Сценарий командной оболочки (скрипт) – текстовый файл, содержащий программу на скриптовом языке.

Программа исполняется командной оболочкой в режиме интерпретатора, **файл сценария должен иметь разрешение на исполнение**.

В начале файла сценария необходимо указать программу, которая будет исполнять скрипт (эта программа будет вызвана с указанием файла сценария в качестве аргумента командной строки). Конструкция языка, определяющая программу для исполнения, называется *шебанг*

#!/bin/bash

Сценарий позволяет определять переменные.

Определение переменной можно соединить с присваиванием ей значения, например

а=5

***Важно!*** Пробелы вокруг символа ‘=’ не допускаются!

Для получения значения переменной необходимо использовать префикс ‘$’, например $a. Для исключения неопределенности можно брать имя переменной в фигурные скобки, например ${a}

Параметры командной строки, переданные в сценарий, доступны в нем в виде значений переменных $1, $2, … $9, … где число обозначает номер параметра командной строки

Если параметров более 9, то нужно брать число в фигурные скобки, например, ${18}

Переменная $# содержит число параметров, переданных в скрипт.

Можно использовать индексирую переменную для доступа к параметрам, тогда синтаксис будет такой:

i=3  
${!i}

Команда echo позволяет выводить на экран сообщения и значения переменных, например

echo “Значение переменной a=$a”

Команда (утилита) test

Переменная $? содержит статус завершения предыдущей команды или программы.

Например, можно написать в командной строке

ls <имя существующего файла>  
echo $?

Должен быть выведен 0

ls <имя НЕ существующего файла>  
echo $?

Должен быть выведен НЕ 0

Есть специальная команда (утилита) командной строки test (Можно набрать команду which test, чтобы увидеть, какой именно файл реализует утилиту test)

Команда test позволяет вычислять условия и возвращает 0 в переменной $? если условие выполняется. Логическое выражение указывается как параметры командной строки утилиты test, например

test 3 -gt 2  
echo $?

Вычисление выражения 3 больше 2 должно дать 0 (истина).

Некоторые условия:

1. Сравнение чисел  
   A -gt B – число A больше числа B  
   A -ge B – число A больше или равно числа B  
   A -eq B – число A равно числу B  
   A -ne B – число A не равно числу B  
   A -le B – число A меньше или равно числа B  
   A -lt B – число A меньше числа B
2. Сравнение строк по алфавиту  
   Знаки > < = или !=  
   Не забывать экранировать! “строка 1” \> “строка 2”
3. Проверка строк  
   -n “строка” – строка не пустая  
   -z “строка” – строка пустая
4. Проверка файлов  
   -e <файл> – файл существует  
   -s <файл> – файл существует и имеет ненулевую длину  
   -x <файл> – файл существует и является исполняемым  
   -r <файл> – файл существует и доступен для чтения  
   -d <файл> – файл существует и является каталогом  
   -L <файл> – файл существует и является символической ссылкой

Подробнее см. man test

Команда if

if <команда>  
then  
# действия если команда выполнена правильно ($? = 0)  
fi

if <команда>  
then  
# действия если команда выполнена правильно ($? = 0)  
else  
# действия если команда выполнена неправильно ($? != 0)  
fi

if <команда>  
then  
# действия если команда выполнена правильно ($? = 0)  
elif <команда 2>  
then  
# действия если команда 2 выполнена правильно ($? = 0)  
elif <команда N>  
then  
…  
else  
# действия если последняя команда выполнена неправильно ($? != 0)  
fi

В качестве команды в if чаще всего используют test

if test -e “$1”  
then  
 if test ! -s “$1”  
 then  
 rm “$1”  
 fi  
fi

Если файл, имя которого передано в первом параметре командной строки при вызове сценария, существует и имеет нулевую длину, то удалить его

Более короткая запись

if [ -e “$1” ]  
then  
 if [ ! -s “$1” ]  
 then  
 rm “$1”  
 fi  
fi

Т.е. вместо слова test можно использовать квадратные скобки.

***Важно!*** Пробелы вокруг квадратных скобок обязательны!

Утилита expr

Позволяет вычислять значения выражений, например

expr 3 + 2

Подробнее см. man expr

Подстановка команд

В сценарии можно получить доступ к выводу команды на экран (подстановка команды). Синтаксис:

$(команда)

Например

s=$(expr 2 + 3)

Сумма 2 + 3 будет сохранена в переменной s

echo “2 + 3 = $(expr 2 + 3)”

Выведет на экран 2 + 3 = 5

Цикл while

while <команда>  
do  
…  
done

Например, вывод чисел от 1 до 10

i=1  
while [ $i -le 10 ]  
do  
 echo $i  
 i=$(expr $i + 1)  
done

Оператор выбора case

case $variable in  
 pattern-1)  
 commands  
 ;;  
 pattern-2)  
 commands  
 ;;  
…  
 pattern-N)  
 commands  
 ;;  
 \*) # по умолчанию, если ни один шаблон не подошел  
 commands  
 ;;  
esac

Команда shift <k>

Сдвигает влево на k позиций значения переменных ${n}, где n - число (переменные, содержащие значения параметров сценария)

Например, после выполнения команды shift 1 значения переменных меняется следующим образом

$1 $2 $3 $4 …

теряется

$1 $2 $3 …

После выполнения команды shift 2 значения переменных меняется следующим образом

$1 $2 $3 $4 …

теряются

$1 $2 …

Таким образом, при большом числе аргументов можно анализировать аргументы в нескольких первых переменных $n, затем сделать вызов shift на необходимое число позиций и анализировать следующий блок параметров.

Пример: передача параметров в сценарий. Напишем скрипт example-1, который выводит на экран значения переданных ему опций a и b и остаток командной строки после опций

Синтаксис вызова скрипта

example-1 -a <значение> -b <значение> -- <текст>

while [ -n “$1” ]  
do  
 case “$1” in  
 -a)  
 echo “a = $2”  
 shift 2  
 ;;  
 -b)  
 echo “b = $2”  
 shift 2  
 ;;  
 --)  
 shift  
 break  
 ;;  
 \*)  
 echo “Invalid argument”  
 echo “usage $0 -a <text> -b <text> -- <text>”  
 exit 1  
 ;;  
 esac  
done  
  
echo $1

Команда (утилита) getopts

Выполняет анализ аргументов командной строки по указанному шаблону. Синтаксис вызова

getopts optstring variable

Команда getopts просматривает список аргументов в поисках опций, указанных в optstring и заносит первую найденную опцию в переменную variable. Если будет встречена опция, которой нет в optstring, то getopts установит переменную variable в значение ?

При последующих вызовах поиск опций продолжается. Когда будет просмотрена вся командная строка getopts вернет ненулевое значение в $?

Если опция содержит значение параметра, то в строке optstring после имени опции нужно поставить двоеточие. Значение параметра getopts занесет в переменную OPTARG

Далее приводится реализация примера example-1 на основе getopts

while getopts a:b: c  
do  
 case "$c" in  
 a)  
 echo “a = $OPTARG”  
 ;;  
 b)  
 echo “b = $OPTARG”  
 ;;  
 \?)  
 echo "Usage: $0 -n <n> -t <t> -- <text>"  
 exit 1  
 ;;  
 esac  
done  
shift $(expr $OPTIND - 1)  
  
echo $1

Массивы

bash поддерживает массивы. Индексами массива являются целые числа, включая 0, но они не обязательно должны идти подряд без пропусков. Например, можно определить массив из 2 элементов с индексами 5 и 7:

arr[5]=1  
arr[7]=2

Можно сразу инициализировать все элементы массива, если перечислить их в круглых скобках:

arr=(A B C D)

Определяет массив из 4 элементов с индексами 0, 1, 2 и 3 (arr[0] равен A и т.д.)

При необходимости, индексы можно указать явно, например

arr=(A B [5]=C [9]=D)

Для получения значения элемента массива используется следующий синтаксис

${arr[index]}

В качестве индексов можно использовать значения переменных, например

i=5  
echo ${arr[$i]}

Чтобы удалить ненужный массив (или любую переменную) можно использовать команду unset

unset arr

Также определены следующие специальные конструкции для работы с массивами:

${arr[\*]} – выбрать все элементы в массива одной строкой

${arr[@]} – выбрать все элементы в массива отдельной строкой для каждого элемента, даже если написать в кавычках “${arr[@]}”

${!arr[\*]} – все индексы в массиве

${#arr[\*]} – количество элементов в массиве

${#arr[n]} – длина элемента n (если рассматривать его как строку)

Рассмотрим пример:

arr=(1 2 3)  
arr2=(”${arr[\*]}”)  
arr3=(”${arr[@]}”)  
echo ${#arr2[\*]} # выводит 1  
echo ${#arr3[\*]} # выводит 3

Инициализация массива параметрами командной строки

arr=(“$@”)

Цикл for по элементам множества

Синтаксис

for <переменная> in <множество значений>  
do  
…  
done

Пример: вывод на экран чисел 1, 3, 7 и 4

for i in 1 3 7 4  
do  
 echo $i

done

В качестве множества может быть использован массив, тогда будет выполнен перебор элементов массива

arr=(1 3 7 4)  
for i in ${arr[\*])  
do  
 echo $i

done

Если в качестве множителя передать каталог строку вида <каталог>/\*, то будут перечислены файлы из указанного каталога (кроме скрытых). Например, вывод имен файлов из текущего каталога

for file in ./\*  
do  
 echo $file  
done

Также можно использовать в качестве множителя любой текст с разделителями (по умолчанию – пробел), в том числе подстановка команды.

Переменная IFS

IFS – Internal Field Separator – Внутренний разделитель полей. Переменная IFS содержит символ, который трактуется как разделитель элементов в перечислении (множестве)

Рассмотрим пример: скрипт self-print, который выводит на экран себя.

#!/bin/bash  
  
for s in $(cat self-print)  
do  
 echo $l  
done

Он выводит на экран следующий текст

#!/bin/bash  
for  
i  
in  
$(cat  
self-print)  
do  
echo  
$i  
done

Чтобы получить построчный вывод файла просто удалим разделитель полей и строки не будут разделяться по пробелам

#!/bin/bash  
  
IFS=  
for s in $(cat self-print)  
do  
 echo $l  
done

Функции

Сценарии командной оболочки позволяют определять функции.

<имя функции>() {  
…  
}

вызов функции:

<имя функции> [<аргумент 1> [<аргумент 2> […]]]

т.е. в функцию можно передавать аргументы, которые будут доступны в функции по номерам аргументов (как и параметры самого сценария): $1, $2, …

В функции можно использовать команду return, тогда результат работы функции будет доступен через $?.

**Важно!** return не предназначен для возвращения результата работы функции, только для указания статуса завершения (0 – успех, не 0 – ошибка). **Возвращаемое значение может быть в диапазоне [0,255]**

Для возвращения результата вычисления можно установить какую-либо переменную или использовать echo с последующей подстановкой команды, например

sum() {  
 echo $(expr $1 + $2)  
}

a=2  
b=3  
c=$(sum $a $b)  
echo $c

Возможен рекурсивный вызов функции, например, напишем скрипт для вычисления факториала

Вызов: factorial <число>

#!/bin/bash  
  
fact() {  
 if [ $1 -eq 1 ]  
 then  
 echo 1  
 else  
 echo $(expr $1 \\* $(fact $(expr $1 - 1)))  
 fi  
}  
  
fact $1