

## 6.5 Shell Arithmetic

The shell allows arithmetic expressions to be evaluated, as one of the shell expansions or by using the `((` compound command, the `let` builtin, or the `-i` option to the `declare` builtin.

Evaluation is done in fixed-width integers with no check for overflow, though division by 0 is trapped and flagged as an error. The operators and their precedence, associativity, and values are the same as in the C language. The following list of operators is grouped into levels of equal-precedence operators. The levels are listed in order of decreasing precedence.

***id++ id--***

variable post-increment and post-decrement

***++id --id***

variable pre-increment and pre-decrement

***- +***

unary minus and plus

***! ~***

logical and bitwise negation

***\*\****

exponentiation

***\* / %***

multiplication, division, remainder

***+ -***

addition, subtraction

***<< >>***

left and right bitwise shifts

***<= >= < >***

comparison

***== !=***

equality and inequality

***&***

bitwise AND

***^***

bitwise exclusive OR

|  
bitwise OR

&&  
logical AND

||  
logical OR

**expr ? expr : expr**  
conditional operator

= \*= /= %= += -= <<= >>= &= ^= |=  
assignment

**expr1 , expr2**  
comma

Shell variables are allowed as operands; parameter expansion is performed before the expression is evaluated. Within an expression, shell variables may also be referenced by name without using the parameter expansion syntax. A shell variable that is null or unset evaluates to 0 when referenced by name without using the parameter expansion syntax. The value of a variable is evaluated as an arithmetic expression when it is referenced, or when a variable which has been given the integer attribute using 'declare -i' is assigned a value. A null value evaluates to 0. A shell variable need not have its integer attribute turned on to be used in an expression.

Integer constants follow the C language definition, without suffixes or character constants. Constants with a leading 0 are interpreted as octal numbers. A leading '0x' or '0X' denotes hexadecimal. Otherwise, numbers take the form [*base#*]*n*, where the optional *base* is a decimal number between 2 and 64 representing the arithmetic base, and *n* is a number in that base. If *base#* is omitted, then base 10 is used. When specifying *n*, if a non-digit is required, the digits greater than 9 are represented by the lowercase letters, the uppercase letters, '@', and '\_', in that order. If *base* is less than or equal to 36, lowercase and uppercase letters may be used interchangeably to represent numbers between 10 and 35.

Operators are evaluated in order of precedence. Sub-expressions in parentheses are evaluated first and may override the precedence rules above.