Основы скриптов на bash

ОСНОВЫ СКРИПТОВ

- Скрипт или как его еще называют — сценарий, это последовательность команд, которые по очереди считывает и выполняет программа-интерпретатор, в нашем случае это программа командной строки — bash.
- Скрипт — это обычный текстовый файл, в котором перечислены обычные команды, которые мы привыкли вводить вручную, а также указана программа, которая будет их выполнять. Загрузчик, который будет выполнять скрипт не умеет работать с переменными окружения, поэтому ему нужно передать точный путь к программе, которую нужно запустить. А дальше он уже передаст ваш скрипт этой программе и начнется выполнение.

- Простейший пример скрипта для командной оболочки Bash:
  - #!/bin/bash
    echo "Hello world“
- Первая строка особая, она задает программу, которая будет выполнять команды.
- Вообще говоря, мы можем создать скрипт на любом другом языке программирования и указать нужный интерпретатор, например, на python:
  - !/usr/bin/env python
  - print("Hello world")
- Или на PHP:
  - !/usr/bin/env php
  - echo "Hello world";

- Любой bash-скрипт должен начинаться со строки:
  - #!/bin/bash
- в этой строке после #! указывается путь к bash-интерпретатору, поэтому если он у вас установлен в другом месте(где, вы можете узнать набрав whereis bash) поменяйте её на ваш путь.
- Комментарии начинаются с символа # (кроме первой строки).
- В bash переменные не имеют типа(о них речь пойдет ниже)

Выполнение скриптов

- Чтобы сделать файл исполняемым в linux выполните:
- chmod ugo+x файл_скрипта
- Теперь выполняем нашу небольшую первую программу:
- ./файл_скрипта

Использование переменных

- Переменные позволяют хранить в файле сценария информацию, например — результаты работы команд для использования их другими командами.
- Нет ничего плохого в исполнении отдельных команд без хранения результатов их работы, но возможности такого подхода весьма ограничены.
- Существуют два типа переменных, которые можно использовать в bash-скриптах:
  - Переменные среды
  - Пользовательские переменные

Некоторые переменные среды

- $# - общее количество параметров переданных скрипту
  $* - все аргументы переданные скрипту(выводятся в строку)
  $@ - тоже самое, что и предыдущий, но параметры выводятся в столбик
  $! - PID последнего запущенного в фоне процесса
  $$ - PID самого скрипта

  - #!/bin/bash
  - # display user home
  - echo "Home for the current user is: $HOME"

Пользовательские переменные

- В дополнение к переменным среды, bash-скрипты позволяют задавать и использовать в сценарии собственные переменные. Подобные переменные хранят значение до тех пор, пока не завершится выполнение сценария.
- Как и в случае с системными переменными, к пользовательским переменным можно обращаться, используя знак доллара:
- #!/bin/bash
  - # testing variables
  - grade=5
  - person="Adam"
  - echo "$person is a good boy, he is in grade $grade"

Подстановка команд

- Одна из самых полезных возможностей bash-скриптов — это возможность извлекать информацию из вывода команд и назначать её переменным, что позволяет использовать эту информацию где угодно в файле сценария.
- Сделать это можно двумя способами.
  - С помощью значка обратного апострофа «`»
    - mydir=`pwd`
  - С помощью конструкции $()
    - mydir=$(pwd)

Математические операции

- Для выполнения математических операций в файле скрипта можно использовать конструкцию вида $((a+b)):

```
#!/bin/bash
var1=$(( 5 + 5 ))
echo $var1
var2=$(( $var1 * 2 ))
echo $var2
```

Управляющая конструкция if-then

- В некоторых сценариях требуется управлять потоком исполнения команд. Например, если некое значение больше пяти, нужно выполнить одно действие, в противном случае — другое.
- Подобное применимо в очень многих ситуациях, и здесь нам поможет управляющая конструкция if-then. В наиболее простом виде она выглядит так:
  - if команда
  - then
  - команды
  - else
  - команды
  - fi

```
#!/bin/bash
user=anotherUser
if grep $user /etc/passwd
then
    echo "The user $user Exists"
else
    echo "The user $user doesn’t exist"
fi
```

- #!/bin/bash
- user=anotherUser
- if grep $user /etc/passwd
- then
- echo "The user $user Exists"
- elif ls /home | grep $user
- then
- echo "The user doesn’t exist but anyway there is a directory under /home"
- fi

Сравнение чисел

- n1 -eq n2 Возвращает истинное значение, если n1 равно n2.
- n1 -ge n2 Возвращает истинное значение, если n1 больше или равно n2.
- n1 -gt n2 Возвращает истинное значение, если n1 больше n2.
- n1 -le n2 Возвращает истинное значение, если n1 меньше или равно n2.
- n1 -lt n2 Возвращает истинное значение, если n1 меньше n2.
- n1 -ne n2 Возвращает истинное значение, если n1 не равно n2.

- #!/bin/bash
- val1=6
- if [$val1 -gt 5]
- then
- echo "The test value $val1 is greater than 5"
- else
- echo "The test value $val1 is not greater than 5"
- fi

Сравнение строк

- tr1 = str2 Проверяет строки на равенство, возвращает истину, если строки идентичны.
- str1 != str2 Возвращает истину, если строки не идентичны.
- str1 < str2 Возвращает истину, если str1 меньше, чем str2.
- str1 > str2 Возвращает истину, если str1 больше, чем str2.
- -n str1 Возвращает истину, если длина str1 больше нуля.
- -z str1 Возвращает истину, если длина str1 равна нулю.

- #!/bin/bash
- val1=text
- val2="another text"
- if [$val1 \> $val2]
- then
- echo "$val1 is greater than $val2"
- else
- echo "$val1 is less than $val2"
- fi

Проверки файлов

- -d file Проверяет, существует ли файл, и является ли он директорией.
- -e file Проверяет, существует ли файл.
- -f file Проверяет, существует ли файл, и является ли он файлом.
- -r file Проверяет, существует ли файл, и доступен ли он для чтения.
- -s file Проверяет, существует ли файл, и не является ли он пустым.
- -w file Проверяет, существует ли файл, и доступен ли он для записи.
- -x file Проверяет, существует ли файл, и является ли он исполняемым.
- file1 -nt file2 Проверяет, новее ли file1, чем file2.
- file1 -ot file2 Проверяет, старше ли file1, чем file2.
- -O file Проверяет, существует ли файл, и является ли его владельцем текущий пользователь.
- -G fileПроверяет, существует ли файл, и соответствует ли его идентификатор группы идентификатору группы текущего пользователя.

- #!/bin/bash
- mydir=/home/likegeeks
- if [-d $mydir]
- then
    - echo "The $mydir directory exists"
    - cd $mydir
    - ls
- else
    - echo "The $mydir directory does not exist"
- fi

Циклы for

- #!/bin/bash
- for var in first second third fourth fifth
- do
    - echo The $var item
- done

- #!/bin/bash
- for var in first "the second" "the third" "I'll do it"
- do
    - echo "This is: $var"
- done

- #!/bin/bash
- file="myfile"
- for var in $(cat $file)
- do
    - echo " $var"
- done

- #!/bin/bash
- file="/etc/passwd"
- IFS=$'\n'
- for var in $(cat $file)
- do
    - echo " $var"
- done

Обход файлов, содержащихся в директории

- #!/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

Циклы for в стиле C

- #!/bin/bash
- for ((i=1; i <= 10; i++))
- do
  - echo "number is $i"
- done

Цикл while

- #!/bin/bash
- var1 = 5
- while [$var1 -gt 0]
- do
  - echo $var1
  - var1=$[$var1 - 1]
- done

Обработка содержимого файла

- #!/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

Управление циклами

- #!/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

- #!/bin/bash
- for ((var1 = 1; var1 < 15; var1++))
- do
  - if [$var1 -gt 5] && [$var1 -lt 10]
  - then
    - continue
  - fi
  - echo "Iteration number: $var1"
- done

Обработка вывода, выполняемого в цикле

- #!/bin/bash
- for ((a = 1; a < 10; a++))
- do
  - echo "Number is $a"
- done > myfile.txt
- echo "finished."

Пример: поиск исполняемых файлов

- #!/bin/bash
- IFS=:
- for folder in $PATH
  - do
  - echo "$folder:"
  - for file in $folder/*
  - do
    - if [-x $file]
    - then
      - echo " $file"
    - fi
  - done
- done

Параметры скрипта

- Не всегда можно создать bash скрипт, который не зависит от ввода пользователя. В большинстве случаев нужно спросить у пользователя какое действие предпринять или какой файл использовать. При вызове скрипта мы можем передавать ему параметры. Все эти параметры доступны в виде переменных с именами в виде номеров.
- Переменная с именем 1 содержит значение первого параметра, переменная 2, второго и так далее. Этот bash скрипт выведет значение первого параметра:
  - !/bin/bash
  - echo $1

Использование параметров

- #!/bin/bash
  parametr1=$1
  script_name=$0
- echo "Вы запустили скрипт с именем $script_name и параметром $parametr1"
- echo 'Вы запустили скрипт с именем $script_name и параметром $parametr1'
- exit 0 #Выход с кодом 0 (удачное завершение работы скрипта)

- ite@ite-desktop:~$ ./test.sh qwerty
- Вы запустили скрипт с именем ./test.sh и параметром qwerty
- Вы запустили скрипт с именем $script_name и параметром $parametr1

- #!/bin/bash
- if [-n "$1"]
- then
  - echo Hello $1.
- else
  - echo "No parameters found. "
- fi

Команда shift

- Использовать команду shift в bash-скриптах следует с осторожностью, так как она, в прямом смысле слова, сдвигает значения позиционных параметров.
- Когда вы используете эту команду, она, по умолчанию, сдвигает значения позиционных параметров влево. Например, значение переменной $3 становится значением переменной $2, значение $2 переходит в $1, а то, что было до этого в $1, теряется. Обратите внимание на то, что при этом значение переменной $0, содержащей имя скрипта, не меняется.
- Воспользовавшись командой shift, рассмотрим ещё один способ перебора переданных скрипту параметров:

- #!/bin/bash
- count=1
- while [-n "$1"]
- do
  - echo "Parameter #$count = $1"
  - count=$(($count + 1))
  - shift
- done

Ключи командной строки

- #!/bin/bash
- echo
- while [-n "$1"]
- do
  - case "$1" in
    - -a) echo "Found the -a option" ;;
    - -b) echo "Found the -b option" ;;
    - -c) echo "Found the -c option" ;;
    - *) echo "$1 is not an option" ;;
  - esac
  - shift
- done

Обработка ключей со значениями

- По мере усложнения ваших скриптов, вы столкнётесь с ситуациями, когда обычных ключей уже недостаточно, а значит, нужно будет использовать ключи с некими значениями. Например, вызов сценария в котором используется подобная возможность, выглядит так:
  - ./myscript -a -b test1 -c
- Скрипт должен уметь определять, когда вместе с ключами командной строки используются дополнительные параметры:

```
#!/bin/bash
while [ -n "$1" ]
do
    case "$1" in
        -a) echo "Found the -a option";;
        -b) param="$2"
        echo "Found the -b option, with value $param"
        shift ;;
        -c) echo "Found the -c option";;
        --) shift
        break ;;
        *) echo "$1 is not an option";;
    esac
    shift
done
count=1
for param in "$@"
do
    echo "Parameter #$count: $param"
    count=$(( $count + 1 ))
done
```

Использование стандартных ключей

- -a Вывести все объекты.
- -c Произвести подсчёт.
- -d Указать директорию.
- -e Развернуть объект.
- -f Указать файл, из которого нужно прочитать данные.
- -h Вывести справку по команде.
- -i Игнорировать регистр символов.
- -l Выполнить полноформатный вывод данных.
- -n Использовать неинтерактивный (пакетный) режим.
- -o Позволяет указать файл, в который нужно перенаправить вывод.
- -q Выполнить скрипт в quiet-режиме.
- -r Обрабатывать папки и файлы рекурсивно.
- -s Выполнить скрипт в silent-режиме.
- -v Выполнить многословный вывод.
- -x Исключить объект.
- -y Ответить «yes» на все вопросы.

Получение данных от пользователя

- #!/bin/bash
- echo -n "Enter your name: "
- read name
- echo "Hello $name, welcome to my program.”

- #!/bin/bash
- read -p "Enter your name: " first last
- echo "Your data for $last, $first…"

- #!/bin/bash
- read -s -p "Enter your password: " pass
- echo "Is your password really $pass?”

- #!/bin/bash
- count=1
- cat myfile | while read line
- do
  - echo "Line $count: $line"
  - count=$(($count + 1))
- done
- echo "Finished"

Стандартные дескрипторы файлов

- Всё в Linux — это файлы, в том числе — ввод и вывод. Операционная система идентифицирует файлы с использованием дескрипторов.
- Каждому процессу позволено иметь до девяти открытых дескрипторов файлов. Оболочка bash резервирует первые три дескриптора с идентификаторами 0, 1 и 2. Вот что они означают.
- 0, STDIN — стандартный поток ввода.
- 1, STDOUT — стандартный поток вывода.
- 2, STDERR — стандартный поток ошибок.

- #!/bin/bash
- echo "This is an error" >&2
- echo "This is normal output«

- #!/bin/bash
- exec 1>outfile
- echo "This is a test of redirecting all output"
- echo "from a shell script to another file."
- echo "without having to redirect every line"

Объявление функций

- Функцию можно объявить так:
  - function Name {
  - }
- Или так:
  - function Name() {
  - }
- Функцию можно вызвать без аргументов и с аргументами.

Использование функций

- #!/bin/bash
- function myfunc {
  - echo "This is an example of using a function"
- }
- count=1
- while [$count -le 3]
- do
  - myfunc
  - count=$(($count + 1))
- done
- echo "This is the end of the loop"
- myfunc
- echo "End of the script"

Использование команды return

- #!/bin/bash
- function myfunc {
  - read -p "Enter a value: " value
  - echo "adding value"
  - return $(($value + 10))
- }
- myfunc
- echo "The new value is $?«

- Учтите, что максимальное число, которое может вернуть команда return — 255.
- Если функция должна возвращать большее число или строку, понадобится другой подход.

Запись вывода функции в переменную

- #!/bin/bash
- function myfunc {
  - read -p "Enter a value: " value
  - echo $(($value + 10))
- }
- result=$(myfunc)
- echo "The value is $result"

- #!/bin/bash
- function addnum {
  - if [$# -eq 0] || [$# -gt 2]
  - then
    - echo -1
  - elif [$# -eq 1]
  - then
    - echo $(($1 + $1))
  - else
    - echo $(($1 + $2))
  - fi
- }

- echo -n "Adding 10 and 15: "
- value=$(addnum 10 15)
- echo $value
- echo -n "Adding one number: "
- value=$(addnum 10)
- echo $value
- echo -n "Adding no numbers: "
- value=$(addnum)
- echo $value
- echo -n "Adding three numbers: "
- value=$(addnum 10 15 20)
- echo $value

Создание и использование библиотек

- файл myfuncs, который содержит следующее:

- function addnum {
- echo $(($1 + $2))
- }

- #!/bin/bash
- . ./myfuncs
- result=$(addnum 10 20)
- echo "The result is: $result"