

Практическая работа № 3

Тема «Основы работы с Bash Scriptами»

по дисциплине «Технология разработки программных приложений»

Часть 1. Базовые Bash скрипты

Данная часть работы посвящена написанию базовых bash-скриптов.

По сути своей Bash-скрипты представляют из себя ни что иное как последовательность команд командной строки, объединенных в один файл для решения какой-либо задачи.

Любой скрипт начинается с так называемой шебанг строки, которая указывает на расположение исполняемого файла той оболочки, для которой мы хотим написать скрипт. В нашем случае это bash, следовательно данная строка будет выглядеть следующим образом:

```
#!/bin/bash
```

После чего уже следует непосредственно сам скрипт.

Символом решетки в bash-скриптах обозначаются комментарии.

Напишем самый базовый скрипт, который просто будет выводить текущий каталог. Создадим файл **myscript** и запишем в него следующий код.

```
#!/bin/bash  
# This is a comment  
pwd
```

Запуск скриптов из командной строки осуществляется следующим образом. В директории с файлом скрипта необходимо написать следующую команду **./myscript**.

Однако просто так этого сделать не получится, система выдаст ошибку, необходимо выдать этому файлу права на выполнение.

```
chmod +x ./myscript
```

После чего скрипт может быть запущен.

Для вывода текста в скриптах можно воспользоваться уже известной командой **echo**.

Переменные

Как и любой другой язык `bash` обладает возможностью создания переменных.

Переменные бывают как пользовательскими, создаваемыми при работе скрипта, так и переменными среды, созданными для хранения параметров ОС.

Для обращения к переменным среды следует использовать следующую конструкцию: **\$ИМЯ_ПЕРЕМЕННОЙ**.

К примеру, выведем в сообщении путь к домашней директории текущего пользователя.

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

Заметьте, что строковый литерал — двойные кавычки — никак не повлиял на распознавание переменной среды.

Пользовательские переменные в `bash`-скриптах имеют динамический тип. Вот пример их задания с последующим выводом.

```
#!/bin/bash
# testing variables
grade=5
person="Adam"
echo "$person is a good boy, he is in grade $grade"
```

В данном случае было создано 2 переменные: **grade** и **person**.

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 «`»
- С помощью конструкции `$()`

Используя первый подход, проследите за тем, чтобы вместо обратного апострофа не ввести одиночную кавычку. Команду нужно заключить в два таких значка:

```
mydir=`pwd`
```

При втором подходе то же самое записывают так:

```
mydir=$(pwd)
```

А скрипт, в итоге, может выглядеть так:

```
#!/bin/bash
mydir=$(pwd)
echo $mydir
```

В ходе его работы вывод команды `pwd` будет сохранён в переменной `mydir`, содержимое которой, с помощью команды `echo`, попадёт в консоль.

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 $var2
var3=$(( $var2 / 4 ))
echo $var3
var4=$(( $var3 % 5 ))
echo $var4
var5=$(( $var4 ** 2 ))
echo $var5
```

Управляющая конструкция if-then-else

Для управления потоком исполнения команд в `bash`-скриптах можно использовать управляющую конструкцию `if-then-else`. Работает она как во всех языках программирования, однако имеет свои особенности. Вот её подробный синтаксис:

```

if команда1
then
команды
elif команда2
then
команды
else
команды
fi

```

В отличие от большинства известных вам на данный момент языков bash-скрипты обладают возможностью задать дополнительные условия в конструкции **if-else** при помощи ключевого слова **elif**. Пример скрипта с **if-then-else**.

```

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

```

Пример скрипта с **elif**.

```

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

```

Команда `grep` используется для поиска информации о пользователе, чье имя записано в переменной **user**, в файле `/etc/passwd`, который содержит информацию обо всех пользователях в системе.

Сравнение чисел

В отличие от привычных операций сравнения в языках программирования bash-скрипты обладают специфическим синтаксисом

операций сравнения. Например, для чисел применяются следующие операции:

- `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`.

При использовании подобных операций выражения необходимо заключать в квадратные скобки, например:

```
#!/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

Для сравнения строк используются чуть более привычные операторы, однако и они не лишены новшеств.

- `str1 = 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.

Пример сравнения строк

```
#!/bin/bash
user ="likegeeks"
if [$user = $USER]
then
echo "The user $user is the current logged in user"
fi
```

При работе с операторами < и > есть несколько особенностей, которые стоит учитывать. Во-первых, их необходимо экранировать при помощи символа \. Во-вторых, необходимо заключать имена переменных в двойные кавычки "\$val2". Приведем пример работы с подобным оператором.

```
#!/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

Наиболее используемыми командами для bash скриптов являются команды 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 file – Проверяет, существует ли файл, и соответствует ли его идентификатор группы идентификатору группы текущего пользователя.

Пример скрипта, в котором используются эти команды.

```
#!/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
```

Операторы цикла

Bash-скрипты поддерживают несколько вариантов циклов для перебора последовательностей значений. В число этих циклов входят:

- for;
- while.

Для начала рассмотрим цикл **for**. Базовая структура такого цикла выглядит следующим образом.

```
for var in list
do
команды
done
```

Самый простой пример – перебор списка простых значений.

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

Для перебора сложных значений необходимо заключать эти значения в строковые литералы – двойные кавычки, к примеру:

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

Список для цикла for может быть получен из результата выполнения какой-либо команды, полученного при помощи рассмотренной ранее подстановки команд. Как пример – перебор вывода команды cat.

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

Однако проблема такого вывода в том, что файл будет обрабатываться по словам, а не по строкам, как того хотелось бы.

Это связано с тем, что оболочка `bash` считает разделителем строки следующий набор символов:

- Пробел;
- Знак табуляции;
- Знак перевода строки.

Для того, чтобы задать разделитель необходимо использовать переменную окружения `IFS` (Internal Field Separator). Пример того, как можно выполнить итерацию лишь по строкам файла.

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

По мере работы скрипта возможно изменение переменной `IFS`, так что она может задаваться в зависимости от контекста.

Для обхода файлов, находящихся в директориях, также используется цикл `for`. Вот пример такого обхода с выводом списка файлов и директорий.

```
#!/bin/bash
for file in /home/<username>/*
do
if [ -d "$file" ]
```



```
then
echo "$file is a directory"
elif [ -f "$file" ]
then
echo "$file is a file"
fi
done
```

Как можно видеть из примера для обхода файлов можно использовать следующий путь, который подразумевает сбор всех файлов и директорий - **/home/<username>/***

Вторая разновидность циклов – цикл **while**.

```
while команда проверки условия
do
другие команды
done
```

Пример скрипта с таким циклом.

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

Для управления циклами также могут применяться привычные команды операторы **break** и **continue**.

Результат работы цикла также могут быть выведены в файл при помощи перенаправления вывода. Вот пример такого скрипта.

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

Более подробно с написанием скриптов можно ознакомиться по следующим ссылкам:

- <https://habrahabr.ru/company/ruvds/blog/325522/>
- <https://habr.com/ru/company/ruvds/blog/325928/>

Задания на выполнение 1 части практической работы:

1. Напишите сценарий, который выводит дату, время, список зарегистрировавшихся пользователей, и uptime системы и сохраняет эту информацию в файл.
2. Напишите сценарий, который выводит содержимое любого каталога или сообщение о том, что его не существует.
3. Напишите сценарий, который с помощью цикла прочитает файл и выведет его содержимое.
4. Напишите сценарий, который с помощью цикла выведет список файлов и директорий из текущего каталога, укажет, что есть файл, а что директория.
5. Напишите сценарий, который подсчитает объем диска, занимаемого директорией. В качестве директории можно выбрать любую директорию в системе.
6. Напишите сценарий, который выведет список всех исполняемых файлов в директории, для которых у текущего пользователя есть права на исполнение.

Часть 2. Развертка и запуск проекта при помощи Bash Script

1. Определение зависимостей проекта

Любой проект зависит от ряда библиотек, которые предоставляют тот или иной функционал. Для развертывания приложения необходимо, чтобы данные библиотеки были установлены в том окружении, где предполагается это самое развертывание.

На основании этого для начала необходимо определить, какие зависимости имеет проект. По ссылке <https://www.dropbox.com/s/ija7ax3sj6ysb0p/blocknote-master.tar.gz> расположен проект для скачивания. Будет скачан архив с непонятным названием, распаковать его можно при помощи команды `tar -xvf имя_архива` `имя_директории_для_распаковки`. Проект написан на языке программирования Python. Необходимо составить список зависимостей проекта в виде `requirements.txt` файла. Данный файл содержит в себе список библиотек, которые необходимо установить в окружение для запуска приложения. Подробнее про составление данного файла можно почитать по ссылке https://semakin.dev/2020/04/requirements_txt/.

Зависимости в Python можно определить по `import`'ам в файлах, однако некоторые библиотеки включены в стандартную библиотеку языка, поэтому также необходимо будет определить, является ли библиотека внешней или же встроенной в язык.

2. Создание виртуального окружения

Python позволяет создавать так называемое виртуальное окружение. Данное окружение представляет из себя отдельную копию Python с собственным набором библиотек. Оно позволяет работать с проектами, не загрязняя основной интерпретатор ненужными глобально, то есть для всей системы, библиотеками. Подробнее про создание такого рода окружений можно прочесть по ссылке <https://ru.hexlet.io/courses/python-setup-environment>.

Необходимо на основании составленного в прошлом шаге списка команд написать скрипт скачивания указанного в прошлом шаге проекта с последующим созданием виртуального окружения и настройкой его под проект, то есть установкой всех необходимых библиотек.

3. Написание скрипта запуска приложения на новой системе

Bash-скрипты позволяют создать с нуля всё необходимое окружение в системе, начиная с установки самого python-а и всего необходимого ПО для запуска приложения и заканчивая запуском самого приложения.

Для начала необходимо установить python 3. Сделать это можно при помощи команды `sudo apt install python3`.

Далее необходимо загрузить к себе на машину собственный проект.

После этого необходимо воссоздать полученное на прошлом этапе виртуальное окружение со всеми зависимостями.

Затем необходимо запустить проект из виртуального окружения при помощи следующих команд:

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

Отчет

В результате выполнения всех заданий в отчет необходимо включить следующее:

1. Титульный лист;
2. Задание;
3. Выполнение заданий;
 - а. Часть 1. Результат выполнения bash-скриптов, а также текст самого скрипта.
 - б. Часть 2. Описание найденных зависимостей, текст или скриншот файла requirements.txt, текст или скриншот итогового скрипта, скриншоты выполненной работы.
4. Небольшой вывод по проделанной работе.

Отчет о практической работе необходимо загрузить на сайт ЦДО.