<Teach Me Skills />

# Языки скриптинга

часть II

Teach Me Skills

# Вопросы по предыдущим темам или ДЗ

Teach Me Skills

# Mini-quize по новой теме:

1. **Что такое bash?**
2. **Что такое python?**
3. **При помощи чего можно выполнять скрипты по расписанию?**
4. **Где лучше всего хранить скрипты?**
5. **Нужны ли какие-то специальные программы для написания скриптов?**

Teach Me Skills

# План занятия

1. В чем разница между скриптом и программой
2. Поработаем со скриптами Bash
3. Поработаем со скриптами python
4. Создадим коллаборацию скриптов
5. Научимся правильно их запускать

# Языки скриптинга

**bash-скрипт** - это сценарии командной строки, написанные для оболочки bash.

**Сценарии командной строки** — это наборы тех же самых команд, которые можно вводить с клавиатуры, собранные в файлы и объединённые некоей общей целью. При этом результаты работы команд могут представлять либо самостоятельную ценность, либо служить входными данными для других команд. Сценарии — это мощный способ автоматизации часто выполняемых действий.

## Как устроены bash-скрипты

Создайте пустой файл с использованием команды `touch`. В его первой строке нужно указать, какую именно оболочку мы собираемся использовать. Нас интересует `bash`, поэтому первая строка файла будет такой:

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

В других строках этого файла символ решётки используется для обозначения комментариев, которые оболочка не обрабатывает. Однако, первая строка — это особый случай, здесь решётка, за которой следует восклицательный знак (эту последовательность называют шебанг) и путь к `bash`, указывают системе на то, что сценарий создан именно для `bash`.

# Языки скриптинга

Команды оболочки отделяются знаком перевода строки, комментарии выделяют знаком решётки. Вот как это выглядит:

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

whoami
```

## Установка разрешений для файла сценария

Сделаем файл исполняемым:

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

Теперь попытаемся его выполнить:

```
./myscript
```

## Вывод сообщений

Для вывода текста в консоль Linux применяется команда `echo`. Воспользуемся знанием этого факта и отредактируем наш скрипт, добавив пояснения к данным, которые выводят уже имеющиеся в нём команды:

```
#!/bin/bash
# our comment is here
echo "The current directory is:"
pwd
echo "The user logged in is:"
whoami
```

## Переменные среды

Иногда в командах оболочки нужно работать с некими системными данными. Вот, например, как вывести домашнюю директорию текущего пользователя:

```
#!/bin/bash
# display user home

echo "Home for the current user is: $HOME"
```

Обратите внимание на то, что мы можем использовать системную переменную `$HOME` в двойных кавычках, это не помешает системе её распознать.

# Языки скриптинга

## 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.

Практически всегда 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.

# Языки скриптинга

## 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
fi
```

А вот рабочий пример:

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

# Языки скриптинга

## Управляющая конструкция if-then-else

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`.

Напишем такой скрипт:

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

## 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.

## Сравнение строк

В сценариях можно сравнивать и строковые значения. Операторы сравнения выглядят довольно просто. Вот список операторов.

`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.

# Языки скриптинга

## Проверки файлов

Нижеприведённые команды используются в bash-скриптах чаще всего. Они 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.

Teach
Me
Skills

# Спасибо за внимание!