12.8. Команды выполнения математических операций

factor

Разложение целого числа на простые множители.

```
bash$ factor 27417
27417: 3 13 19 37
```

bc

Bash не в состоянии выполнять действия над числами с плавающей запятой и не содержит многих важных математических функций. К счастью существует **bc**.

Универсальная, выполняющая вычисления с произвольной точностью, утилита **bc** обладает некоторыми возможностями, характерными для языков программирования.

Синтаксис **bc** немного напоминает язык C.

Поскольку это утилита UNIX, то она может достаточно широко использоваться в сценариях на языке командной оболочки, в том числе и в конвейерной обработке данных.

Ниже приводится простой шаблон работы с утилитой **bc** в сценарии. Здесь используется прием подстановки команд.

```
variable=$(echo "OPTIONS; OPERATIONS" | bc)
```

Пример 12-32. Ежемесячные выплаты по займу

```
#!/bin/bash
# monthlypmt.sh: Расчет ежемесячных выплат по займу.
```

```
#  Это измененный вариант пакета "mcalc" (mortgage calculator),
#+ написанного Jeff Schmidt и Mendel Cooper (ваш покорный слуга).
#   http://www.ibiblio.org/pub/Linux/apps/financial/mcalc-1.6.tar.gz  [15k]

echo
echo "Введите сумму займа, процентную ставку и срок займа,"
echo "для расчета суммы ежемесячных выплат."

bottom=1.0

echo
echo -n "Сумма займа (без запятых -- с точностью до доллара) "
read principal
echo -n "Процентная ставка (процент) "  # Если 12%, то нужно вводить "12", а не
".12".
read interest_r
echo -n "Срок займа (месяцев) "
read term


 interest_r=$(echo "scale=9; $interest_r/100.0" | bc) # Здесь "scale" -- точность
вычислений.


 interest_rate=$(echo "scale=9; $interest_r/12 + 1.0" | bc)


 top=$(echo "scale=9; $principal*$interest_rate^$term" | bc)

 echo; echo "Прошу подождать. Вычисления потребуют некоторого времени."

 let "months = $term - 1"
# ============================================================================
 for ((x=$months; x > 0; x--))
 do
   bot=$(echo "scale=9; $interest_rate^$x" | bc)
   bottom=$(echo "scale=9; $bottom+$bot" | bc)
#  bottom = $(($bottom + $bot"))
 done
# ----------------------------------------------------------------------------
#  Rick Boivie предложил более эффективную реализацию
#+ цикла вычислений, который дает выигрыш по времени на 2/3.

# for ((x=1; x <= $months; x++))
# do
```

```
#   bottom=$(echo "scale=9; $bottom * $interest_rate + 1" | bc)
# done


#  А затем нашел еще более эффективную альтернативу,
#+ которая выполняется в 20 раз быстрее !!!

# bottom=`{
#     echo "scale=9; bottom=$bottom; interest_rate=$interest_rate"
#     for ((x=1; x <= $months; x++))
#     do
#          echo 'bottom = bottom * interest_rate + 1'
#     done
#     echo 'bottom'
#     } | bc`       # Внедрить цикл 'for' в конструкцию подстановки команд.

# ============================================================================

 # let "payment = $top/$bottom"
 payment=$(echo "scale=2; $top/$bottom" | bc)
 # Два знака после запятой, чтобы показать доллары и центы.

 echo
 echo "ежемесячные выплаты = \$$payment"  # Вывести знак "доллара" перед числом.
 echo


 exit 0

 # Упражнения:
 #   1) Добавьте возможность ввода суммы с точностью до цента.
 #   2) Добавьте возможность ввода процентной ставки как в виде процентов, так и в
виде десятичного числа -- доли целого.
 #   3) Если вы действительно честолюбивы,
 #      добавьте в сценарий вывод полной таблицы помесячных выплат.
```

Пример 12-33. Перевод чисел из одной системы счисления в другую

```
:
##########################################################################
# Shellscript:  base.sh - вывод чисел в разных системах счисления (Bourne Shell)
# Author     :  Heiner Steven (heiner.steven@odn.de)
# Date       :  07-03-95
# Category   :  Desktop
# $Id: base.sh,v 1.2 2000/02/06 19:55:35 heiner Exp $
##########################################################################
# Description
#
```

```
# Changes
# 21-03-95 stv  исправлена ошибка, возникающая при вводе числа 0xb (0.2)
##############################################################################

# ==> Используется в данном документе с разрешения автора.
# ==> Комментарии добавлены автором документа.

NOARGS=65
PN=`basename "$0"`                              # Имя программы
VER=`echo '$Revision: 1.2 $' | cut -d' ' -f2`  # ==> VER=1.2

Usage () {
    echo "$PN - вывод чисел в различных системах счисления, $VER (stv '95)
Порядок использования: $PN [number ...]

Если число не задано, то производится ввод со stdin.
Число может быть:
    двоичное            должно начинаться с комбинации символов 0b (например
0b1100)
    восьмеричное        должно начинаться с 0  (например 014)
    шестнадцатиричное   должно начинаться с комбинации символов 0x (например 0xc)
    десятичное          в любом другом случае (например 12)" >&2
    exit $NOARGS
}   # ==> Функция вывода сообщения о порядке использования.

Msg () {
    for i   # ==> [список] параметров опущен.
    do echo "$PN: $i" >&2
    done
}

Fatal () { Msg "$@"; exit 66; }

PrintBases () {
    # Определение системы счисления
    for i      # ==> [список] параметров опущен...
    do         # ==> поэтому работает с аргументами командной строки.
        case "$i" in
            0b*)                ibase=2;;       # двоичная
            0x*|[a-f]*|[A-F]*)  ibase=16;;      # шестнадцатиричная
            0*)                 ibase=8;;       # восьмеричная
            [1-9]*)             ibase=10;;      # десятичная
            *)
                Msg "Ошибка в числе $i - число проигнорировано"
                continue;;
        esac
```

```
        # Удалить префикс и преобразовать шестнадцатиричные цифры в верхний регистр
(этого требует bc)
        number=`echo "$i" | sed -e 's:^0[bBxX]::' | tr '[a-f]' '[A-F]'`
        # ==> вместо "/", здесь используется символ ":" как разделитель для sed.

        # Преобразование в десятичную систему счисления
        dec=`echo "ibase=$ibase; $number" | bc`  # ==> 'bc' используется как
калькулятор.
        case "$dec" in
            [0-9]*)     ;;       # все в порядке
            *)          continue;; # ошибка: игнорировать
        esac

        # Напечатать все преобразования в одну строку.
        # ==> 'вложенный документ' -- список команд для 'bc'.
        echo `bc <<!
            obase=16; "hex="; $dec
            obase=10; "dec="; $dec
            obase=8;  "oct="; $dec
            obase=2;  "bin="; $dec
!
    ` | sed -e 's: :    :g'

    done
}

while [ $# -gt 0 ]
do
    case "$1" in
        --)     shift; break;;
        -h)     Usage;;          # ==> Вывод справочного сообщения.
        -*)     Usage;;
        *)      break;;          # первое число
    esac   # ==> Хорошо бы расширить анализ вводимых символов.
    shift
done

if [ $# -gt 0 ]
then
    PrintBases "$@"
else                                  # чтение со stdin
    while read line
    do
        PrintBases $line
    done
fi
```

Один из вариантов вызова **bc** -- использование вложенного документа, внедряемого в блок с подстановкой команд. Это особенно актуально, когда сценарий должен передать **bc** значительный по объему список команд и аргументов.

```
variable=`bc << LIMIT_STRING
options
statements
operations
LIMIT_STRING
`
```

...или...

```
variable=$(bc << LIMIT_STRING
options
statements
operations
LIMIT_STRING
)
```

Пример 12-34. Пример взаимодействия bc со "встроенным документом"

```
#!/bin/bash
# Комбинирование 'bc' с
# 'вложенным документом'.


var1=`bc << EOF
18.33 * 19.78
EOF
`
echo $var1       # 362.56


#  запись $( ... ) тоже работает.
v1=23.53
v2=17.881
v3=83.501
v4=171.63

var2=$(bc << EOF
scale = 4
a = ( $v1 + $v2 )
b = ( $v3 * $v4 )
a * b + 15.35
EOF
)
```

```
echo $var2       # 593487.8452


var3=$(bc -l << EOF
scale = 9
s ( 1.7 )
EOF
)
# Возвращается значение синуса от 1.7 радиана.
# Ключом "-l" вызывается математическая библиотека 'bc'.
echo $var3       # .991664810


# Попробуем функции...
hyp=             # Объявление глобальной переменной.
hypotenuse ()    # Расчет гипотенузы прямоугольного треугольника.
{
hyp=$(bc -l << EOF
scale = 9
sqrt ( $1 * $1 + $2 * $2 )
EOF
)
# К сожалению, функции Bash не могут возвращать числа с плавающей запятой.
}

hypotenuse 3.68 7.31
echo "гипотенуза = $hyp"    # 8.184039344


exit 0
```

Пример 12-35. Вычисление числа "пи"

```
#!/bin/bash
# cannon.sh: Аппроксимация числа "пи".

# Это очень простой вариант реализации метода "Monte Carlo",
#+ математическое моделирование событий реальной жизни,
#+ для эмуляции случайного события используются псевдослучайные числа.

#  Допустим, что мы располагаем картой квадратного участка поверхности со стороной
квадрата 10000 единиц.
#  На этом участке, в центре, находится совершенно круглое озеро,
#+ с диаметром в 10000 единиц.
#  Т.е. озеро покрывает почти всю карту, кроме ее углов.
#  (Фактически -- это квадрат со вписанным кругом.)
#
#  Пусть по этому участку ведется стрельба железными ядрами из древней пушки
```

```
#  Все ядра падают где-то в пределах данного участка,
#+ т.е. либо в озеро, либо на сушу, по углам участка.
#  Поскольку озеро покрывает большую часть участка,
#+ то большинство ядер будет падать в воду.
#  Незначительная часть ядер будет падать на твердую почву.
#
#  Если произвести достаточно большое число неприцельных выстрелов по данному
участку,
#+ то отношение попаданий в воду к общему числу выстрелов будет примерно равно
#+ значению PI/4.
#
#  По той простой причине, что стрельба фактически ведется только
#+ по правому верхнему квадранту карты.
#  (Предыдущее описание было несколько упрощено.)
#
#  Теоретически, чем больше будет произведено выстрелов, тем точнее будет
результат.
#  Однако, сценарий на языке командной оболочки, в отличие от других языков
программирования,
#+ в которых доступны операции с плавающей запятой, имеет некоторые ограничения.
#  К сожалению, это делает вычисления менее точными.


DIMENSION=10000  # Длина стороны квадратного участка поверхности.
                 # Он же -- верхний предел для генератора случайных чисел.

MAXSHOTS=1000    # Количество выстрелов.
                 # 10000 выстрелов (или больше) даст лучший результат,
                                                                  # но потребует
значительного количества времени.
PMULTIPLIER=4.0  # Масштабирующий коэффициент.

get_random ()
{
SEED=$(head -1 /dev/urandom | od -N 1 | awk '{ print $2 }')
RANDOM=$SEED                                  #  Из примера "seeding-random.sh"

let "rnum = $RANDOM % $DIMENSION"             #  Число не более чем 10000.
echo $rnum
}

distance=        # Объявление глобальной переменной.
hypotenuse ()    # Расчет гипотенузы прямоугольного треугольника.
{                # Из примера "alt-bc.sh".
distance=$(bc -l << EOF
scale = 0
sqrt ( $1 * $1 + $2 * $2 )
```

```
EOF
)
#  Установка "scale" в ноль приводит к округлению результата "вниз",
#+ это и есть то самое ограничение, накладываемое командной оболочкой.
#  Что, к сожалению, снижает точность аппроксимации.
}


# main() {

# Инициализация переменных.
shots=0
splashes=0
thuds=0
Pi=0

while [ "$shots" -lt  "$MAXSHOTS" ]           # Главный цикл.
do

  xCoord=$(get_random)                        # Получить случайные координаты X и
Y.
  yCoord=$(get_random)
  hypotenuse $xCoord $yCoord                  #  Гипотенуза = расстоянию.
  ((shots++))

  printf "#%4d   " $shots
  printf "Xc = %4d  " $xCoord
  printf "Yc = %4d  " $yCoord
  printf "Distance = %5d  " $distance         #  Растояние от
                                              #+ центра озера,
                                              #+ с координатами (0,0).

  if [ "$distance" -le "$DIMENSION" ]
  then
    echo -n "ШЛЕП!  "                        # попадание в озеро
    ((splashes++))
  else
    echo -n "БУХ!    "                       # попадание на твердую почву
    ((thuds++))
  fi

  Pi=$(echo "scale=9; $PMULTIPLIER*$splashes/$shots" | bc)
  # Умножение на коэффициент 4.0.
  echo -n "PI ~ $Pi"
  echo

done
```

```
echo
echo "После $shots выстрела, примерное значение числа \"пи\" равно $Pi."
# Имеет тенденцию к завышению...
# Вероятно из-за ошибок округления и несовершенства генератора случайных чисел.
echo

# }

exit 0

#  Самое время задуматься над тем, является ли сценарий удобным средством
#+ для выполнения большого количества столь сложных вычислений.
#
#  Тем не менее, этот пример может расцениваться как
#  1) Доказательство возможностей языка командной оболочки.
#  2) Прототип для "обкатки" алгоритма перед тем как перенести
#+    его на высокоуровневые языки программирования компилирующего типа.
```

dc

Утилита **dc** (**d**esk **c**alculator) -- это калькулятор, использующий "Обратную Польскую Нотацию", и ориентированный на работу со стеком.

Многие стараются избегать испоьзования **dc**, из-за непривычной формы записи операндов и операций. Однако, **dc** имеет и своих сторонников.

Пример 12-36. Преобразование чисел из десятичной в шестнадцатиричную систему счисления

```
#!/bin/bash
# hexconvert.sh: Преобразование чисел из десятичной в шестнадцатиричную систему
счисления.

BASE=16     # Шестнадцатиричная.

if [ -z "$1" ]
then
  echo "Порядок использования: $0 number"
  exit $E_NOARGS
  # Необходим аргумент командной строки.
fi
# Упражнение: добавьте проверку корректности аргумента.



hexcvt ()
{
if [ -z "$1" ]
then
  echo 0
```

```
  return    # "Return" 0, если функции не был передан аргумент.
fi

echo ""$1" "$BASE" o p" | dc
#                 "o" устанавливает основание системы счисления для вывода.
#                   "p" выводит число, находящееся на вершине стека.
# См. 'man dc'.
return
}

hexcvt "$1"

exit 0
```

Изучение страниц *info* **dc** позволит детальнее разобраться с утилитой. Однако, отряд "гуру", которые могут похвастать своим знанием этой мощной, но весьма запутанной утилиты, весьма немногочислен.

Пример 12-37. Разложение числа на простые множители

```
#!/bin/bash
# factr.sh: Разложение числа на простые множители

MIN=2       # Не работает с числами меньше 2.
E_NOARGS=65
E_TOOSMALL=66

if [ -z $1 ]
then
  echo "Порядок использования: $0 number"
  exit $E_NOARGS
fi

if [ "$1" -lt "$MIN" ]
then
  echo "Исходное число должно быть больше или равно $MIN."
  exit $E_TOOSMALL
fi

# Упражнение: Добавьте проверку типа числа (не целые числа должны отвергаться).

echo "Простые множители для числа $1:"
# ------------------------------------------------------------------------------
echo "$1[p]s2[lip/dli%0=1dvsr]s12sid2%0=13sidvsr[dli%0=1lrli2+dsi!>.]ds.xd1<2" | dc
# ------------------------------------------------------------------------------
# Автор вышеприведенной строки: Michel Charpentier <charpov@cs.unh.edu>.
# Используется с его разрешения (спасибо).
```

```
exit 0
```

awk

Еще один способ выполнения математических операций, над числами с плавающей запятой, состоит в создании сценария-обертки, использующего математические функции awk.

Пример 12-38. Расчет гипотенузы прямоугольного треугольника

```
#!/bin/bash
# hypotenuse.sh: Возвращает "гипотенузу" прямоугольного треугольника.
#                ( корень квадратный от суммы квадратов катетов)

ARGS=2                # В сценарий необходимо передать два катета.
E_BADARGS=65          # Ошибка в аргументах.

if [ $# -ne "$ARGS" ] # Проверка количества аргументов.
then
  echo "Порядок использования: `basename $0` катет_1 катет_2"
  exit $E_BADARGS
fi


AWKSCRIPT=' { printf( "%3.7f\n", sqrt($1*$1 + $2*$2) ) } '
#            команды и параметры, передаваемые в awk


echo -n "Гипотенуза прямоугольного треугольника, с катетами $1 и $2, = "
echo $1 $2 | awk "$AWKSCRIPT"

exit 0
```