

Лабораторная №8. Основы работы со сценариями bash

1. Цель и задачи работы

Цель работы:

Научиться создавать и использовать сценарии командной оболочки bash для автоматизации задач в Linux

Задачи работы:

1. Создать простой сценарий bash
2. Создать сценарий с конструкцией if-then-else
3. Создать составной сценарий, выполняющий несколько действий

2. Задание

В данной лабораторной работе вам необходимо создать несколько файлов сценариев (скриптов) командной оболочки bash.

3. Теоретические сведения

bash – это самая популярная командная оболочка (командный интерпретатор) Linux. Основное предназначение bash – выполнение команд, введенных пользователем. Пользователь вводит программу, bash ищет программу, соответствующую команде, в каталогах, указанных в переменные окружения PATH. Если такая программа найдена, то bash запускает ей и передаёт ей введенные пользователем параметры. В противном случае выводится сообщение о невозможности выполнения команды.

Сценарий (скрипт) оболочки Bash представляет собой исполняемый текстовый файл с набором команд, которые необходимо выполнить в определенном порядке и/или условиями. Bash-скрипт призван избавить пользователей Linux от ввода множества различных команд для решения частых задач.

Не копируйте чужой код

Копирование чужого кода на свой компьютер/сервер опасно. Ранее существовал «патч Бармина», представляющий из себя команду `rm -rf /*`. Ее очень любили давать новичкам Linux на некоторых конференциях в качестве универсального средства от всех проблем. Суть команды — рекурсивное удаление всех каталогов внутри корневого каталога, т. е. всех системных и пользовательских файлов. Сейчас эта команда не работает во всех актуальных версиях Linux, но раньше она служила злой шуткой и наказанием тем, кто копировал чужие скрипты на свои серверы и выполнял их. Способов навредить серверу/компьютеру все еще достаточно, но они не столь очевидны.

4. Методические указания

4.1 Создание файла сценария

Включите виртуальную машину. После загрузки и входа в операционную систему появится окно терминала.

В домашней директории пользователя создайте директорию с названием **lab8** и перейдите в эту директорию командой **cd**.

Запуск Hello, World!

Первая программа, которую обычно пишут программисты это Hello, World! — простой вывод этой строки. За вывод строки в консоль отвечает команда **echo**. Прямо в консоли вы можете напечатать `echo "Hello, World!"` и получить соответствующий вывод.

Сделаем это программой. Команда **touch helloworld.sh** создаст файл **helloworld.sh**. Команда **nano helloworld.sh** откроет этот файл для редактирования. Заполним файл нашей программой:

```
#!/bin/bash
echo "Hello, World!"
```

Для выхода с сохранением из nano надо нажать CTRL + O для сохранения (после чего нажать enter для перезаписи текущего открытого файла), а потом CTRL + X для выхода. Можно выходить без сохранения, при этом он спросит, точно ли вы хотите выйти без сохранения. Если да, надо нажать N для выхода без сохранения. Если вы нажмете Y, он спросит куда сохранить измененный файл, можно нажать enter для перезаписи редактируемого файла.

Первой строкой идет **#!/bin/bash** — фактически это указание на местоположение интерпретатора. Чтобы при запуске скрипта не требовалось указывать отдельно интерпретатор. Убедиться, что ваш bash интерпретатор лежит по этому пути можно через команду `which bash`:

```
root@geneviev:~ # which bash
/usr/bin/bash
```

Запустить ваш скрипт/команду можно двумя способами.

Способ №1: `bash helloworld.sh`. Вы вызываете интерпретатор и в аргументе передаете ему имя файла для исполнения.

Способ №2: Сначала надо разрешить системе исполнять скрипт: **`chmod +x helloworld.sh`**. Эта команда делает файл исполняемым. Теперь вы можете запустить его как любой бинарный файл в linux: **`./helloworld.sh`**.

Первая программа готова, она просто выводит строку в консоль.

4.2 Использование переменных

Переменные в Bash — это именованные области памяти, в которых можно хранить строки или числа для дальнейшего использования в сценариях. В отличие от многих языков программирования, в Bash нет строгой типизации: любая переменная по умолчанию считается строкой, но в определенных контекстах может обрабатываться как число.

1. Основные правила работы с переменными

При создании переменной важно соблюдать синтаксис. Самая частая ошибка — лишние пробелы.

Правильно: `NAME="Ivan"`

Ошибка: `NAME = "Ivan"` (система подумает, что NAME — это команда).

2. Обращение к значению

Чтобы получить значение переменной, перед её именем ставится знак доллара \$.

```
USER_NAME="Student"
echo $USER_NAME
```

Иногда лучше использовать фигурные скобки `${}`, чтобы избежать путаницы, если за переменной сразу следует другой текст:

```
FILE_NAME="report"
mv $FILE_NAME ${FILE_NAME}_old.txt # Без скобок Bash искал бы переменную
FILE_NAME_old
```

4.3 Кавычки: Одинарные или Двойные

Двойные кавычки (" "): Разрешают интерпретацию. Внутри них работают переменные (\$VAR) и подстановка команд \$(...).

Одинарные кавычки (' '): "Замораживают" текст. Всё, что внутри них, воспринимается как обычные символы (литералы).

```
PRICE=100

echo "Цена: $PRICE" # Выведет: Цена: 100

echo 'Цена: $PRICE' # Выведет: Цена: $PRICE
```

4.4 Подстановка команд \$(...)

Чтобы сохранить результат работы команды в переменную, используется конструкция \$(команда).

Пример: MY_DIR=\$(pwd) — теперь в переменной лежит путь к вашей текущей папке.

```
echo $MY_DIR
```

Арифметика \$((...))

Bash считает только внутри двойных скобок:

SUM=\$((5 + 10)) — верно.

SUM=5+10 — неверно (будет просто строка "5+10").

4.5 Управляющая конструкция if-then-else

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
команда
fi
```

Рабочий пример:

```
#!/bin/bash
if pwd
then
echo "It works"
fi
```

В данном случае, если выполнение команды **pwd** завершится успешно, в консоль будет выведен текст «it works».

Рассмотрим более сложный сценарий. Скажем, надо найти некоего пользователя в /etc/passwd, и если найти его удалось, сообщить о том, что он существует.

```
#!/bin/bash
user=nobody
if grep -q $user /etc/passwd
then
echo "The user $user Exists"
fi
```

Здесь использована команда **grep** для поиска пользователя в файле `/etc/passwd`, `-q`. В этом примере, если пользователь найден, скрипт выведет соответствующее сообщение. А если найти пользователя не удалось? В данном случае скрипт просто завершит выполнение, ничего нам не сообщив. Хотелось бы, чтобы он сказал нам и об этом, поэтому усовершенствуем код скрипта.

Для того, чтобы программа смогла сообщить и о результатах успешного поиска, и о неудаче, воспользуемся конструкцией **if-then-else**. Вот как она устроена:

```
if команда
then
команды
else
команды
fi
```

Если первая команда возвратит ноль, что означает её успешное выполнение, условие окажется истинным и выполнение **не** пойдёт по ветке **else**. В противном случае, если будет возвращено что-то, отличающееся от нуля, что будет означать неудачу, или ложный результат, будут выполнены команды, расположенные после **else**.

Что если надо проверить не одно условие, а несколько? Например, если нужный пользователь найден, надо вывести одно сообщение, если выполняется ещё какое-то условие — ещё одно сообщение, и так далее. В подобной ситуации нам помогут вложенные условия. Выглядит это так:

```
if команда1
then
команды
elif команда2
then
команды
fi
```

Если первая команда вернёт ноль, что говорит о её успешном выполнении, выполнятся команды в первом блоке **then**, иначе, если первое условие окажется ложным, и, если вторая команда вернёт ноль, выполнится второй блок кода.

Создайте файл с названием **find_or_create_user.sh**. первые две строки файла должны выглядеть следующим образом:

```
#!/bin/bash
user=<какое-то имя пользователя>
```

Задание

Дополните скрипт таким образом, чтобы он проверял, есть ли в системе пользователь с указанным именем. Если пользователь существует, скрипт должен вывести сообщение об этом. Если такого пользователя нет, скрипт должен вывести соответствующее сообщение, создать пользователя и задать ему пароль. Для создания пользователя можно использовать команду **useradd**, а для назначения пароля – команду **passwd** ИЛИ параметр **-p** команды **useradd**. Для того, чтобы итоговый скрипт отработал корректно, необходимо запускать его с привилегиями суперпользователя (т.е. перед командой запуска скрипта поставить **sudo**). Также не забудьте сделать файл скрипта исполняемым.

4.6 Сравнение чисел

В скриптах можно сравнивать числовые значения. Ниже приведён список соответствующих команд.

```
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.
```

В качестве примера опробуем один из операторов сравнения. Обратите внимание на то, что выражение заключено в квадратные скобки.

```
#!/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
```

Проверьте, что выведет такой скрипт.

4.7 Проверки файлов

Нижеприведённые команды позволяют проверять различные условия, касающиеся файлов. Вот список этих команд.

```
-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 file Проверяет, существует ли файл, и соответствует ли его
идентификатор группы идентификатору группы текущего пользователя.
```

Пример использования одной из команд в скрипте:

```
#!/bin/bash
mydir=/var/log
if [ -d $mydir ]# пробелы должны стоять перед и после
then
echo "The $mydir directory exists"
cd $mydir
ls
else
echo "The $mydir directory does not exist"
fi
```

Этот скрипт выведет содержимое директории при условии, что директория она существует. Запустите скрипт, убедитесь, что он сработал корректно.

Задание

Измените ваш скрипт **find_or_create_user.sh** таким образом, чтобы после создания пользователя проверялось наличие в директории `/home` поддиректории пользователя (название директории должно совпадать с именем пользователя). В случае, если такой директории нет, скрипт должен вывести соответствующее сообщение, создать директорию и поменять для неё владельца и группу на вновь созданного пользователя.

Для смены владельца используйте команду **chown**.

Синтаксис команды для нашего случая **chown** следующий:

```
chown username:group FILE  
chown "$user:$user" "$my_dir"
```

Для нового пользователя `username` и `group` совпадают. В качестве `FILE` используется путь до директории, владельца которой необходимо изменить. `chown` — команда для смены владельца (и, при необходимости, группы) файла или директории.

`"$user:$user"` — это аргумент, указывающий нового владельца и новую группу. Всё, что до двоеточия (`$user`) — имя нового владельца (пользователя). Всё, что после двоеточия (`$user`) — имя новой группы.

В данном случае и владелец, и группа имеют одно и то же имя — имя созданного пользователя (например, `test1:test1`). При выполнении может возникнуть ошибка, что система не всегда создаётся одноимённая группа, поэтому необходимо ключ `-U` при создании пользователя: `useradd -U "$user"`

Для того, чтобы подставить имя пользователя в путь до домашней директории внутри скрипта, используется также название переменной со знаком `$`. Например, если имя пользователя хранится в переменной `username`, то путь до домашней директории можно сохранить в другую переменную следующим образом:

```
username=Kolobok  
mydir=/home/$username
```

```
linux-cojn:~ # bash find_or_create_user.sh  
Пользовательне test4 не найден. Создаём...  
New password:  
BAD PASSWORD: it is based on a dictionary word  
Retype new password:  
passwd: password updated successfully  
Директория отсутствует. Создаём...  
Директория создана, владелец и группа установлены
```

```
linux-cojn:/home # ls -l  
total 16  
drwxr-xr-x 21 test  users 4096 May  3 21:33 test  
drwxr-xr-x  2 root  root  4096 May  3 22:29 test2  
drwxr-xr-x  2 test3 test3 4096 May  3 22:33 test3  
drwxr-xr-x  2 test4 test4 4096 May  3 22:40 test4  
linux-cojn:/home #
```

4.8 Аргументы командной строки

При запуске скрипта ему можно передавать параметры (аргументы). Это позволяет делать скрипты более гибкими и универсальными, не запрашивая данные через `read` каждый раз.

Переменная	Описание	Пример
<code>\$0</code>	Имя скрипта	<code>./myscript.sh</code>
<code>\$1</code> , <code>\$2</code> , <code>\$3</code> ...	Аргументы по порядку (до 9)	<code>\$1</code> — первый аргумент
<code>\${10}</code> , <code>\${11}</code> ...	Аргументы начиная с 10-го	<code>\${10}</code> — десятый аргумент
<code>\$#</code>	Количество переданных аргументов	3
<code>\$@</code>	Все аргументы как отдельные слова	<code>"arg1" "arg2" "arg3"</code>

Переменная	Описание	Пример
\$*	Все аргументы как одна строка	"arg1 arg2 arg3"
\$\$	PID (идентификатор) текущего скрипта	12345
\$?	Код возврата последней выполненной команды	0 (успех)

```
#!/bin/bash
echo "Имя скрипта: $0"
echo "Первый аргумент: $1"
echo "Второй аргумент: $2"
echo "Третий аргумент: $3"

$ ./script.sh hello world 42
Имя скрипта: ./script.sh
Первый аргумент: hello
Второй аргумент: world
Третий аргумент: 42
```

4.9 Ввод данных от пользователя (команда read)

Часто скриптам требуется получать данные не только из аргументов командной строки, но и непосредственно от пользователя во время выполнения. Для этого используется команда **read**.

```
#!/bin/bash
echo "Как вас зовут?"
read name
echo "Привет, $name!"
```

```
#!/bin/bash
echo "Введите имя и возраст:"
read name age
echo "Пользователь: $name, Возраст: $age"
```

```
#!/bin/bash
read -s -p "Введите пароль: " password
echo
echo "Пароль принят"
```

-p "текст"	Вывести приглашение перед вводом
------------	----------------------------------

-s	Скрытый ввод (не отображается на экране)
----	--

Задание

Модифицируйте ваш скрипт **find_or_create_user.sh** следующим образом:

1. Если пользователь существует, спросить: "Удалить существующего пользователя? (y/n)"
2. При ответе y — удалить пользователя;
3. При ответе n — просто вывести сообщение и завершить работу
echo "Работа завершена. Пользователь \$user не изменён."
exit 0

Подсказка: Для удаления пользователя используйте команду **userdel -r user**

Примечание: # Дополнительно удаляем одноимённую группу, если она осталась

```
if getent group "$user" >/dev/null
then
groupdel "$user"
echo "Группа $user удалена."
fi
echo "Пользователь $user удалён."
```