Администрирование операционной системы Linux (6 семестр)

Лекция 4. Сценарии командной оболочки bash.

Старший преподаватель, к.т.н. Митричев Иван Игоревич

Москва 2020

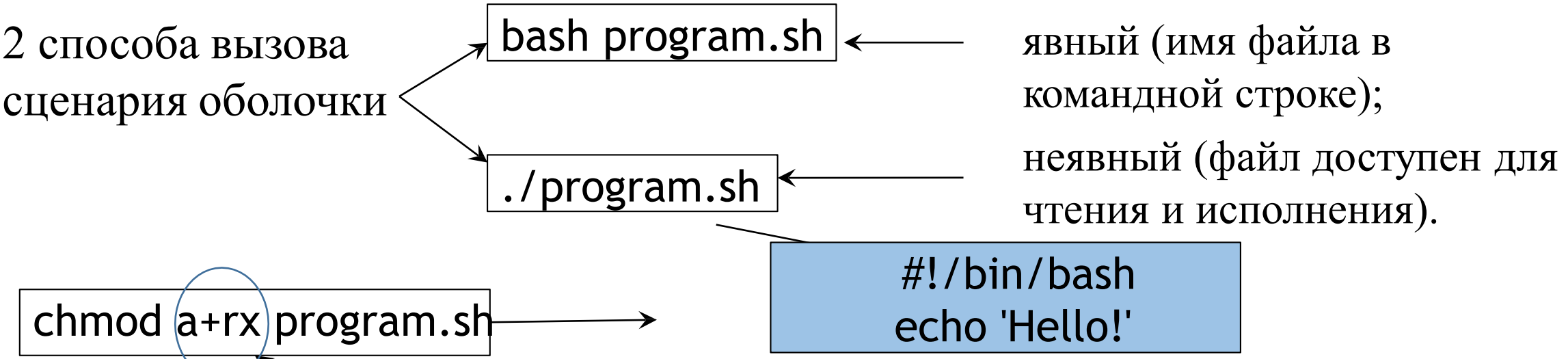
План лекции

- Сценарии оболочки.
- Переменные в bash.
- Интерактивная установка значений переменных.
- Вызов сценария.
- Экранирование переменных.
- Массивы.
- Позиционные и специальные параметры.
- Установка аргументов.
- Команда test.
- Сравнение файлов, строк, чисел.
- Примеры скриптов.

Сценарии оболочки

Сценарий командной оболочки (скрипт) – программа, выполняемая командной оболочкой. Состоит из отдельных команд, объединенных общей целью. Скрипты позволяют автоматизировать часто повторяемые действия, сэкономить время. Стандартное расширение **.sh**

2 способа вызова сценария оболочки



сделать файл программы доступным для чтения и исполнения для всех пользователей.

Опции bash

Опции -v и -x bash важны при отладке сценариев.

```
$ bash -v myscr1.sh
#!/bin/bash
echo 'Privet!'
Privet!
$ bash -x myscr1.sh
+ echo 'Privet!'
Privet!
$ bash myscr1.sh
Privet!
```

-v – показывать команды скрипта перед интерпретацией.

-x – показывать результаты интерпретации команд.

```
#/bin/bash
set -x # activate debugging from here
w
set +x # stop debugging from here
```

←

- Выводить информацию только для команды **w**

```
set -n; ./script.sh
```

←

- Проверка скрипта на ошибки без исполнения

http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_03.html

Вызов сценария из другого сценария

Вызов сценария g.sh из сценария program1.sh

Явный вызов – g.sh
выполняется в своей (дочерней, порожденной) оболочке (subshell)

```
#!/bin/bash

./g.sh
```

program.sh

```
> ps a | grep bash
22949 pts/0    S+     0:00 bash g.sh
22950 pts/0    S+     0:00 bash g.sh
```

Inline-подстановка текста –
g.sh видит все переменные текущей оболочки

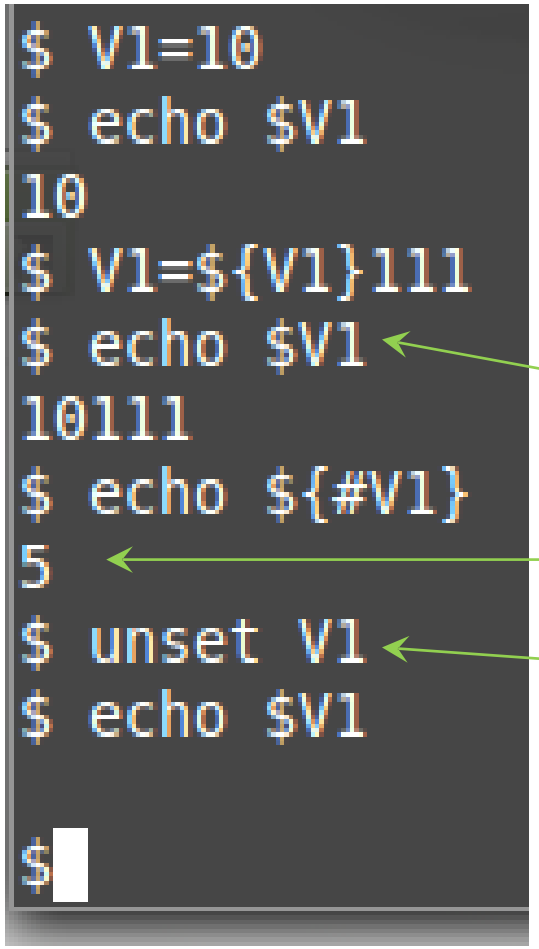
```
#!/bin/bash
. g.sh
```

program.sh

Оператор inline-подстановки

```
22803 pts/0    S+     0:00 bash g.sh
```

Работа с переменными



При обращении к несуществующей переменной будет выдана в качестве результата пустая строка. Чтобы предотвратить использование неинициализированных переменных и возможные ошибки используйте команду

set –u

- для обращения к значению переменной указывайте перед именем **$**
- количество символов
- для уничтожения переменной – **unset**, указав в качестве ее аргумента имя переменной.

Переменные не типизированы! Рассматриваются как строки, над которыми можно выполнять арифметические операции, если они содержат только числа.

Интерактивная установка значений переменных

```
echo -n 'Введите строку:' ; read var1; echo $var
```

- считать значение из стандартного потока ввода;

```
read var1 var2 var3
```

- ввести значения для трех переменных. Если введено больше значений, все значения с третьего и далее будут сохранены в var3. Если введено меньше значений, оставшиеся переменные будут пустыми.

Полезные встроенные команды

printf – печать строки с форматированием

```
declare -r PI=3.14159265358979     # переменная только для чтения
printf "Pi с 2 десятичными знаками = %1.2f" $PI
```

eval – выполняет соединение всех аргументов в строку и ее выполнение. Никогда не используйте eval в том месте, где как аргумент может быть подставлен чужой код/ввод пользователя (небезопасно)

```
c1="ps"
c2="ax"
bash$ eval "$c1" "$c2"
```

Переменные и значения

В имени переменной можно использовать только символы английского алфавита, цифры и подчеркивание.

```
$ f1=str
$ echo $f1
str
$ echo ${f1}1
str1
$ echo "$f1"1
str1
```

- экранирование имени переменной;

- установка значения по умолчанию.

```
$ echo ${V2:=12345}
12345
$ echo ${f1:=12345}
str
```

Если переменная уже определена, то установка значения по умолчанию не повлияет на ее значение.

Экранирование

Способы экранирования:

'пришло 100$ на счет' – экранирует все, кроме ' ‘

"двойные кавычки" – экранирует все, кроме " ", $, \, `

\ – экранирует все, кроме \n

`var='Hi Dude!'` ←

`var=Hi\ Dude` ←

- переменная содержит строку с пробелами, строка экранирована с помощью одиночных или двойных кавычек;
- перед пробелами – экранирующий символ обратной косой черты «\».

Типизирование переменных

В большинстве случаев переменные в сценариях bash интерпретируются как строки. Но можно явно указать, как оболочке рассматривать переменную.

declare ← • встроенная функция для объявления типизированных переменных.

Опции declare: -a – массив;
-i – целое число;
-f – функция (без аргументов список функций);
-r – только для чтения;
-x – переменная на экспорт.

```
$ declare -i val
$ val=3
$ echo $val
3
$ val=abc
$ echo $val
0
```

сценарий интерпретирует переменную “val" как целое число;
строка “abc“ интерпретируется как целое число.

```
$ declare -r f1
f1=56
```

← `bash: f1: доступная только на чтение переменная`

Массивы

Массивы объявляются путем перечисления или присвоения элементов. Можно пропускать некоторые элементы, и они останутся неинициализированными.

```
europe=(sweden france england spain)
```

```
europe[0]=sweden
europe[1]=france
europe[2]=england
europe[3]=spain
```

- объявление массива;

```
${europe[i]}
```

- доступ к элементу массива i;

```
@
```

- специальный индекс.

```
~$ area[2]=10
~$ echo -n "area[7] = ${area[2]}"
area[7] = 10~$
```

- Как заставить оболочку выполнить указанное в тексте слева присвоение?

Позиционные параметры

Позиционные параметры позволяют получить имя сценария и переданные ему в командной строке параметры:

$0 – имя команды;

$1, ..., $9 – значения девяти аргументов командной строки.

Если используется более 9 параметров, нужно использовать ${10} для обращения к десятому параметру.

Для получения всех параметров скрипта shell удобно использовать цикл for в комбинации с shift, или $@, или $* (следующие слайды)

Специальные параметры

$*, $@ – строка, составленная из значений всех аргументов командной строки;
"$@" – символы в двойных кавычках будут считываться как один параметр;
"$*" - все символы будут считаны как один параметр, где стандартный разделитель можно заменить на первый символ из переменной IFS
$# – количество аргументов командной строки;
$? – код возврата предыдущей команды по POSIX; (или 128 + код сигнала, который привел команду к остановке выполнения)
$$ – PID оболочки.

```
$ cat comline.sh
#!/bin/bash  echo
$*
$ ./comline.sh 11 aa 22
bbb

11 aa 22 bbb
```

← • сценарий выводит все аргументы командной строки.

Сдвиг позиционных параметров

shift <n>

- сдвигает позиционные параметры на n (по умолчанию на 1 вправо) $1 принимает значение от $2, $2 от $3 и т.д. **Сдвиг необратим!** Общее число параметров $# изменится в соответствии с оставшимся количеством параметров.

set <value1> <value2> <value3> ...

- при вызове внутри скрипта устанавливает значения позиционных параметров.

$1 $2 $3

set $1 $2 newarg $4

- указаны позиционные параметры, не подлежащие изменению. Изменить только третий параметр.

```
$ cat param.sh
#!/bin/bash
echo $1
echo $2
echo shifted:
shift
echo $1 echo $2
```

```
$ ./param.sh first second third
first second
shifted:
second third
```

Команда test

test ← проверка выполнения условия, 0 - условие истинно, 1 - условие ложно.

Опции для файлов:

-e – файл существует; -f – обычный файл; -d – директория;

-h, -L – является символической ссылкой; -s – не пустой;

-r – доступен для чтения; -w – доступен для записи;

-x – доступен для исполнения; -N – был изменен со времени последнего прочтения

Эквивалентная запись команды test: [ ]

```
$ [ -e /etc/passwd ]
$ echo $?  0
```

Пример:

```
$ test -e /etc/passwd
$ echo $?
0
$ test -e not_existent_file
$ echo $?
1
```

- нулевой код возврата, т.к. файл /etc/passwd существует;
- условие ложно

test для проверок двух файлов

`[ file1 -nt file2 ]` ← • первый файл модифицирован позднее;

`[ file1 -ot file2 ]` ← • первый файл модифицирован раньше;

`[ file1 -ef file2 ]` ← • существует жесткая связь между файлами.

`[ -o <опция> ]` ← • проверка установки опции оболочки.

Пример:

```
$ set -o noclobber
$ [ -o noclobber ]
$ echo $?  0
```

```
$ set +o noclobber
$ [ -o noclobber ]
$ echo $?  1
```

Сравнение строк

`[ $str1 = $str2 ]` ← • проверка на совпадение строк;

`[ $str1 != $str2 ]` ← • проверка на несовпадение строк;

`[ $str1 \< $str2 ]` ← • при сортировке str1 раньше, чем str2;

`[ -z $str ]` ← • истина, если длина строки нулевая;

`[ -n $str ]` ← • истина, если длина строки ненулевая.

Пример:

```
$ [ abc = abc ]
$ echo $?
0
$ [ abc = abs ]
$ echo $?
1
```

• сравнение строк (для простых строк - можно без кавычек)

Сравнение целых чисел

-eq – равенство;
-lt – меньше;
-gt – больше;

-ne – неравенство;
-le – меньше или равно;
-ge – больше или равно.

Пример:

```
if [ "$name" -eq 5 ];
then
...
```

```
$ [ 1 -lt 2 ]
$ echo $?  0
$ [ 1 -eq 2 ]
$ echo $?
1
```

Сравнение используется в управляющих операторах bash: if, while, until, for и case.

```
if [ "$var1“ -gt "$var2“  -a "$var1“ -gt "$var3" ]
```

- логическое «И»

```
if [ "$var1“ -gt "$var2“  -o "$var1“ -gt "$var3" ]
```

- логическое «ИЛИ»

Пример скрипта для сравнения строк

Пример скрипта для сравнения двух строк

```
#!/bin/bash
	S1='string'
	S2='String'
	if [ $S1=$S2 ];
	then
		echo "S1('$S1') is not equal to S2('$S2')"
	fi
	if [ $S1=$S1 ];
	then
		echo "S1('$S1') is equal to S1('$S1')"
	fi
```

[G., Mike. BASH Programming - Introduction HOW-TO [Электронный документ] Режим доступа: http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-11.html Дата обращения 16.03.2018]

Порядок и логика разработки скрипта

Перед написанием полезно продумать логическую структуру программы.

Пример: вычисление положительной степени целого числа.

1.Введенная степень и число являются целыми, а степень положительной?
- Если "да":
  Создать временную переменную и поместить в нее число.
- Если "нет", перейдите к шагу 6.

1.Умножить временную переменную на число, и сохранить результат во временную переменную.
2.Вычесть единицу из степени.
3.Степень больше 1?
- Если "да": перейдите к шагу 2.
- Если "нет": перейдите к шагу 5.

1.Выведите число на экран.
2.Выход