Скрипты оболочки широко используются в мире UNIX. Они отлично подходят для ускорения повторяющихся задач и упрощения сложной логики выполнения. Они могут быть такими же простыми, как набор команд, или они могут организовывать сложные задачи. В этом уроке мы узнаем больше о языке сценариев Bash, написав примерный скрипт шаг за шагом.

Проблема с Fizz-Buzz

Например, один из лучших способов узнать о новом языке. Начнем с одного.

Проблема Fizz-Buzz очень проста. Он прославился после того, как программист, названный Имран, использовал его в качестве теста на собеседование. Оказывается, 90-99,5% кандидатов на задание по программированию просто не могут написать простейшую программу. Имран взял эту простую игру Fizz-Buzz и попросил кандидатов решить ее. Многие последовали примеру Имрана, и сегодня он является одним из наиболее часто задаваемых вопросов для задания программирования. Если вы нанимаете и нуждаетесь в способе проверке кандидатов,в 90% - это большая проблема.

Вот правила:

- Возьмите и напечатайте цифры от 1 до 100.
- Когда число делится на 3, напечатайте «Fizz» вместо номера.
- Когда оно делится на 5, вместо этого напечатайте «Buzz».
- Когда оно делится на 3 и 5, напечатайте «FizzBuzz».

Вот и все. Я уверен, что большинство из вас уже могут визуализировать

два или три оператора `if`, чтобы решить эту проблему. Поработаем с использованием языка сценариев Bash.

Шебанг

Шебанг относится к комбинации символов хэша и восклицательных знаков: `&num;!`. Программа-загрузчик будет искать shebang в первой строке скрипта и использовать указанный в нем интерпретатор. Шебанг состоит из следующего синтаксиса: `&num;! Interpreter [parameters]`. Интерпретатор - это программа, которая используется для интерпретации нашего языка. Для сценариев bash это будет `/bin/bash`. Например, если вы хотите создать скрипт в PHP и запустить его на консоли, вы, вероятно, захотите использовать `/usr/bin/php` (или путь к исполняемому файлу PHP на вашем компьютере) в качестве интерпретатора.

```
#!/usr/bin/php
<?php
phpinfo();
```

Да, это действительно сработает! Разве это не просто? Просто сначала сделайте свой файл исполняемым. Как только вы это сделаете, этот скрипт выведет вашу информацию PHP, как и следовало ожидать.

> **Совет:** Чтобы ваш скрипт работал на максимально возможном количестве систем, вы можете использовать `/bin/env` в shebang. Таким образом, вместо `/bin/bash` вы можете использовать `/bin/env bash`, который будет работать в системах, где исполняемый файл bash не находится в `/bin`.

Вывод текста

Результат скрипта будет равен, как и следовало ожидать, тому, что выводится из вашей команды. Однако, если мы явно хотим что-то написать на экране, мы можем использовать `echo`.

```
#!/bin/bash

echo "Hello World"
```

Запуск этого скрипта приведет к печати «Hello World» в консоли.

```
csaba@csaba ~ $ ./helloWorld.sh
Hello World
csaba@csaba ~ $
```

Введение переменных

Как и при любом языке программирования, при написании сценариев оболочки вы можете использовать переменные.

```
#!/bin/bash

message="Hello World"
echo $message
```

Этот код создает точно такое же сообщение «Hello World». Как вы можете видеть, чтобы присвоить значение переменной, просто напишите ее имя - исключите знак доллара перед ним. Кроме того, будьте осторожны с пробелами; между именем переменной и знаком равенства

не должно быть пробелов. Поэтому `message="Hello"` вместо `message = 'Hello'`

Если вы хотите использовать переменную, вы можете взять значение из нее так же, как в команде `echo`. Превращение `$` в имя переменной вернет свое значение.

Совет: точки с запятой не требуются в сценариях bash. Вы можете использовать их в большинстве случаев, но будьте осторожны: они могут иметь другое значение, чем то, что вы ожидаете.

Печать номеров от 1 до 100

Продолжая наш демо-проект, нам нужно циклически перебирать все числа от 1 до 100. Для этого нам нужно использовать цикл `for`.

```
#!/bin/bash

for number in {1..100}; do
    echo $number
done
```

В этом примере стоит отметить несколько новых вещей, которые, кстати, печатают все числа от 1 до 100, по одному числу за раз.

- Синтаксис for для Bash: `для параметра VARIABLE в RANGE; сделайте COMMAND.`
- В нашем примере фигурные скобки будут преобразовывать в диапазоне `1..100`. Они также используются в других контекстах, которые мы рассмотрим в ближайшее время.
- `do` и `for` - фактически две отдельные команды. Если вы хотите разместить две команды в одной строке, вам нужно как-то их отделить. Один из способов - использовать точку с запятой. В качестве альтернативы вы можете написать код без точки с запятой, перемещаясь `do` до следующей строки.

```
#!/bin/bash

for number in {1..100}
do
    echo $number
done
```

Первое решение

Теперь, когда мы знаем, как печатать все числа от 1 до 100, пришло время принять наше первое решение.

```
#!/bin/bash

for number in {1..100}; do
    if [ $((number%3)) -eq 0 ]; then
        echo "Fizz"
    else
        echo $number
    fi
done
```

В этом примере будет выводиться «Fizz» для чисел, делящихся на 3. Опять же, нам нужно иметь дело с небольшим количеством нового синтаксиса. Давайте возьмем их один за другим.

- `if..then..else..fi` - это классический синтаксис для оператора `if` в Bash. Конечно, часть `else` является необязательной, но в этом случае необходима для нашего алгоритма.
- `if COMMAND-RETURN-VALUE; then...` - `if` будет выполнено, если возвращаемое значение команды равно нулю. Да, логика в Bash основана на нулевом значении, что означает, что команды, которые успешно завершают работу , у них код равен 0. Если что-то пойдет не так, с другой стороны, будет возвращено положительное целое число. Для упрощения вещей: ничего, кроме 0, считается `false`.
- Математические выражения в Bash задаются двойными скобками. `$((number%3))` вернет оставшееся значение деления переменной, `number` на 3. Обратите внимание, что мы не использовали `$` в скобках - только перед ними.

Вам может быть интересно, где команда в нашем примере. Разве нет только скобки с нечетным выражением в ней? Ну, оказывается, что `[` на самом деле является исполняемой командой. Чтобы поиграть с этим, попробуйте выполнить следующие команды на консоли.

```
csaba@csaba ~ $ which [
```

```
/usr/bin/[
csaba@csaba ~ $ [ 0 -eq 1 ]
csaba@csaba ~ $ echo $?
1
csaba@csaba ~ $ [ 0 -eq 0 ]
csaba@csaba ~ $ echo $?
0
```

> ***Совет:*** *значение переменной команды всегда возвращается в переменную, `?` (вопросительный знак). Он перезаписывается после выполнения каждой новой команды.*

Проверка для Buzz

Пока у нас все хорошо. У нас есть «Fizz»; теперь давайте сделаем часть «Buzz».

```
#!/bin/bash

for number in {1..100}; do
    if [ $((number%3)) -eq 0 ]; then
        echo "Fizz"
    elif [ $((number%5)) -eq 0 ]; then
        echo "Buzz"
    else
        echo $number
    fi
done
```

Выше мы ввели еще одно условие делимости на 5: заявление `elif`. Это, конечно, переводит в *else if*, и будет выполняться, если команда, следующая за ней, возвращает `true` (или `0`). Как вы можете заметить, условные утверждения внутри `[]` обычно оцениваются с помощью параметров, таких как `-eq`, что означает "равно".

> *Для синтаксиса `arg1 OP arg2`, `OP` является одним из `-eq`, `-ne`, `-lt`, `-le`, `-gt` или `-ge`. Эти арифметические двоичные операторы возвращают `true`, если `arg1` равно, не равному, меньше чем, меньше или равно, больше чем или больше или равно `arg2` соответственно. `arg1` и `arg2` могут быть положительными или отрицательными целыми числами. - Руководство по Bash*

Когда вы пытаетесь сравнить строки, вы можете использовать известный символ `==` или даже один знак равенства. `! =` возвращает `true`, когда строки разные.

Но Кодекс не вполне корректен

Пока что код работает, но алгоритм неверен. Когда число делится на 3 и 5, наш алгоритм будет выводить только «Fizz». Давайте изменим наш

код, чтобы удовлетворить последнее требование FizzBuzz.

```
#!/bin/bash

for number in {1..100}; do
    output=""
    if [ $((number%3)) -eq 0 ]; then
        output="Fizz"
    fi
    if [ $((number%5)) -eq 0 ]; then
        output="${output}Buzz"
    fi
    if [ -z $output ]; then
        echo $number
    else
        echo $output;
    fi
done
```

Опять же, нам пришлось внести несколько изменений. Наиболее примечательным является введение переменной, а затем объединение «Buzz» на нее, если это необходимо. Строки в bash обычно определяются между двойными кавычками ("). Одиночные кавычки также можно использовать, но для более простой взаимосвязи лучше всего выбирать парные. В этих двойных кавычках вы можете ссылаться на переменные: `некоторый текст $variable какой-то другой текст` «заменит `$переменную` своим содержимым. Если вы хотите объединить переменные со строками без пробелов между ними, вы можете поместить имя переменной в фигурные скобки. В большинстве случаев, как и PHP, вы не обязаны это делать, но это очень помогает, когда дело доходит до читаемости кода.

Совет: Вы не можете сравнивать пустые строки. Это вернет отсутствующий параметр.

Поскольку аргументы внутри `[]` рассматриваются как параметры, для `"["` они должны отличаться от пустой строки. Таким образом, это выражение, хотя и логическое, выдает ошибку: `[$ output!= ""]`. Вот почему мы использовали `[-z $output]`, который возвращает `true`, если длина строки равна нулю.

Метод извлечения для логического выражения

Один из способов улучшить наш пример - извлечь в функции математического выражения из операторов `if`, например:

```
#!/bin/bash

function isDivisibleBy {
    return $(($1%$2))
}

for number in {1..100}; do
    output=""
    if isDivisibleBy $number 3; then
        output="Fizz"
    fi
    if isDivisibleBy $number 5; then
        output="${output}Buzz"
    fi
    if [ -z $output ]; then
        echo $number
    else
        echo $output;
    fi
done
```

Мы взяли выражения, сравнивающие остальные с нулем, и переместили их в функцию. Более того, мы устранили сравнение с нулем, потому что

нуль означает для нас истинное. Нам нужно вернуть значение из математического выражения - это очень просто!

> **Совет.** Определение функции должно предшествовать его вызову.

В Bash вы можете определить метод как ф`ункцию func_name {commands; }`. Необязательно, существует второй синтаксис для объявления функций: `func_name () {commands; }`. Итак, мы можем удалить строку, `function` и добавить «()» после ее имени. Я лично предпочитаю этот вариант, как показано в примере выше. Он более явный и напоминает традиционные языки программирования.

Вам не нужно указывать параметры для функции в Bash. Отправка параметров функции выполняется путем простого перечисления по ним после вызова функции, разделенного пробелами. Не размещайте запятые или скобки в вызове функции - это не сработает.

Полученные параметры автоматически присваиваются переменным по числу. Первый параметр равен `$1`, второй - `$2` и т. д. Специальная переменная `$0` обозначает имя файла текущего скрипта.

Давайте поиграемся с параметрами

```
01 #!/bin/bash
02
03 function exampleFunc {
04     echo $1
05     echo $0
06     IFS="X"
07     echo "$@"
08     echo "$*"
09 }
```

```
10
11  exampleFunc "one" "two" "three"
```

Этот код будет выдавать следующий результат:

```
1  csaba@csaba ~ $ ./parametersExamples.sh
2  one
3  ./parametersExamples.sh
4  one two thre
5  oneXtwoXthre
```

Давайте проанализируем источник, построчно.

- Последняя строка - вызов функции. Мы называем это тремя строковыми параметрами.
- Первая строка после shebang - это определение функции.
- Первая строка функции выводит первый параметр: «один». Пока все так просто.
- Вторая строка выводит имя файла текущего скрипта. Опять же, очень просто.
- Третья строка изменяет разделитель символов по умолчанию на букву "X". По умолчанию это " " (пробел). Вот как Bash знает, как параметры разделены.
- Четвертая строка выводит специальную переменную `$@`. Она представляет все параметры как одно слово, точно так же, как указано в вызове функции.
- В последней строке выводится другая специальная переменная `$*`. Она представляет все параметры, взятые один за другим и объединенные с первой буквой переменной IFS. Вот почему результат `oneXtwoXthre`.

Возврат строк из функций

Как я уже отмечал ранее, функции в Bash могут возвращать только целые числа. Таким образом, запись `return "строка"` будет неправильным кодом. Тем не менее, во многих ситуациях вам нужно больше нуля или одного. Мы можем реорганизовать наш пример FizzBuzz, чтобы в инструкции `for` мы просто выполнили вызов функции.

```
#!/bin/bash

function isDivisibleBy {
    return $(($1%$2))
}

function fizzOrBuzz {
    output=""
    if isDivisibleBy $1 3; then
        output="Fizz"
    fi
    if isDivisibleBy $1 5; then
        output="${output}Buzz"
    fi
    if [ -z $output ]; then
        echo $1
    else
        echo $output;
    fi
}

for number in {1..100}; do
    fizzOrBuzz $number
done
```

Ну, это первый шаг. Мы просто извлекли весь код в функцию, называемую `fizzOrBuzz`, а затем заменили `$number` на `$1`. Однако всякий вывод выполняется в функции `fizzOrBuzz`. Мы хотим вывести из цикла `for` оператор `echo`, чтобы мы могли добавить каждую строку к другой строке. Мы должны захватить вывод функции `fizzOrBuzz`.

```
#[...]
for number in {1..100}; do
    echo "-`fizzOrBuzz $number`"
    fizzBuzzer=$(fizzOrBuzz $number)
    echo "-${fizzBuzzer}"
done
```

Мы немного обновили цикл `for` (никаких других изменений). Мы повторили все два раза двумя разными способами, чтобы продемонстрировать различия между двумя решениями по одной и той же проблеме.

Первым решением для захвата вывода функции или другой команды является использование обратных ссылок. В 99% случаев это будет работать нормально. Вы можете просто ссылаться на переменную в backticks по их именам, как это было в случае с `$number`. Первые несколько строк вывода должны теперь выглядеть так:

```
csaba@csaba ~/Personal/Programming/NetTuts/The Basics of BASH Scripti
-1
-1
-2
-2
-Fizz
-Fizz
-4
-4
-Buzz
-Buzz
-Fizz
-Fizz
-7
-7
```

Как вы можете видеть, все дублируется. Тот же выход.

Для второго решения мы решили сначала присвоить возвращаемое

значение переменной. В этом назначении мы использовали `$()`, который в этом случае разворачивает скрипт, выполняет код и возвращает свой вывод.

;, && и ||

Вы помните, что мы использовали точку с запятой здесь и там? Они могут использоваться для выполнения нескольких команд, написанных на одной строке. Если вы разделите их точкой с запятой, они просто будут выполнены.

Более сложный случай - использовать `&&` между двумя командами. Да, это логично И; это означает, что вторая команда будет выполнена только в том случае, если первая возвращает `true` (она выходит с 0). Это полезно; мы можем упростить утверждения `if` в этих сокращениях:

```
#!/bin/bash

function isDivisibleBy {
    return $(($1%$2))
}

function fizzOrBuzz {
    output=""
    isDivisibleBy $1 3 && output="Fizz"
    isDivisibleBy $1 5 && output="${output}Buzz"
    if [ -z $output ]; then
        echo $1
    else
        echo $output;
    fi
}

for number in {1..100}; do
    echo "-`fizzOrBuzz $number`"
done
```

Поскольку наша функция `isDivisibleBy` возвращает правильное возвращаемое значение, мы можем затем использовать `&&` для установки нужной переменной. Что после `&&` будет исполнено, только если условие `true`. Таким же образом мы можем использовать `||` (двойной символ) в качестве логического ИЛИ. Ниже приведен краткий пример.

```
csaba@csaba ~ $ echo "bubu" || echo "bibi"
bubu
csaba@csaba ~ $ echo false || echo "bibi"
false
csaba@csaba ~ $
```

Последние мысли

Это все для этого урока! Я надеюсь, что вы взяли несколько новых советов и приемов для написания собственных сценариев Bash. Спасибо за чтение, и следите за новостями на эту тему.