

# Вычисление с помощью dc

## Введение

dc (1) - нестандартный, но часто встречающийся настольный калькулятор с обратной полировкой. По словам Кена Томпсона, "dc - самый старый язык в Unix; он был написан на PDP-7 и портирован на PDP-11 до того, как был портирован [сам] Unix".

Исторически стандарт bc(1) был реализован как *интерфейс для dc*.

## Простой расчет

Вкратце, *обратная польская нотация* означает, что числа сначала помещаются в стек, а затем к ним применяется операция. Вместо того, чтобы писать `1+1`, вы пишете `1 1+`.

По умолчанию `dc`, в отличие `bc` от, ничего не печатается, результат помещается в стек. Вы должны использовать команду "p", чтобы напечатать элемент в верхней части стека. Таким образом, простая операция выглядит следующим образом:

```
$ dc <<< '1 1 + p'
2
```

Я использовал "here string", присутствующую в bash 3.x, ksh93 и zsh. если ваша оболочка не поддерживает это, вы можете использовать `echo '1 1+p' | dc` или, если у вас есть GNU dc, вы можете использовать `dc -e '1 1 +p'`.

Конечно, вы также можете просто запустить `dc` и ввести команды.

Классическими операциями являются:

- сложение: `+`
- вычитание: `-`
- деление: `/`
- умножение: `*`
- остаток (по модулю): `%`
- возведение в степень: `^`
- квадратный корень: `v`

GNU dc добавляет еще пару.

Для ввода отрицательного числа вам нужно использовать символ `_` (подчеркивание):

```
$ dc <<< '1_1-p'
2
```

Вы можете использовать *цифры* 0-9 и *буквы* A-F в качестве чисел, а точку ( . ) в качестве десятичной точки. A-F **должно** быть заглавными буквами, чтобы не путать с командами, указанными строчными символами. Число с буквой считается шестнадцатеричным:

```
dc <<< 'Ap'
10
```

**Выходные** данные по умолчанию преобразуются в **базовую 10**

## Масштаб и база

dc является вычислителем с абсолютной точностью, по умолчанию эта точность равна 0. таким образом dc <<< "5 4/p" , выводится "1".

Мы можем увеличить точность с помощью k команды. Он выводит значение в верхней части стека и использует его в качестве аргумента точности:

```
dc <<< '2k5 4/p' # выводит 1.25
dc <<< '4k5 4 /p' # выводит 1.2500
dc <<< '100k 2vp'
1.4142135623730950488016887242096980785696718753769480731766797379907
\
324784621070388503875343276415727
```

dc поддерживает аргументы *большой* точности.

Вы можете изменить базу, используемую для вывода (*печати*) чисел, o и базу, используемую для ввода (*ввода*) чисел с помощью i :

```
dc << EOF
20 p # выводит 20, вывод находится в базе 10
16o # результат теперь находится в базе 2 16
20p # выводит 14 в шестнадцатеричном формате
16i # вывод теперь в шестнадцатеричном формате
p # выводит 14, это не изменяет число в стеке
10p # выводит 10, вывод выполняется в базе 16
EOF
```

Примечание: при изменении входного значения изменяется база для всех команд, включая i :

```
dc << EOF
16i 16o # база равна 16 для ввода и вывода
10p # выводит 10
10i # ! установите базовое значение равным 10, то есть 16 десятичным
17p # выводит 17
EOF
```

Этот код выводит 17, хотя мы могли бы подумать, что `10i` это возвращает базовое значение обратно в 10, и, следовательно, число должно быть преобразовано в шестнадцатеричное и напечатано как 11. Проблема в том, что 10 было введено при вводе базы 16, таким образом, база была установлена на 10 шестнадцатеричных, то есть 16 десятичных.

```
dc << EOF
16o16o10p #выводит 10
Ai # установите базовое значение в A в шестнадцатеричном формате, то
    есть 10
17p # выводит 11 в базе 16
EOF
```

## Стек

Есть две основные команды для управления стеком:

- `d` дублирует верхнюю часть стека
- `c` очищает стек

```
$ dc << EOF
2 # поместить 2 в стек
d # дублировать, т.е. поместить еще 2 в стек
* p # умножить и вывести
c p # очистить и распечатать
EOF
4
dc: стек пуст
```

`c p` как и следовало ожидать, приводит к ошибке, поскольку `c` удаляет все в стеке.

*Примечание: мы можем использовать `#` для добавления комментариев в сценарий.*

Если вы заблудились, вы можете проверить (т. Е. Распечатать) стек с помощью команды `f`. Стек остается неизменным:

```
dc <<< '1 2 d 4+f'
6
2
1
```

Обратите внимание, что сначала печатается первый элемент, который будет извлечен из стека, если вы привыкли к калькулятору HP, все наоборот.

Не стесняйтесь приводить `f` примеры из этого руководства, это не меняет результат, и это хороший способ увидеть, что происходит.

## Регистры

В dc руководстве GNU говорится, что dc имеет не менее **256 регистров** в зависимости от диапазона unsigned char . Я не уверен, как вы должны использовать нулевой байт. Использовать регистр просто:

```
dc <
12 # поместить 12 в стек
sa # удалить его из стека (стеков) и поместить в регистр 'a'
10 # поместить 10 в стек
la # прочитать (l) значение регистра 'a' и поместить его в стек
+p # добавьте 2 значения и выведите
EOF
```

Приведенный выше фрагмент использует новые строки для вставки комментариев, но на самом деле это не имеет значения, вы можете использовать `echo '12sa10la+p'|dc` с теми же результатами.

Регистр может содержать больше, чем просто значение, **каждый регистр сам по себе является стеком**.

```
dc <
12sa #сохранить 12 в 'a'
6Sa # с большой буквы S 6 удаляется
# из основного стека и помещается в стек "a"
lap # выводит 6, значение в верхней части стека "a"
круг # все еще выводит 6
Lap # также выводит 6, но с заглавной буквы L, это переводит значение
в "a"
# к основному стеку и извлекает его из стека "a"
lap # выводит 12, который теперь находится в верхней части стека
EOF
```

## Макросы

dc позволяет помещать произвольные строки в стек, когда строки заключены в `[]` . Вы можете распечатать его с `p` помощью: `dc <<< '[hello World!]'p` и вы можете оценить его с помощью `x`: `dc <<< '[1 2+]xp'` .

Это не так интересно, пока не будет объединено с регистрами. Во-первых, допустим, мы хотим вычислить квадрат числа (не забудьте включить `f` , если вы заблудились!):

```
dc << EOF
3 # помещаем наше число в стек
d # дублируем его, т.е. снова помещаем 3 в стек
d ** p # снова дублируем и вычисляем произведение и печатаем
EOF
```

Теперь у нас есть несколько кубов для вычисления, которые мы могли бы использовать `dd**` несколько раз или использовать макрос.

```
dc << EOF
[dd **] # вставить строку
sa # сохранить ее в регистре a
3 # нажмите 3 в стеке
слабый # вставьте строку "dd **" в стек и выполните ее
p # выведите результат
4laxp # та же операция для 4, в одной строке
EOF
```

## Условные выражения и циклы

dc может выполнять макрос, хранящийся в регистре, используя `lR` х комбинацию, но он также может выполнять макросы условно. `>a` выполнит макрос, сохраненный в регистре `a`, если вершина стека *больше, чем* второй элемент стека. Примечание: верхняя часть стека содержит последнюю запись. При написании это выглядит как обратное тому, что мы привыкли читать:

```
dc << EOF
[[Hello World] p] sR # сохранить в 'R' макрос, который печатает Hello
World
2 1> R # ничего не делать 1 вверху 2 - второй элемент
1 2 > R # выводит Привет, мир
EOF
```

У некоторых dc есть `>R` `<R` `=R`, у GNU dc было еще несколько, проверьте свое руководство. Обратите внимание, что тест "потребляет" свои операнды: 2 первых элемента удаляются из стека (вы можете убедиться, что `dc <<< "[f]sR 2 1 >R 1 2 >R f"` это ничего не печатает)

Вы заметили, как мы можем *включать* макрос (строку) в макрос? и поскольку dc мы полагаемся на стек, мы можем, по сути, использовать макрос рекурсивно (приготовьте свою любимую комбинацию клавиш control-c ;)) :

```
dc << EOF
[ [Hello World] p # наш макрос начинается с вывода Hello World
  lRx ] # а затем выполняет макрос в R
sR # мы сохраняем его в регистре R
lRx # и, наконец, выполняем его.
EOF
```

У нас есть рекурсивность, у нас есть тест, у нас есть циклы:

```
dc << EOF
[ li # помещаем наш индекс i в стек
p # распечатываем его, чтобы увидеть, что происходит
1 - # мы уменьшаем индекс на единицу
si # сохраняем уменьшенный индекс (i = i-1)
0 li >L # если i > 0, то выполните L
] sL # сохраните наш макрос с именем L

10 si # давайте присвоим нашему индексу значение 10
lLx # и запускаем наш цикл
EOF
```

Конечно, код, написанный таким образом, слишком легко читается! Обязательно удалите все эти лишние пробелы, новые строки и комментарии:

```
dc <<< '[lip1-si0li>L]sL10silLx'
dc <<< '[p1-d0
```

Я позволю вам разобраться во втором примере, это не сложно, он использует стек вместо регистра для индекса.

## Далее

*Проверьте свое руководство по dc, я не описал все, например, массивы (задокументированы только с помощью "; : используются bc (1) для операций с массивами" на solaris, вероятно, потому, что echo '1 0: a 0Sa 2 0: a La 0; ap' | dc приводит к ошибке сегментации (ядродамп), последняя версия solaris использует GNU dc)*

Вы можете найти больше информации и программ dc здесь:

- [http://en.wikipedia.org/wiki/Dc\\_\(Unix\)](http://en.wikipedia.org/wiki/Dc_(Unix)) ([http://en.wikipedia.org/wiki/Dc\\_\(Unix\)](http://en.wikipedia.org/wiki/Dc_(Unix)))

И еще один пример, а также реализация dc на python здесь:

- [http://en.literateprograms.org/Category:Programming\\_language:dc](http://en.literateprograms.org/Category:Programming_language:dc)  
([http://en.literateprograms.org/Category:Programming\\_language:dc](http://en.literateprograms.org/Category:Programming_language:dc))
- [http://en.literateprograms.org/Desk\\_calculator\\_%28Python%29](http://en.literateprograms.org/Desk_calculator_%28Python%29)  
([http://en.literateprograms.org/Desk\\_calculator\\_%28Python%29](http://en.literateprograms.org/Desk_calculator_%28Python%29))

Руководство для dc 1971 года от Bell Labs:

- <http://cm.bell-labs.com/cm/cs/who/dmr/man12.ps> (<http://cm.bell-labs.com/cm/cs/who/dmr/man12.ps>) (мертвая ссылка)



## Обсуждение

ричард Хартсхорн, [2012/02/26 22:20 \(\)](#), [2012/03/04 17:08 \(\)](#)

Я использую `gnu dc` от 2006 года, действительно играю с ним, и оператор `|` дает разные результаты...



```
dc -e "0 k 3 4 5 | p q"
```

дает 1, правильно!  $(3^4) \bmod 5$  равен 81  $\bmod 5$  равен 1.

Здесь появляется шаткость...

```
dc -e "8 k 3 4 5 | p q"
```

дает ноль, неверно! Несмотря на то, что настройка масштаба не имеет отношения к вычислению. Как вы думаете, ошибка? Или историческая "особенность", к которой привыкли все остальные.

 [howto/calculate-dc.txt](#)  Последнее редактирование: 2018/06/21 23:36 автор [izxle](#)

---

Этот сайт поддерживается Performing Databases - вашими экспертами по администрированию баз данных

---

Bash Hackers Wiki

---



Если не указано иное, содержимое этой вики лицензируется по следующей лицензии:  
Лицензия GNU Free Documentation 1.3