Расширение скобок

```
{string1,string2,...,stringN}
{<НАЧАЛО>..<КОНЕЦ>}

{<НАЧАЛО>..<КОНЕЦ>..<ВКЛ>} (Bash 4)

<ПРЕФИКС>{........}

{........}< СУФФИКС>

<ПРЕФИКС>{........}< СУФФИКС>
```

Расширение в фигурных скобках используется для генерации произвольных строк. Указанные строки используются для генерации **всех возможных комбинаций** с необязательными окружающими префиксами и суффиксами.

Обычно оно используется для генерации массовых аргументов для команды, которые следуют определенной схеме именования.

Это самый первый шаг в обработке расширения, важно это понимать. Когда вы используете

```
echo {a,b}$PATH
```

тогда расширение фигурных скобок **не расширяет переменную** - это делается на более **позднем этапе**. Расширение скобок просто делает его:

```
echo a $PATH b $PATH
```

Еще одна распространенная ошибка - предположить, что диапазон like `{1..200}` может быть выражен с использованием переменных `{$a..$b}` . Из-за того, что я описал выше, это **просто невозможно**, потому что это самый первый шаг в создании расширений. Возможный способ добиться этого, если вы действительно не можете справиться с этим другим способом, - это использовать `eval` команду, которая в основном дважды вычисляет командную строку:

```
eval echo {$a .. $b}
```

Например, при внедрении в цикл for :

```
для i в $(eval echo {$a..$b})
```

Для этого требуется, чтобы вся команда была правильно экранирована, чтобы избежать неожиданных расширений. Если расширение последовательности должно быть назначено массиву, возможен другой метод с использованием команд объявления:

```
объявить -a 'pics=(img{'"$a..$b"'}.png)'; mv "${pics[@]}" ../imgs
```

Это значительно безопаснее, но все равно нужно быть осторожным, чтобы контролировать значения $ a и $ b. Важны как точное цитирование, так и явное включение "-a".

Расширение в виде фигурных скобок представлено в двух основных формах: **списках строк** и **диапазонах**.

Его можно включать и выключать во время выполнения с помощью `set` встроенного и опции `-B` и `+B` или опции long `braceexpand` . Если расширение фигурных скобок включено, список строк в `SHELLOPTS` содержит `braceexpand` .

Списки строк

```
{string1,string2,...,stringN}
```

Без необязательных строк префикса и суффикса результатом является просто разделенный пробелом список заданных строк:

```
$ echo {Я, хочу, мои, деньги, обратно}
Я хочу вернуть свои деньги
```

При использовании строк с префиксом или суффиксом результатом является разделенный пробелом список **всех возможных комбинаций** строк, указанных с префиксом или суффиксом:

```
$ echo _ {Я, хочу, мои, деньги, обратно}
_ Я _ хочу _ свои _ деньги _ назад

$ echo {Я, хочу, мои, деньги, обратно}_
Я хочу вернуть свои деньги_

$ echo _ {Я, хочу, мои, деньги, обратно}-
_I- _want- _my- _money- _back-
```

Расширение в фигурных скобках выполняется только в том случае, если данный список строк действительно является **списком строк**, т.Е. Если существует минимум один " , " (запятая)! Что-то вроде `{money}` не расширяется до чего-то особенного, на самом деле это всего лишь текст " `{money}` ".

Диапазоны

```
{<НАЧАЛО>..<КОНЕЦ>}
```

Расширение в скобках с использованием диапазонов записывается с указанием начальной точки и конечной точки диапазона. Это "выражение последовательности". Последовательности могут быть двух типов

- целые числа (необязательно дополненные нулем, необязательно с заданным шагом)
- Персонажи

```
$ echo {5..12}
5 6 7 8 9 10 11 12

$ echo {c..k}
c d e f g h i j k
```

При смешивании этих двух типов расширение фигурных скобок **не** выполняется:

```
$ echo {5..k}
{5..k}
```

Когда вы обнуляете одно из чисел (или оба) в диапазоне, то сгенерированный диапазон также дополняется нулем:

```
$ echo {01..10}
01 02 03 04 05 06 07 08 09 10
```

В конце этой статьи есть глава об изменениях расширения скобок в Bash 4.

Аналогично расширению с использованием stringlists, вы можете добавлять строки с префиксами и суффиксами:

```
$ echo 1.{0..9}
1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9

$ echo --- {A..E}---
---A--- --- B-----C ----- D--- --E---
```

Объединение и вложение

Когда вы объединяете несколько расширений фигурных скобок, вы эффективно используете расширение фигурных скобок в качестве префикса или суффикса для другого. Давайте сгенерируем все возможные комбинации прописных букв и цифр:

```
$ echo {A..Z}{0..9}
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 C0 C1 C2
C3 C4 C5 C6
C7 C8 C9 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9
F0 F1 F2 F3
F4 F5 F6 F7 F8 F9 G0 G1 G2 G3 G4 G5 G6 G7 G8 G9 H0 H1 H2 H3 H4 H5 H6
H7 H8 H9 I0
I1 I2 I3 I4 I5 I6 I7 I8 I9 J0 J1 J2 J3 J4 J5 J6 J7 J8 J9 K0 K1 K2 K3
K4 K5 K6 K7
K8 K9 L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 M0 M1 M2 M3 M4 M5 M6 M7 M8 M9 N0
N1 N2 N3 N4
N5 N6 N7 N8 N9 O0 O1 O2 O3 O4 O5 O6 O7 O8 O9 P0 P1 P2 P3 P4 P5 P6 P7
P8 P9 Q0 Q1
Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 S0 S1 S2 S3 S4
S5 S6 S7 S8
S9 T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 U0 U1 U2 U3 U4 U5 U6 U7 U8 U9 V0 V1
V2 V3 V4 V5
V6 V7 V8 V9 W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 X0 X1 X2 X3 X4 X5 X6 X7 X8
X9 Y0 Y1 Y2
Y3 Y4 Y5 Y6 Y7 Y8 Y9 Z0 Z1 Z2 Z3 Z4 Z5 Z6 Z7 Z8 Z9
```

Привет.. это **сэкономит вам время на написании** 260 строк!

Расширения фигурных скобок могут быть вложенными, но их слишком много, как правило, приводит к тому, что вы немного теряете обзор 🧐

Вот пример создания алфавита: сначала заглавные буквы, затем строчные:

```
$ echo {{A..Z},{a..z}}
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i
j k l m n o p q r s t u v w x y z
```

Общее использование и примеры

Массовая загрузка из Интернета

В этом примере `wget` используется для загрузки документации, которая разбита на несколько пронумерованных веб-страниц.

`wget` я не увижу твоих брекетов. Он увидит **6 разных URL**-адресов для загрузки.

```
wget http://docs.example.com/documentation/slides_part {1,2,3,4,5,6}.
html
```

Конечно, это возможно и даже проще сделать с помощью последовательности:

```
wget http://docs.example.com/documentation/slides_part {1..6}.html
```

Создайте структуру подкаталогов

У тебя тяжелая жизнь? Давайте немного упростим ситуацию - для этого и существуют оболочки.

```
mkdir /home/bash/test/{foo, bar, baz, cat, dog}
```

Генерируйте числа с префиксом 001 002 ...

- Использование префикса:

```
для i в 0{1..9} 10; выполните printf "%s \ n" "$ i";готово
```

Если вам нужно создать слова со встроенным числом, вы можете использовать вложенную фигурную скобку:

```
printf "%s \n" img{00{1..9},0{10..99},{100..999}}. png
```

- Форматирование чисел с помощью printf:

```
echo $(printf "img%02d.png " {1..99})
```

Смотрите текст ниже для нового метода Bash 4.

Повторяющиеся аргументы или слова

```
somecommand -в -в -в -в -в
```

Может быть записано как

```
somecommand -v{,,,,}
```

... что является своего рода взломом, но, эй, это работает.

Больше удовольствия

Наиболее оптимальное возможное расширение фигурных скобок для расширения n аргументов, конечно, состоит из n простых множителей. Мы можем использовать программу "factor" в комплекте с GNU coreutils, чтобы создать расширение фигурных скобок, которое расширит любое количество аргументов.

```
функция braceify {
 [[ $1 == +([[: digit:]]) ]] || возвращает
набор текста -a a
read -ra a < <(коэффициент "$1")
 вычислить "echo $(printf '{$(printf ,%%.s {1..%s})}' "${a[@]:
1}")"
}

printf 'evalprintf "$arg"%s' "$(braceify 1000000)"
```

"Braceify" генерирует сам код расширения. В этом примере мы вводим этот вывод в шаблон, который отображает самый краткий код расширения фигурных `"$arg"` скобок, который при оценке расширился бы в 1 000 000 раз. В этом случае вывод будет:

```
вычислить printf "$arg"{,,}{,,}{,,}{,,}{,,}{,,}{,,}{,,}{,,,,}{,,,,}
{,,,,}{,,,,}{,,,,}{,,,,}{,,,,}
```

Новое в Bash 4.0

Расширение с добавлением нулевого числа

Префикс любого из чисел в числовом диапазоне с `0` , чтобы дополнить расширенные числа правильным количеством нулей:

```
$ echo {0001..5}
0001 0002 0003 0004 0005
```

Увеличение

Теперь можно указать приращение с помощью диапазонов:

```
{<НАЧАЛО>..<КОНЕЦ>..<ВКЛ>}
```

`<INCR>` является числовым, вы можете использовать отрицательное целое число, но правильный знак в любом случае выводится из порядка `<START>` и `<END>` .

```
$ echo {1..10..2}
1 3 5 7 9
$ echo {10..1..2}
10 8 6 4 2
```

Интересная особенность: спецификация увеличения также работает для диапазонов букв:

```
$ echo {a..z..3}
a d g j m p s v y
```

Смотрите также

- Введение в расширение и замену

Обсуждение

Элджей, 2010/08/25 01:02 ()

Что касается вашего заявления... "Еще одна распространенная ошибка - предположить, что диапазон, подобный {1..200}, может быть выражен с помощью переменных с использованием {$a .. $b}".

Я могу обойти это, используя eval.

$ A = 1

$ B = 100

$ eval echo $ {A .. $ B}

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Ян Шампера, 2010/08/25 06:03 ()

Да, я знаю, что это возможно с помощью eval. Я поклялся себе никогда не **рекомендовать** eval, но в этом случае я просто "отмечу это" выше, так что я в безопасности :)

Спасибо за отзыв.

Раджи, 2011/02/07 06:03 ()

приведите несколько примеров для условного оператора n зацикливания..

Ян Шампера, 2011/02/12 08:56 ()

Здесь. для расширения скобки? Тогда я не понимаю, что вы имеете в виду.

Дыня (http://neoretro.net), 2012/03/27 23:59 ()

Можно ли получить доступ к тому, какая итерация используется в данный момент?

Например:

mv {1..5}.что-то.{1..5} $ n1.$ n2.что-то

* Где $ n1 и $ n2 - это ссылка bash на текущую итерацию

Так что это будет его поведение:

mv 1.что-то.1 1.1.что-то

mv 1.что-то.1 1.2.что-то

mv 1.что-то.1 1.3.что-то

Спасибо: D

Ян Шампера, 2012/04/07 08:07 ()

Нет, это невозможно. Расширение фигурных скобок больше похоже на функциональность текстового макроса. Это происходит раньше всего остального

Дэниел, 2012/04/11 17:21 ()

Можно ли использовать расширение фигурных скобок в цикле for для переименования нескольких файлов? Если нет, то почему?

Ян Шампера, 2012/04/21 10:44 ()

Это возможно. Как в "генераторе чисел". То, что вы делаете с этими числами, не зависит от этой темы:

```
для i в {1..10}; сделать
 ...
Выполнено

# эквивалент: для i в 1 2 3 4 5 6 7 8 9 10; делать
```

ур, 2012/07/28 00:57 ()

Есть ли хороший способ присвоить результат расширения? Я знаю, что могу использовать подоболочку, например

```
x= $(echo {1..5})
```

но это некрасиво.

джонатан Кросс (http://www.jonathanccross.com), 2015/01/21 12:24 ()

В 4.3.11 я могу просто использовать это:

```
x=({1..5})
```

Результат:

```
echo ${x[@]}
1 2 3 4 5
```

Чарли Дайсон, 08.02.14 11:00 ()

Относительный порядок приоритета между расширением скобок и конвейером подоболочек изменился в Bash 4:

bash3_machine$ paste -d \| <(echo {first,second}) первый | второй

bash4_machine$ paste -d \| <(echo {first,second}) первая секунда

Я нашел старое поведение более полезным - например, для сравнения вывода длинной цепочки команд в двух разных файлах.

Ян Шампера, 2012/09/03 09:38 ()

Я не уверен, но я думаю, что это было больше похоже на "исправление ошибок". Новое поведение угрожает коду внутри `<()` отдельно. Это более прямолинейно.

(Я не говорю, что это более полезно или бесполезно, я просто говорю о том, что, по моему мнению, было причиной)

Филипп Петринко, 2014/06/19 07:13 ()

Я отметил это дополнительное поведение, учитывая нулевое заполнение:

Если начальный и конечный диапазоны дополнены нулем, если заполнение _different_ , то будет использоваться самый длинный.

таким образом, это будет использовать 6 цифр, а не 3!

```
для x в {001..000010} ; выполнить эхо “заполнение : $x:”; готово
```

HTH

– Филипп

Густав, 2014/12/30 00:07 ()

В (симпатичной) функции braceify есть ошибка: она должна вычитать единицу из каждого простого множителя.

{,,} расширяется три раза, а не два.

Кроме того, оно не будет генерировать кратчайшее представление для множителей 4; это было бы {,,,}, а не {,}{,}

syntax/expansion/brace.txt Последнее редактирование: 2020/06/28 01:16 автор fgrouse