```

Just as the **=** operator tests for equivalence, the **!=** operator tests for non-equivalence. If the two values being tested are not equivalent, the result is true (**1**), otherwise the result is false (**0**). **5** is equivalent to **5**, so the result is false:

```
0
expr 5 \!= "5"
```

"Equivalence" is not the same as "equality". **5** is a number, and **"5"** is a string, so technically they are not "equal," but **expr** considers them *equivalent* because it reads the string for its contents, sees that it is a number, and uses the value of that number in the comparison. So the value **5** is equivalent to a string containing the number **"5"**. Therefore the answer here is false; they are *not non*-equivalent:

```
0
```

This next example will show, in a series of commands, how **expr** can be used to increment the value of a variable.

If we define a variable named **count**, setting it to zero:

```
count=0
```

...we can output the value of that variable with the **echo** command:

```
echo $count
0
```

...now we can increment it by setting it to the value of an **expr** evaluation, which returns the value of the variable, plus one:

```
count=`expr $count + 1`
```

...and we can check the updated value with another **echo**:

```
echo $count
1
```