Bash-скрипты, часть 2: циклы

<https://likegeeks.com/bash-scripting-step-step-part2/>

* [Блог компании RUVDS.com](https://habr.com/company/ruvds/),
* [Настройка Linux](https://habr.com/hub/linux/),
* [Серверное администрирование](https://habr.com/hub/s_admin/)
* Перевод

[Bash-скрипты: начало](https://habrahabr.ru/company/ruvds/blog/325522/)  
[Bash-скрипты, часть 2: циклы](https://habrahabr.ru/company/ruvds/blog/325928/)  
[Bash-скрипты, часть 3: параметры и ключи командной строки](https://habrahabr.ru/company/ruvds/blog/326328/)   
[Bash-скрипты, часть 4: ввод и вывод](https://habrahabr.ru/company/ruvds/blog/326594/)  
[Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями](https://habrahabr.ru/company/ruvds/blog/326826/)   
[Bash-скрипты, часть 6: функции и разработка библиотек](https://habrahabr.ru/company/ruvds/blog/327248/)  
[Bash-скрипты, часть 7: sed и обработка текстов](https://habrahabr.ru/company/ruvds/blog/327530/)  
[Bash-скрипты, часть 8: язык обработки данных awk](https://habrahabr.ru/company/ruvds/blog/327754/)  
[Bash-скрипты, часть 9: регулярные выражения](https://habrahabr.ru/company/ruvds/blog/327896/)  
[Bash-скрипты, часть 10: практические примеры](https://habrahabr.ru/company/ruvds/blog/328346/)  
[Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит](https://habrahabr.ru/company/ruvds/blog/328436/)

В [прошлый раз](https://habrahabr.ru/company/ruvds/blog/325522/) мы рассказали об основах программирования для bash. Даже то немногое, что уже разобрано, позволяет всем желающим приступить к автоматизации работы в Linux. В этом материале продолжим рассказ о bash-скриптах, поговорим об управляющих конструкциях, которые позволяют выполнять повторяющиеся действия. Речь идёт о циклах for и while, о методах работы с ними и о практических примерах их применения.  
  
[](https://habrahabr.ru/company/ruvds/blog/325929/)  
  
**Внимание:** в посте спрятана выгода!  
  
[https://habrastorage.org/files/1ba/550/d25/1ba550d25e8846ce8805de564da6aa63.png](https://ruvds.com/ru-rub/#order)

Циклы for

Оболочка bash поддерживает циклы for, которые позволяют организовывать перебор последовательностей значений. Вот какова базовая структура таких циклов:

for var in list

do

команды

done

В каждой итерации цикла в переменную var будет записываться следующее значение из списка list. В первом проходе цикла, таким образом, будет задействовано первое значение из списка. Во втором — второе, и так далее — до тех пор, пока цикл не дойдёт до последнего элемента.

Перебор простых значений

Пожалуй, самый простой пример цикла for в bash-скриптах — это перебор списка простых значений:

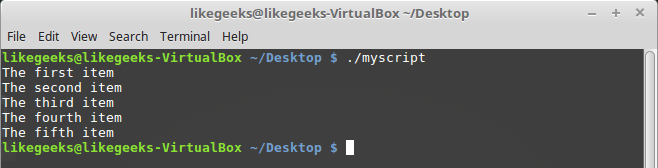
#!/bin/bash

for var in first second third fourth fifth

do

echo The  $var item

done

Ниже показаны результаты работы этого скрипта. Хорошо видно, что в переменную $varпоследовательно попадают элементы из списка. Происходит так до тех пор, пока цикл не дойдёт до последнего из них.  
  
  
*Простой цикл for*  
  
Обратите внимание на то, что переменная $var сохраняет значение при выходе из цикла, её содержимое можно менять, в целом, работать с ней можно как с любой другой переменной.

Перебор сложных значений

В списке, использованном при инициализации цикла for, могут содержаться не только простые строки, состоящие из одного слова, но и целые фразы, в которые входят несколько слов и знаков препинания. Например, всё это может выглядеть так:

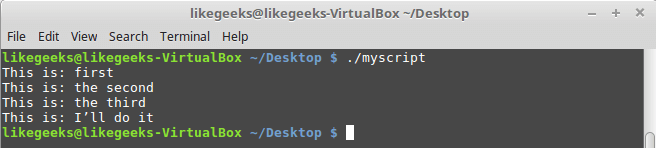
#!/bin/bash

for var in first "the second" "the third" "I’ll do it"

do

echo "This is: $var"

done

Вот что получится после того, как этот цикл пройдётся по списку. Как видите, результат вполне ожидаем.  
  
  
*Перебор сложных значений*  
TNW-CUS-FMP — промо-код на 10% скидку на наши услуги, доступен для активации в течение 7 дней"

Инициализация цикла списком, полученным из результатов работы команды

Ещё один способ инициализации цикла for заключается в передаче ему списка, который является результатом работы некоей команды. Тут используется подстановка команд для их исполнения и получения результатов их работы.

#!/bin/bash

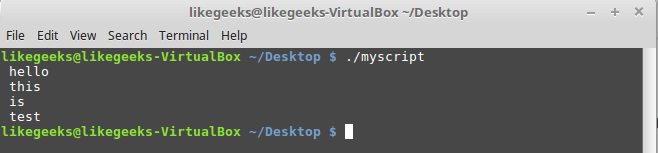
file="myfile"

for var in $(cat $file)

do

echo " $var"

done

В этом примере задействована команда cat, которая читает содержимое файла. Полученный список значений передаётся в цикл и выводится на экран. Обратите внимание на то, что в файле, к которому мы обращаемся, содержится список слов, разделённых знаками перевода строки, пробелы при этом не используются.  
  
  
*Цикл, который перебирает содержимое файла*  
  
Тут надо учесть, что подобный подход, если ожидается построчная обработка данных, не сработает для файла более сложной структуры, в строках которого может содержаться по несколько слов, разделённых пробелами. Цикл будет обрабатывать отдельные слова, а не строки.   
  
Что, если это совсем не то, что нужно?

Разделители полей

Причина вышеописанной особенности заключается в специальной переменной окружения, которая называется IFS (Internal Field Separator) и позволяет указывать разделители полей. По умолчанию оболочка bash считает разделителями полей следующие символы:

* Пробел
* Знак табуляции
* Знак перевода строки

Если bash встречает в данных любой из этих символов, он считает, что перед ним — следующее самостоятельное значение списка.  
  
Для того, чтобы решить проблему, можно временно изменить переменную среды IFS. Вот как это сделать в bash-скрипте, если исходить из предположения, что в качестве разделителя полей нужен только перевод строки:

IFS=$'\n'

После добавления этой команды в bash-скрипт, он будет работать как надо, игнорируя пробелы и знаки табуляции, считая разделителями полей лишь символы перевода строки.

#!/bin/bash

file="/etc/passwd"

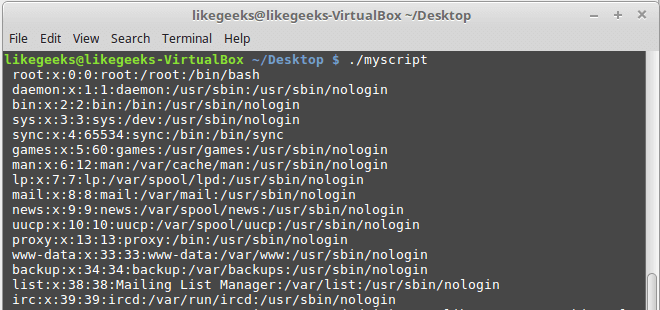
IFS=$'\n'

for var in $(cat $file)

do

echo " $var"

done

Если этот скрипт запустить, он выведет он именно то, что от него требуется, давая, в каждой итерации цикла, доступ к очередной строке, записанной в файл.  
  
  
*Построчный обход содержимого файла в цикле for*  
  
Разделителями могут быть и другие символы. Например, выше мы выводили на экран содержимое файла /etc/passwd. Данные о пользователях в строках разделены с помощью двоеточий. Если в цикле нужно обрабатывать подобные строки, IFS можно настроить так:

IFS=:

Обход файлов, содержащихся в директории

Один из самых распространённых вариантов использования циклов for в bash-скриптах заключается в обходе файлов, находящихся в некоей директории, и в обработке этих файлов.  
  
Например, вот как можно вывести список файлов и папок:

#!/bin/bash

for file in /home/likegeeks/\*

do

if [ -d "$file" ]

then

echo "$file is a directory"

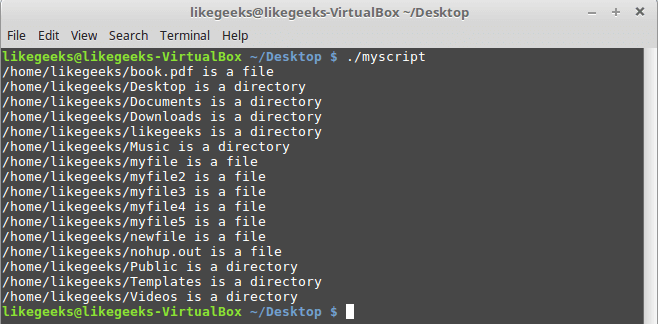
elif [ -f "$file" ]

then

echo "$file is a file"

fi

done

Если вы разобрались с [предыдущим материалом](https://habrahabr.ru/company/ruvds/blog/325522/) из этой серии статей, вам должно быть понятно устройство конструкции if-then, а так же то, как отличить файл от папки. Если вам сложно понять вышеприведённый код, перечитайте этот материал.  
  
Вот что выведет скрипт.  
  
  
*Вывод содержимого папки*  
  
Обратите внимание на то, как мы инициализируем цикл, а именно — на подстановочный знак «\*» в конце адреса папки. Этот символ можно воспринимать как шаблон, означающий: «все файлы с любыми именами». он позволяет организовать автоматическую подстановку имён файлов, которые соответствуют шаблону.  
  
При проверке условия в операторе if, мы заключаем имя переменной в кавычки. Сделано это потому что имя файла или папки может содержать пробелы.

Циклы for в стиле C

Если вы знакомы с языком программирования C, синтаксис описания bash-циклов for может показаться вам странным, так как привыкли вы, очевидно, к такому описанию циклов:

for (i = 0; i < 10; i++)

{

printf("number is %d\n", i);

}

В bash-скриптах можно использовать циклы for, описание которых выглядит очень похожим на циклы в стиле C, правда, без некоторых отличий тут не обошлось. Схема цикла при подобном подходе выглядит так:

for (( начальное значение переменной ; условие окончания цикла; изменение переменной ))

На bash это можно написать так:

for (( a = 1*; a < 10; a++ ))*

А вот рабочий пример:

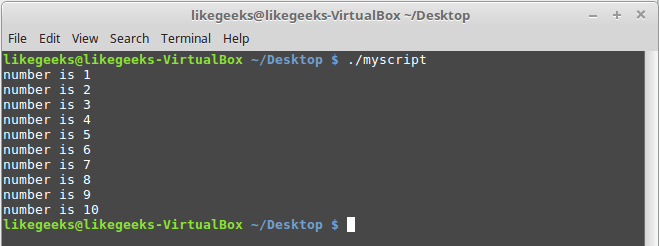
#!/bin/bash

for (( i=1; i <= 10; i++ ))

do

echo "number is $i"

done

Этот код выведет список чисел от 1 до 10.  
  
  
*Работа цикла в стиле C*

Цикл while

Конструкция for — не единственный способ организации циклов в bash-скриптах. Здесь можно пользоваться и циклами while. В таком цикле можно задать команду проверки некоего условия и выполнять тело цикла до тех пор, пока проверяемое условие возвращает ноль, или сигнал успешного завершения некоей операции. Когда условие цикла вернёт ненулевое значение, что означает ошибку, цикл остановится.  
  
Вот схема организации циклов while  
while команда проверки условия  
do  
другие команды  
done  
  
Взглянем на пример скрипта с таким циклом:

#!/bin/bash

var1=5

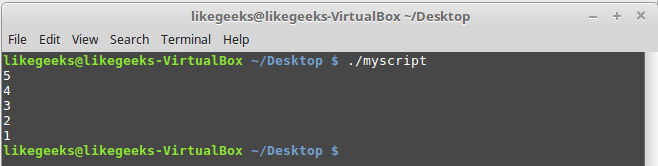
while [ $var1 -gt 0 ]

do

echo $var1

var1=$[ $var1 - 1 ]

done

На входе в цикл проверяется, больше ли нуля переменная $var1. Если это так, выполняется тело цикла, в котором из значения переменной вычитается единица. Так происходит в каждой итерации, при этом мы выводим в консоль значение переменной до его модификации. Как только $var1 примет значение 0, цикл прекращается.  
  
  
*Результат работы цикла while*  
  
Если не модифицировать переменную $var1, это приведёт к попаданию скрипта в бесконечный цикл.

Вложенные циклы

В теле цикла можно использовать любые команды, в том числе — запускать другие циклы. Такие конструкции называют вложенными циклами:

#!/bin/bash

for (( a = 1; a <= 3; a++ ))

do

echo "Start $a:"

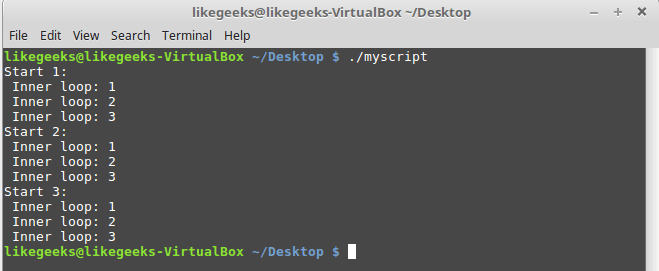
for (( b = 1; b <= 3; b++ ))

do

echo " Inner loop: $b"

done

done

Ниже показано то, что выведет этот скрипт. Как видно, сначала выполняется первая итерация внешнего цикла, потом — три итерации внутреннего, после его завершения снова в дело вступает внешний цикл, потом опять — внутренний.  
  
  
*Вложенные циклы*

Обработка содержимого файла

Чаще всего вложенные циклы используют для обработки файлов. Так, внешний цикл занимается перебором строк файла, а внутренний уже работает с каждой строкой. Вот, например, как выглядит обработка файла /etc/passwd:

#!/bin/bash

IFS=$'\n'

for entry in $(cat /etc/passwd)

do

echo "Values in $entry –"

IFS=:

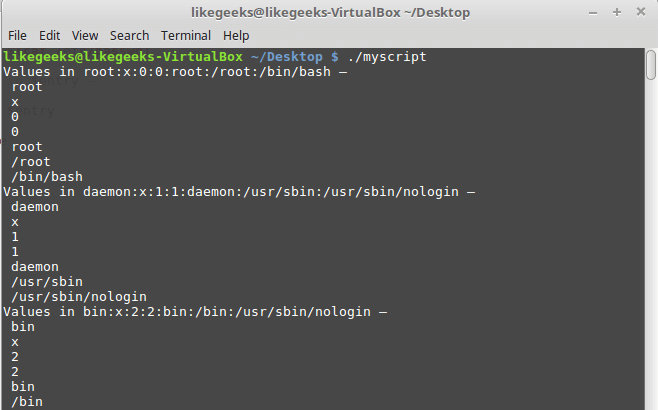
for value in $entry

do

echo " $value"

done

done

В этом скрипте два цикла. Первый проходится по строкам, используя в качестве разделителя знак перевода строки. Внутренний занят разбором строк, поля которых разделены двоеточиями.  
  
  
*Обработка данных файла*  
  
Такой подход можно использовать при обработке файлов формата CSV, или любых подобных файлов, записывая, по мере надобности, в переменную окружения IFS символ-разделитель.

Управление циклами

Возможно, после входа в цикл, нужно будет остановить его при достижении переменной цикла определённого значения, которое не соответствует изначально заданному условию окончания цикла. Надо ли будет в такой ситуации дожидаться нормального завершения цикла? Нет конечно, и в подобных случаях пригодятся следующие две команды:

* break
* continue

Команда break

Эта команда позволяет прервать выполнение цикла. Её можно использовать и для циклов for, и для циклов while:

#!/bin/bash

for var1 in 1 2 3 4 5 6 7 8 9 10

do

if [ $var1 -eq 5 ]

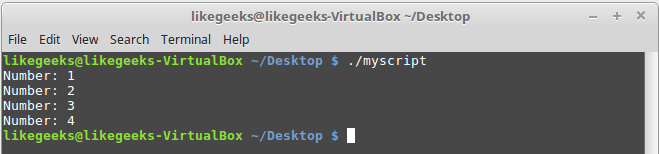
then

break

fi

echo "Number: $var1"

done

Такой цикл, в обычных условиях, пройдётся по всему списку значений из списка. Однако, в нашем случае, его выполнение будет прервано, когда переменная $var1 будет равна 5.  
  
  
*Досрочный выход из цикла for*  
  
Вот — то же самое, но уже для цикла while:

#!/bin/bash

var1=1

while [ $var1 -lt 10 ]

do

if [ $var1 -eq 5 ]

then

break

fi

echo "Iteration: $var1"

var1=$(( $var1 + 1 ))

done

Команда break, исполненная, когда значение $var1 станет равно 5, прерывает цикл. В консоль выведется то же самое, что и в предыдущем примере.

Команда continue

Когда в теле цикла встречается эта команда, текущая итерация завершается досрочно и начинается следующая, при этом выхода из цикла не происходит. Посмотрим на команду continue в цикле for:

#!/bin/bash

for (( var1 = 1; var1 < 15; var1++ ))

do

if [ $var1 -gt 5 ] && [ $var1 -lt 10 ]

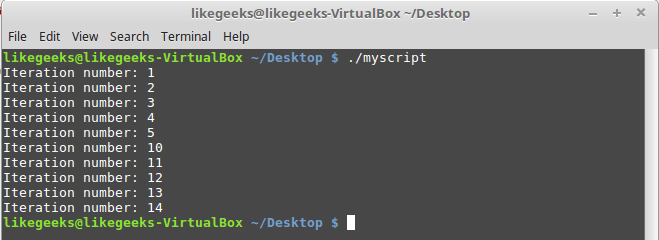
then

continue

fi

echo "Iteration number: $var1"

done

Когда условие внутри цикла выполняется, то есть, когда $var1 больше 5 и меньше 10, оболочка исполняет команду continue. Это приводит к пропуску оставшихся в теле цикла команд и переходу к следующей итерации.  
  
  
*Команда continue в цикле for*

Обработка вывода, выполняемого в цикле

Данные, выводимые в цикле, можно обработать, либо перенаправив вывод, либо передав их в конвейер. Делается это с помощью добавления команд обработки вывода после инструкции done.  
  
Например, вместо того, чтобы показывать на экране то, что выводится в цикле, можно записать всё это в файл или передать ещё куда-нибудь:

#!/bin/bash

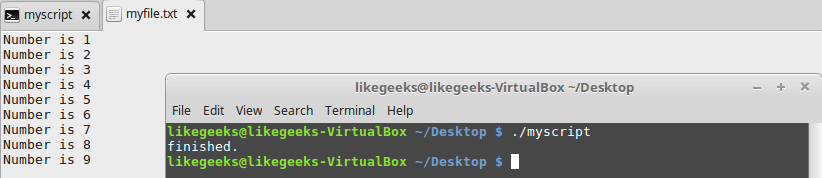
for (( a = 1; a < 10; a++ ))

do

echo "Number is $a"

done > myfile.txt

echo "finished."

Оболочка создаст файл myfile.txt и перенаправит в этот файл вывод конструкции for. Откроем файл и удостоверимся в том, что он содержит именно то, что ожидается.  
  
  
*Перенаправление вывода цикла в файл*

Пример: поиск исполняемых файлов

Давайте воспользуемся тем, что мы уже разобрали, и напишем что-нибудь полезное. Например, если надо выяснить, какие именно исполняемые файлы доступны в системе, можно просканировать все папки, записанные в переменную окружения PATH. Весь арсенал средств, который для этого нужен, у нас уже есть, надо лишь собрать всё это воедино:

#!/bin/bash

IFS=:

for folder in $PATH

do

echo "$folder:"

for file in $folder/\*

do

if [ -x $file ]

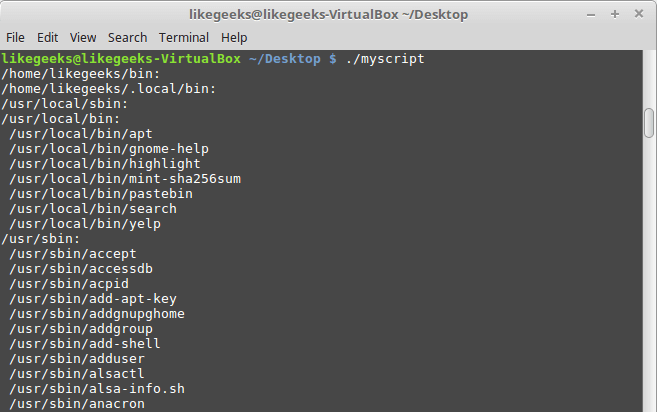
then

echo " $file"

fi

done

done

Такой вот скрипт, небольшой и несложный, позволил получить список исполняемых файлов, хранящихся в папках из PATH.  
  
  
*Поиск исполняемых файлов в папках из переменной PATH*

Итоги

Сегодня мы поговорили о циклах for и while в bash-скриптах, о том, как их запускать, как ими управлять. Теперь вы умеете обрабатывать в циклах строки с разными разделителями, знаете, как перенаправлять данные, выведенные в циклах, в файлы, как просматривать и анализировать содержимое директорий.  
  
Если предположить, что вы — разработчик bash-скриптов, который знает о них только то, что изложено в [первой части](https://habrahabr.ru/company/ruvds/blog/325522/) этого цикла статей, и в этой, второй, то вы уже вполне можете написать кое-что полезное. Впереди — третья часть, разобравшись с которой, вы узнаете, как передавать bash-скриптам параметры и ключи командной строки, и что с этим всем делать.  
  
Уважаемые читатели! В комментариях к [предыдущему материалу](https://habrahabr.ru/company/ruvds/blog/325522/) вы рассказали нам много интересного. Уверены, всё это окажет неоценимую помощь тем, кто хочет научиться программировать для bash. Но тема эта огромна, поэтому снова просим знатоков поделиться опытом, а новичков — впечатлениями.