The let builtin command

Synopsis

```
let arg [arg ...]
```

Description

The `let` builtin command evaluates each supplied word from left to right as an arithmetic expression and returns an exit code according to the truth value of the rightmost expression.

- 0 (TRUE) when `arg` evaluated to not 0 (arithmetic "true")
- 1 (FALSE) when `arg` evaluated to 0 (arithmetic "false")

For this return code mapping, please see this section. They work in the same way as `((` .

Examples

`let` is very similar to ((- the only difference being `let` is a builtin (simple command), and `((` is a compound command. The arguments to `let` are therefore subject to all the same expansions and substitutions as any other simple command - requiring proper quoting and escaping - whereas the contents of `((` aren't subject to word-splitting or pathname expansion (almost never desirable for arithmetic). For this reason, **the arithmetic compound command should generally be preferred over `let`** .

```
$ let 'b = a' "(a += 3) + $((a = 1)), b++"
$ echo "$a - $b - $?"
4 - 2 - 0
```

Is equivalent to the arithmetic evaluation compound command:

```
$ (( b = a, (a += 3) + $((a = 1)), b++ ))
$ echo "$a - $b - $?"
4 - 2 - 0
```

Remember that inside arithmetic evaluation contexts, all other expansions are processed as usual (from left-to-right), and the resulting text is evaluated as an arithmetic expression. Arithmetic already has a way to control precedence using parentheses, so it's very rare to need to nest arithmetic expansions within one another. It's used above only to illustrate how this precedence works.

Unlike `((`, being a simple command `let` has its own environment. In Bash, built-ins that can set variables process any arithmetic under their own environment, which makes the variable effectively "local" to the builtin unless the variable is also set or modified by the builtin. This differs in other shells, such as ksh93, where environment assignments to regular builtins are always local even if the variable is modified by the builtin.

```
 ~ $ ( y=1+1 let x=y; declare -p x y )
declare -- x="2"
bash: declare: y: not found

 ~ $ ( y=1+1 let x=y++; declare -p x y )
declare -- x="2"
declare -- y="3"
```

This can be useful in certain situations where a temporary variable is needed.

Portability considerations

- the `let` command is not specified by POSIX®. The portable alternative is:

  ```
  [ "$(( <EXPRESSION> ))" -ne 0 ]
  ```

  . To make portable scripts simpler and cleaner, `let` can be defined as:

  ```
  # POSIX
  let() {
      IFS=, command eval test '$(($*))' -ne 0
  }
  ```

  Aside from differences in supported arithmetic features, this should be identical to the Bash/Ksh `let`.
- It seems to be a common misunderstanding that `let` has some legacy purpose. Both `let` and ((...)) were ksh88 features and almost identical in terms of portability as everything that inherited one also tended to get the other. Don't choose `let` over `((` expecting it to work in more places.
- expr(1) (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/expr.html#tag_20_42) is a command one is likely to come across sooner or later. While it is more "standard" than `let`, the above should always be preferred. Both arithmetic expansions and the `[` test operator are specified by POSIX® and satisfy almost all of expr's use-cases. Unlike `let`, `expr` cannot assign directly to bash variables but instead returns a result on stdout. `expr` takes each operator it recognizes as a separate word and then concatenates them into a single expression that's evaluated

according to it's own rules (which differ from shell arithmetic). `let` parses each word it recieves on its own and evaluates it as an expression without generating any output other than a return code.
- For unknown reasons, `let` is one of the Bash commands with special parsing for arguments formatted like compound array assignments. See: eval for details. There are no known practical uses for this. Parentheses are treated as grouping operators and compound assignment is not possible by arithmetic expressions.

See also

- Internal: arithmetic expansion
- Internal: arithmetic expressions
- Internal: arithmetic evaluation compound command

Discussion

joshua toyota, 2011/01/13 09:33 ()

I believe the internal links to "arithmetic expansion" should be directed here:

http://wiki.bash-hackers.org/syntax/expansion/arith (http://wiki.bash-hackers.org/syntax/expansion/arith)

Instead of:

http://wiki.bash-hackers.org/syntax/arith_expr (http://wiki.bash-hackers.org/syntax/arith_expr)

Just an fyi...

Jan Schampera, 2011/01/13 17:40 ()

Of course, my bad...

Thanks