Арифметические выражения

Арифметические выражения используются в нескольких ситуациях:

- команда вычисления арифметики
- арифметическое расширение
- расширение параметра подстроки
- встроенная команда "let"
- C-стиль для цикла
- индексация массива
- условные выражения
- Операторы присваивания и аргументы команд объявления переменных с атрибутом integer .

Эти выражения вычисляются в соответствии с некоторыми правилами, описанными ниже. Операторы и правила арифметических выражений в основном заимствованы из языка программирования C.

В этой статье описывается теория используемого синтаксиса и поведения. Чтобы получить практические примеры без подробных объяснений, см. Эту страницу в вики Грега (http://mywiki.wooledge.org/BashGuide/CompoundCommands#Arithmetic_Evaluation).

Константы

Математические константы - это просто фиксированные значения, которые вы пишете: `1` , `3567` , или `4326` . Bash интерпретирует некоторые обозначения специально:

- `0…` (начальный ноль) интерпретируется как **восьмеричное** значение
- `0x…` интерпретируется как **шестнадцатеричное** значение
- `0X…` также интерпретируется как **шестнадцатеричное**
- `<BASE>#…` интерпретируется как число в соответствии с **указанной базой** `<BASE>` , например, `2#00111011` (см. Ниже)

Если у вас есть константа, установленная в переменной, например,

```
x=03254
```

это интерпретируется как восьмеричное значение. Если вы хотите, чтобы оно интерпретировалось как десятичное значение, вам нужно развернуть параметр и указать основание 10:

```
# это интерпретируется как десятичное число:
echo $(( 10 # $ x ))

# это интерпретируется как восьмеричное число:
echo $(( x ))

# это недопустимая цифра для основания 10 ("x")...:
echo $(( 10 #x ))
```

Разные базы

Для константы основание может быть задано с помощью формы

```
<БАЗА>#<ЦИФРЫ ...>
```

Независимо от указанной базы, арифметические выражения, если они когда-либо будут отображаться, будут **отображаться в десятичной** системе счисления!

Когда основание не указано, предполагается основание 10 (десятичное), за исключением случаев, когда присутствуют префиксы, упомянутые выше (октальы, шестнадцатеричные числа). Указанная база может варьироваться от 2 до 64. Для представления цифр в указанной базе, превышающей 10, необходимы символы, отличные от 0 до 9 (в этом порядке, low ⇒ high):

- `0 … 9`
- `a … z`
- `A … Z`
- `@`
- `_`

Давайте быстро придумаем новую систему счисления с основанием 43, чтобы показать, что я имею в виду:

```
$ echo $((43 #1))
1

$ echo $((43 #a))
10

$echo $((43 #A))
36

$ echo $((43 # G))
42

$ echo $((43 # H))
bash: 43 # H: значение слишком велико для base (маркер ошибки "43 #
H")
```

Если вы понятия не имеете, что такое база и почему могут быть другие базы, а также что такое числа и как они построены, тогда вам не нужны разные базы.

Если вы хотите преобразовать между обычными основаниями (восьмеричными, десятичными, шестнадцатеричными), используйте команду printf и ее строки формата.

Переменные оболочки

Переменные оболочки, конечно, могут использоваться в качестве операндов, даже если атрибут integer не включен (by `declare -i <NAME>`). Если переменная пуста (null) или не задана, ее ссылка оценивается в 0. Если переменная не содержит значения, которое выглядит как допустимое выражение (числа или операции), выражение повторно используется для ссылки, например, на именованные параметры, например:

```
тест=строка
строка=3

echo $((тест))
# выведет "3"!
```

Конечно, в конце, когда он, наконец, вычисляет что-то, что **не** является допустимым арифметическим выражением (новые строки, обычный текст, ...), вы получите сообщение об ошибке.

Когда ссылаются на переменные, обозначение `1 + $X` эквивалентно обозначению `1 + X`, оба разрешены.

Когда на переменные ссылаются как `$X`, применяются правила расширения параметров, которые выполняются **до** вычисления выражения. Таким образом, конструкция like `${MYSTRING:4:3}` допустима внутри арифметического выражения.

Правда

В отличие от кодов выхода и возврата команд, арифметические выражения принимают логическое значение "true", если они не равны 0. Когда они равны 0, они оцениваются как "false". Составная команда арифметического вычисления изменяет "истинность" арифметического выражения на противоположную, чтобы оно соответствовало "истинности" кодов выхода команды:

- если арифметическое выражение выводит значение, отличное от 0 (арифметическое true), оно возвращает 0 (оболочка true)
- если арифметическое выражение принимает значение 0 (арифметическое false), оно возвращает 1 (оболочка false)

Это означает, что следующее `if` -предложение выполнит `else` -thread:

```
если ((0)); тогда
echo "true"
else
эхо "ложное"
fi
```

Операторы

Задание

Оператор	Описание
`<ID> = <EXPR>`	обычное назначение
`<ID> *= <EXPR>`	эквивалентно `<ID> = <ID> * <EXPR>`, см. **Операторы вычисления**
`<ID> /= <EXPR>`	эквивалентно `<ID> = <ID> / <EXPR>`, см. **Операторы вычисления**
`<ID> %= <EXPR>`	эквивалентно `<ID> = <ID> % <EXPR>`, см. **Операторы вычисления**
`<ID> += <EXPR>`	эквивалентно `<ID> = <ID> + <EXPR>`, см. **Операторы вычисления**
`<ID> -= <EXPR>`	эквивалентно `<ID> = <ID> - <EXPR>`, см. **Операторы вычисления**
`<ID> <<= <NUMBER>`	эквивалентно `<ID> = <ID> << <NUMBER>`, см. **Битовые операции**
`<ID> >>= <NUMBER>`	эквивалентно `<ID> = <ID> >> <NUMBER>`, см. **Битовые операции**
`<ID> &= <EXPR>`	эквивалентно `<ID> = <ID> & <EXPR>`, см. **Битовые операции**
`<ID> ^= <EXPR>`	эквивалентно `<ID> = <ID> ^ <EXPR>`, см. **Битовые операции**
`<ID> |= <EXPR>`	эквивалентно `<ID> = <ID> | <EXPR>`, см. **Битовые операции**

Вычисления

Оператор	Описание
`*`	умножение
`/`	разделение
`%`	остаток (по модулю)
`+`	сложение
`-`	вычитание

Оператор	Описание
**	возведение в степень

Сравнения

Оператор	Описание
<	сравнение: менее
>	сравнение: больше, чем
<=	сравнение: меньше или равно
>=	сравнение: больше или равно
==	равенство
!=	неравенство

Битовые операции

Оператор	Описание
~	побитовое отрицание
<<	побитовый сдвиг (слева)
>>	побитовый сдвиг (вправо)
&	побитовые И
^	побитовое исключающее ИЛИ (XOR)
|	побитовое ИЛИ

Логические

Оператор	Описание
!	логическое отрицание
&&	логические И
||	логическое ИЛИ

Разное

Оператор	Описание
`id++`	**последующее увеличение** переменной `id` (не требуется POSIX®)
`id--`	**последующее уменьшение** переменной `id` (не требуется POSIX®)
`++id`	**предварительное увеличение** переменной `id` (не требуется POSIX®)
`--id`	**предварительное уменьшение** переменной `id` (не требуется POSIX®)
`+`	унарный плюс
`-`	унарный минус
`<EXPR> ? <EXPR> : <EXPR>`	условный (троичный) оператор <условие> ? <результат, если истина>: <результат, если ложь>
`<EXPR> , <EXPR>`	список выражений
`( <EXPR> )`	Подвыражение (для принудительного приоритета)

Приоритет

Приоритет оператора следующий (самый высокий → самый низкий):

- Постфикс (`id++` , `id--`)
- Префикс (`++id` , `--id`)
- Унарный минус и плюс (`-` , `+`)
- Логическое и побитовое отрицание (`!` , `~`)
- Возведение в степень (`**`)
- Умножение, деление, остаток (`*` , `/` , `%`)
- Сложение, вычитание (`+` , `-`)
- Побитовые сдвиги (`<<` , `>>`)
- Сравнение (`<` , `>` , `<=` , `>=`)
- (В-) равенство (`==` , `!=`)
- Побитовые И (`&`)
- Побитовое XOR (`^`)
- Побитовое ИЛИ (`|`)
- Логические И (`&&`)
- Логическое ИЛИ (`||`)
- Троичный оператор (`<EXPR> ? <EXPR> : <EXPR>`)
- Задания (`=` , `*=` , `/=` , `%=` , `+=` , `-=` , `<<=` , `>>=` , `&=` , `^=` , `|=`)
- Оператор списка выражений (`<EXPR> , <EXPR>`)

Приоритет может быть скорректирован с помощью подвыражений формы `( <EXPR> )` в любое время. Эти подвыражения всегда вычисляются первыми.

Арифметические выражения и коды возврата

Общая языковая конструкция Bash основана на кодах выхода или кодах возврата команд или функций, которые должны быть выполнены. `if` операторы, `while` циклы и т. Д., Все они принимают коды возврата команд в качестве условий.

Теперь проблема заключается в: Коды возврата (0 означает "ИСТИНА" или "УСПЕХ", а не 0 означает "ЛОЖЬ" или "НЕУДАЧА") не соответствуют значению результата арифметического выражения (0 означает "ЛОЖЬ", а не 0 означает "ИСТИНА").

Вот почему все команды и ключевые слова, которые выполняют арифметические операции, пытаются **перевести** арифметическое значение в эквивалентный код возврата. Это просто означает:

- если арифметическая операция оценивается в 0 ("FALSE"), код возврата не равен 0 ("СБОЙ")
- если арифметическая операция оценивается в 1 ("ИСТИНА"), код возврата равен 0 ("УСПЕХ")

Таким образом, вы можете легко использовать арифметические выражения (вместе с командами или ключевыми словами, которые ими управляют) в качестве условий для `if` , `while` и все остальные, в том числе `set -e` для автоматического выполнения при ошибке:

```
MY_TEST_FLAG=0

, если ((MY_TEST_FLAG)); тогда
   echo "MY_TEST_FLAG включен"
, иначе
   echo "MY_TEST_FLAG ВЫКЛЮЧЕН"
fi
```

Будьте осторожны, это `set -e` может изменить поведение скриптов во время выполнения. Например,

Эта неэквивалентность поведения кода заслуживает некоторого внимания. Рассмотрим, что произойдет, если v окажется равным нулю в приведенном ниже выражении:

```
((v += 0))
echo  $?
```

1

("СБОЙ")

```
v = $ ((v + 0))
 echo  $?
```

0

("УСПЕХ")

Как было отмечено, поведение кода возврата не эквивалентно арифметическому поведению.

Обходным путем является использование операции со списком, которая возвращает True , или использование второго стиля присваивания.

```
((v += 0)) || :
echo  $?
```

0

("УСПЕХ")

Это изменение в поведении кода было обнаружено после запуска скрипта под управлением set -e.

Арифметические выражения в Bash

- Цикл for в стиле C
- Арифметическое расширение
- Арифметическая вычислительная составная команда
- Встроенная команда "let"

Обсуждение

sbin_bash, 2011/11/27 10:34 ()

Теперь проблема заключается в: Коды возврата (0 означает "ИСТИНА" или "УСПЕХ", а не 0 означает "ЛОЖЬ" или "НЕУДАЧА") не соответствуют значению результата арифметического выражения (0 означает "ИСТИНА", а не 0 означает "ЛОЖЬ").

=⇒ арифметическое выражение (0 означает "ЛОЖЬ", а не 0 означает "ИСТИНА")

Techlive Чжэн, 2012/11/02 18:01 ()

@sbin_bash, исправлено.

Джоан, 2013/04/19 15:47 ()

Ссылка в начале должна указывать на: http://mywiki .wooledge.org/BashGuide/CompoundCommands#Arithmetic_Evaluation (http://mywiki.wooledge.org/BashGuide/CompoundCommands#Arithmetic_Evaluation)

Ян Шампера, 2013/04/19 18:50 ()

Сделано, спасибо!