# Встроенная команда let

## Краткое описание

```
пусть arg [arg ...]
```

## Описание

`let` Встроенная команда оценивает каждое введенное слово слева направо как арифметическое выражение и возвращает код выхода в соответствии со значением истинности самого правого выражения.

- 0 (ИСТИНА) при `arg` вычислении не 0 (арифметическое "истина")
- 1 (FALSE) при `arg` вычислении до 0 (арифметическое "false")

Для этого сопоставления кода возврата, пожалуйста, смотрите Этот раздел . Они работают так же, как `((` .

## Примеры

`let` очень похожа на (( - с той лишь разницей `let` , что это встроенная (простая команда) и `((` является составной командой. Таким образом, аргументы to `let` подлежат всем тем же расширениям и заменам, что и любая другая простая команда, требующая правильных кавычек и экранирования, тогда как содержимое `((` не подлежит разделению на слова или расширению имени пути (что почти никогда не желательно для арифметики). По этой причине **арифметическая составная команда обычно должна быть предпочтительнее `let` .**

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

Эквивалентна составной команде арифметической оценки:

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

> Помните, что в контекстах арифметической оценки все остальные расширения обрабатываются как обычно (слева направо), а результирующий текст вычисляется как арифметическое выражение. В арифметике уже есть способ управлять приоритетом с помощью круглых скобок, поэтому очень редко возникает необходимость вкладывать арифметические расширения друг в друга. Она используется выше только для иллюстрации того, как работает этот приоритет.

В отличие `((` от простой команды `let` , у нее есть своя среда. В Bash встроенные модули, которые могут устанавливать переменные, обрабатывают любую арифметику в своей собственной среде, что делает переменную фактически "локальной" для встроенной, если только переменная также не установлена или не изменена встроенной. Это отличается в других оболочках, таких как ksh93, где присвоения среды обычным встроенным элементам всегда являются локальными, даже если переменная изменена встроенным компонентом.

```
 ~ $ ( y=1+1 пусть x=y; объявить -p x y )
объявить -- x="2"
bash: объявить: y: не найден

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

Это может быть полезно в определенных ситуациях, когда требуется временная переменная.

# Соображения о переносимости

- `let` команда не указана в POSIX®. Переносимая альтернатива:

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

  . Чтобы сделать переносимые скрипты проще и чище, `let` можно определить как:

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

  Помимо различий в поддерживаемых арифметических функциях, это должно быть идентично Bash / Ksh `let` .
- Похоже, это распространенное недоразумение, `let` имеющее какую-то устаревшую цель. Оба `let` и ((...)) были функциями ksh88 и почти идентичны с точки зрения переносимости, поскольку все, что унаследовало одно, также имело тенденцию получать другое. Не `let ((` ожидайте, что она будет работать в большем количестве мест.
- expr(1) (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/expr.html#tag_20_42) -

это команда, с которой, вероятно, рано или поздно столкнетесь. Хотя она более "стандартная" `let` , чем приведенная выше, всегда следует отдавать предпочтение. Оба арифметических расширения и оператор `[` тестирования задаются POSIX® и удовлетворяют почти всем вариантам использования expr. В отличие `let` от , `expr` не может присваивать переменные bash напрямую, а вместо этого возвращает результат в стандартном выводе. `expr` принимает каждый оператор, который он распознает, как отдельное слово, а затем объединяет их в одно выражение, которое вычисляется в соответствии с его собственными правилами (которые отличаются от арифметики оболочки). `let` анализирует каждое полученное слово самостоятельно и оценивает его как выражение, не генерируя никаких выходных данных, кроме кода возврата.

- По неизвестным причинам `let` является одной из команд Bash со специальным синтаксическим анализом для аргументов, отформатированных как назначения составного массива. Подробности см. в разделе: eval . Для этого нет известных практических применений. Круглые скобки рассматриваются как операторы группировки, а сложное присваивание невозможно с помощью арифметических выражений.

# Смотрите также

- Внутренняя: арифметическое расширение
- Внутренние: арифметические выражения
- Внутренняя: составная команда арифметической оценки

# Обсуждение

Джошуа Тойота, 2011/01/13 09:33 ()

Я считаю, что внутренние ссылки на "арифметическое расширение" должны быть направлены сюда:

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

Вместо:

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

Просто к вашему сведению...

Ян Шампера, 2011/01/13 17:40 ()

Конечно, я виноват...

Спасибо