

Тема: Введение в скрипты bash.

Bash Script
`#!/bin/bash`



План занятия:

1. Правила написания скриптов. Переменные.
2. Условный оператор if.
3. Циклы for и while.
4. Регулярные выражения и утилиты для работы с регулярными выражениями.

Правила написания скриптов.

Переменные

Bash — это командный интерпретатор, работающий, как правило, в интерактивном режиме в текстовом окне.

Bash-скрипты — это сценарии командной строки, то есть наборы обычных команд, которые пользователь вводит с клавиатуры. Для автоматизации каких-то рутинных вещей эти команды объединяются в файл-сценарий, который и носит название скрипт.

Bash-скрипты — это файлы, содержащие последовательность команд, которые могут быть выполнены с помощью интерпретатора Bash.

Bash-скрипты могут быть использованы для автоматизации повторяющихся задач. Например:

- создание и отправка отчётов по электронной почте;
- проверка и обновление системы и приложений;
- сканирование и мониторинг сети;
- уведомление по почте, когда не работает сервер или служба, не создана резервная копия и т.д.

shebang (шебанг) — особая запись в начале файла, которая выглядит как сочетание символов `#!` и путь до интерпретатора. Например, `#!/bin/bash` укажет системе, что для выполнения кода после этой строки необходимо использовать `bash`. Запись `#!/usr/bin/perl` укажет, что для выполнения кода нужно использовать `perl`. После этой строки можем писать скрипт.

Комментарии в `bash` определяются символом `#`. Комментарий может быть добавлен в начале строки или встроен в код, например:

- комментарий в начале строки: `# Определяем переменные;`
- комментарий в коде: `echo "text" # Выводим сообщение на экран терминала`

```
1 #!/bin/bash
2 echo "hello World!"
```

Переменные необходимы для хранения информации. С ними можно выполнить два действия:

- установить значение переменной;
- прочитать значение переменной.

В `bash` нет строгих различий между типами переменных. С точки зрения командного интерпретатора любая переменная является строкой.

Переменная – это объект, которому дано имя. Необходимо для хранения данных и промежуточных результатов вычислений.

Объект – это:

- число
- строка
- практически что угодно



В командном интерпретаторе Bash нет строгих различий между типами переменных. Внутренне все переменные в Bash представляются как строки, и командный интерпретатор не выполняет автоматического преобразования типов данных.

Это означает, что при выполнении операций или сравнении значений переменных, Bash рассматривает их как строки. Например, при сложении двух переменных, Bash просто конкатенирует их значения вместо выполнения арифметической операции, если значения не были предварительно преобразованы в числа.

```
num1=5  
num2=10  
result=$num1$num2  
echo $result # Вывод: 510
```

Однако можно явно преобразовывать значения переменных в другие типы данных, если это необходимо, используя соответствующие операторы и функции в Bash.

Например, для выполнения арифметических операций можно использовать оператор `(())`, а для преобразования строки в число - команду **let** или арифметическое выражение `$(())`.

```
num1=5
num2=10
result=$((num1 + num2))
echo $result # Вывод: 15
```

```
num1=5
num2=10
let "result = num1 + num2"
echo $result # Вывод: 15
```


В Bash нельзя использовать операторы арифметики с числами с плавающей точкой.

Bash поддерживает только целочисленную арифметику.

Для работы с числами с плавающей точкой, можно использовать утилиту **bc**.

```
x=5.2
y=13.985
result=$(echo "$x + $y" | bc)
echo $result
```

scale=2 - точность результата, количество знаков после запятой

```
x=5.2
y=3.14
z=$(echo "scale=2; $x / $y" | bc)
echo $z
```

bash. Типы данных

Строки (Strings): Строки представляют последовательность символов и обозначаются с помощью одинарных или двойных кавычек. Например:

```
name="John"  
message='Hello, World!'
```

Числа (Numbers): В Bash числа могут быть целыми (integer) или с плавающей точкой (floating-point). Числа могут быть заданы напрямую или присвоены переменным. Например:

```
count=10  
pi=3.14159
```

Массивы (Arrays): Массивы в Bash позволяют хранить наборы элементов. Элементы массива могут быть любого типа данных, включая строки и числа.

```
fruits=("apple" "banana" "orange")  
numbers=(1 2 3 4 5)
```

Пустое значение (Null): В Bash пустое значение обозначается ключевым словом `null` или пустыми кавычками `"`.

```
empty_value=null
```

Булевы значения (Boolean)

```
is_true=true  
is_false=false
```

Кроме того, Bash поддерживает различные специальные типы данных и структуры, такие как ассоциативные массивы (associative arrays) и файловые дескрипторы (file descriptors), но они не так часто используются в повседневном программировании на Bash.

Важно отметить, что в Bash типы данных не явно объявляются, и переменные могут изменять свой тип в процессе выполнения программы.

Правила написания скриптов.

Переменные

Переменные bash

Переменные окружения

\$PWD — текущий каталог.
\$ID — покажет имя текущего пользователя и группы, в которых он состоит.
\$PATH — покажет путь до исполняемых файлов

Пользовательские переменные

a=123 присвоит переменной a значение 123
a=\$(ls) присвоит переменной a результат работы команды ls

Специальные переменные

Переменные подстановки: \$0
\$1 ..\$9
\$? — статус выполнения предыдущей команды или скрипта

В оболочке Bash **переменные подстановки** `$0`, `$1`, `$2`, ..., `$9` используются для доступа к аргументам командной строки переданным скрипту или команде. Вот их значение:

`$0`: Эта переменная содержит имя самого скрипта или команды, которая была запущена..

`$1`, `$2`, ...: Эти переменные содержат значения аргументов командной строки, переданных скрипту или команде. `$1` содержит значение первого аргумента, `$2` - второго и так далее. Всего доступно 9 аргументов: от `$1` до `$9`.

Например, предположим, у вас есть скрипт `my_script.sh`, который вы запускаете следующим образом:

```
./my_script.sh arg1 arg2 arg3
```

В этом случае:

`$0` будет равно `my_script.sh` (или полному пути к скрипту).

`$1` будет равно `arg1`.

`$2` будет равно `arg2`.

`$3` будет равно `arg3`.

`$4`, `$5`, ..., `$9` будут пустыми

Некоторые переменные окружения

- **USER:** текущее имя пользователя, использующего систему
- **HOME:** домашний каталог текущего пользователя
- **PATH:** список каталогов, разделенных двоеточиями, в которых система ищет команды
- **PS1:** основная строка приглашения (для определения отображения приглашения оболочки)
- **PWD:** текущий рабочий каталог
- **_:** самая последняя команда, выполненная в системе пользователем
- **SHELL:** оболочка, используемая для интерпретации команд в системе, она может быть много разных (например, bash, sh, zsh или другие)
- **LANG:** кодировка языка, используемая в системе
- **UID:** текущий UID для пользователя
- **HOSTNAME:** имя компьютера системы
- **TERM:** указывает тип терминала
- **OLDPWD:** предыдущий рабочий каталог
- **BASHOPTS:** список параметров, которые использовались при выполнении bash.
- **IFS:** внутренний разделитель полей для разделения ввода в командной строке. По умолчанию это пробел.
- **SHELLOPTS:** параметры оболочки, которые можно установить с помощью параметра set.

Пользовательские переменные

Пользовательские переменные Bash Linux могут быть названы любой текстовой строкой длиной до 20 символов, состоящей из букв, цифр и символа подчёркивания.

В названии учитывается регистр букв, поэтому переменная **Var1** не является переменной **var1**.

Присвоение значения переменной Bash выполняется с помощью знака равенства (=).

Слева и справа от знака не должно быть разделяющих символов по типу пробела.

Обращение к пользовательским переменным осуществляется так же, как и к системным, — с помощью знака доллара (\$). Он не используется, когда переменной присваивается значение.

```
var1=50
var2=-120
var3=hello
var4="Hello World!!!"

echo $var1
(( sum=$var1 + $var2 ))
echo $sum
```


Арифметические операции

В bash существует множество способов выполнения арифметических операций.

Рассмотрим некоторые из них.

В bash нет родной поддержки чисел с плавающей точкой. Но есть утилиты, которые умеют это делать.

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

```
# Арифметика с let
```

```
let sum=2+2 # если выражение без кавычек, пробелов быть не должно  
echo $sum
```

```
let sum="2 + 2"  
echo $sum
```

```
let sum++ # инкремент, sum=sum+1  
echo $sum
```

```
let mult="4 * 5"  
echo $mult
```

```
let x=mult/3  
echo $x
```

```
#!/bin/bash
# Простая арифметика с двойными скобками

A=$(( 4 + 5 )) # Базовый синтаксис. Можно ставить пробелы без использования кавычек.
echo $A # 9

A=$((3+5))      # Работает и без пробелов.
echo $A # 8

B=$(( A + 3 )) # Можно использовать переменные без $ перед ними.
echo $B # 11

B=$(( $A + 4 )) # А можно и с $
echo $B # 12

(( B++ ))      # Увеличение переменной на 1. Символ $ не нужен.
echo $B # 13

(( B += 3 ))   # Увеличение переменной на 3. Это краткая форма записи b = b + 3.
echo $B # 16

A=$(( 4 * 5 )) # символ * не нужно экранировать.
echo $A # 20
```

```
#!/bin/bash
# expr вместо сохранения результата в переменную по умолчанию печатает ответ.

expr 2 + 3      # базовый синтаксис, результат 5

expr "5 + 7"    # в терминал выведет строку "5 + 7"

expr 2+3        # если пробелов нет, выражение будет выведено в терминал без вычисления

expr 20 \* 5     # Некоторые символы нужно экранировать

expr 15 % 4     # остаток от целочисленного деления двух чисел

sum=$( expr 8 + 7 ) # сохраняем результат в переменную sum
echo $sum
```

Условный оператор if и циклы



Условный оператор if и циклы

Операции сравнения (наиболее используемые)

Проверка файлов:

- -e возвращает true (истина), если файл существует (exists);
- -d возвращает true (истина), если каталог существует (directory).

Сравнение строк:

- = или == возвращает true (истина), если строки равны;
- != возвращает true (истина), если строки не равны;
- -z возвращает true (истина), если строка пуста;
- -n возвращает true (истина), если строка не пуста.

Сравнение целых чисел:

- -eq возвращает true (истина), если числа равны (equals);
- -ne возвращает true (истина), если числа не равны (not equal).

```
#!/bin/bash
# проверка числа на четность
echo -n "Введите число: "
read number    # Эта команда берет ввод и сохраняет его в переменной

if [ $number -eq 0 ]
then
    echo "Число равно нулю"
elif [ $(( $number % 2 )) -eq 0 ]
then
    echo "Вы ввели $number. Число четное"
else
    echo "Вы ввели $number. Нечетное число."
fi
.
```

Условный оператор if и циклы

Цикл — последовательность, которая позволяет выполнить определённый участок кода заданное количество раз.

FOR:




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

```
# цикл for
```

```
for number in 1 2 3 4 5
```

```
do
```

```
    echo $number
```

```
done
```

```
for number in {100..105} # {100..105} - массив от 100 до 105 включительно
```

```
do
```

```
    echo $number
```

```
done
```

```
for smb in {a..z} # {100..105} - массив от a до z включительно
```

```
do
```

```
    echo $smb
```

```
done
```

```
for ((i=1; i < 10; i++)) # цикл с ос счетчиком
```

```
do
```

```
    echo $i
```

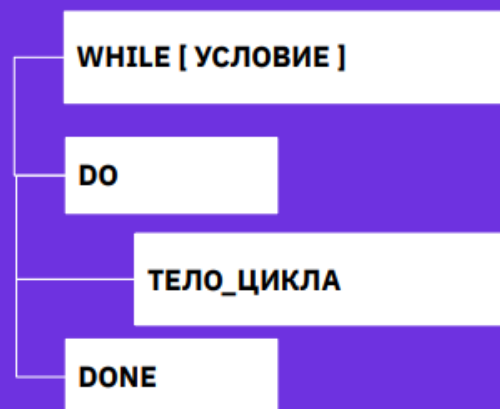
```
done
```

Условный оператор if и циклы

Здесь в качестве [условие] осуществляются операции сравнения и проверки, аналогичные условному оператору if.

Тело_цикла — команды, которые будут выполняться до тех пор, пока условие возвращает true (истина).

WHILE:



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

```
# -gt - больше, -ge - больше или равно
```

```
# -lt - меньше, -le - меньше или равно
```

```
x=1
```

```
while [ $x -lt 5 ]
```

```
do
```

```
echo "Значение счетчика: $x"
```

```
x=$(( $x + 1 ))
```

```
done
```

```
# с помощью while мы можем прочесть несколько строк из стандартного ввода. выход - Ctrl+D
```

```
while read line
```

```
do
```

```
echo $line
```

```
done
```

Команда read

Команда read позволяет считывать значения, которые пользователь вводит с клавиатуры, и сохранять их в переменные.

Синтаксис команды read выглядит следующим образом:

```
read [опции] [переменная]
```

Некоторые распространенные опции команды read включают:

- p "prompt": Отображает приглашение (prompt) для ввода пользователю.
- s: Вводимые символы не будут отображаться на экране.
- n count: Ограничивает количество символов, которое пользователь может ввести.
- t timeout: Устанавливает время ожидания ввода в секундах. Если пользователь не введет значение в указанное время, выполнение команды read продолжится с пустым значением.
- a: запись ввода в массив.

```
echo "Введите ваше имя:"
read name

echo "Привет, $name! Как дела?"
```

```
echo "Введите пароль:"
read -s password
echo "Вы ввели пароль: $password"
```

```
echo "Введите несколько слов через пробел:"
read -a words
echo "Вы ввели следующие слова:"
for word in "${words[@]}"; do
    echo "$word"
done
```

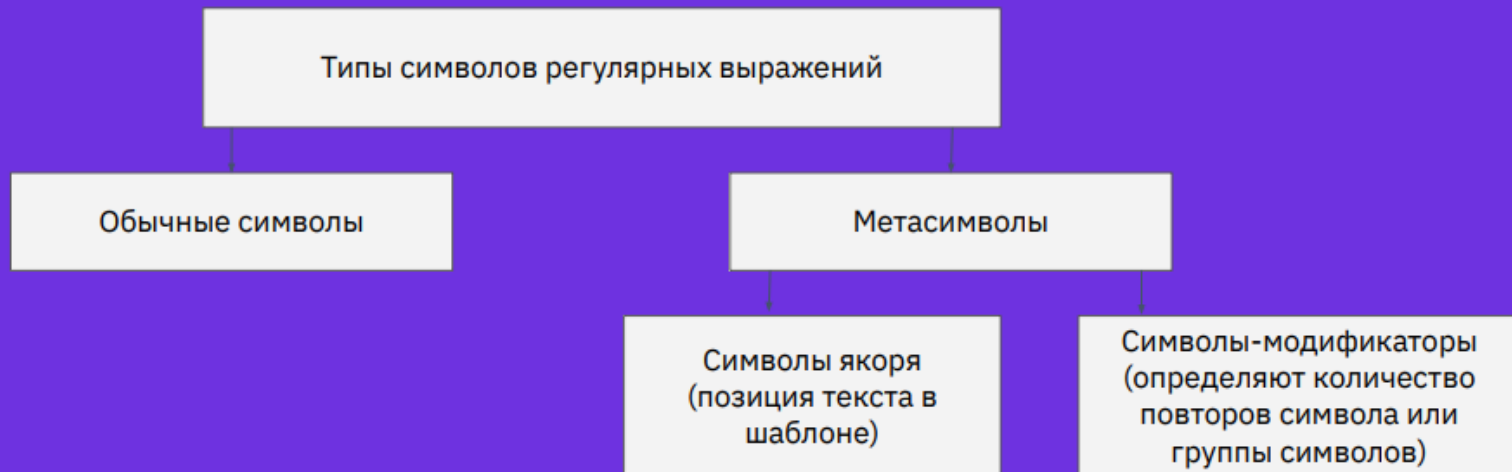
```
#!/bin/bash
read -p "What is your name: " name
echo "Hello, $name"
```

Регулярные выражения — инструмент, предназначенный для поиска, а также обработки текста по заданному шаблону.

Используя регулярные выражения, мы можем изменять текст, искать строки в файле, фильтровать список файлов согласно каким-то условиям и т. д.

Регулярные выражения — неотъемлемая часть командного интерпретатора `bash`. Они постоянно применяются в работе с командной строкой.

Регулярные выражения и утилиты для работы с регулярными выражениями



Основные символы-модификаторы

- \ — с обратной косой черты начинаются буквенные спецсимволы,
- * — указывает, что предыдущий символ может повторяться 0 или больше раз;
- + — указывает, что предыдущий символ должен повториться больше 1 или больше раз;
- ? — предыдущий символ может встречаться 0 или 1 раз;
- {n} — указывает сколько раз (n) нужно повторить предыдущий символ;
- {N,n} — предыдущий символ может повторяться от N до n раз;
- . — любой символ, кроме перевода строки;
- [az] — любой символ, указанный в скобках;
- x|y — символ x или символ y;
- [^az] — любой символ, кроме тех, что указаны в скобках;
- [a-z] — любой символ из указанного диапазона;
- [^a-z] — любой символ, которого нет в диапазоне;
- \b — обозначает границу слова с пробелом;
- \B — означает, что символ должен не быть окончанием слова;
- \d — означает, что символ — цифра;
- \D — нецифровой символ;
- \n — символ перевода строки;
- \s — любой пробельный символ: пробел, табуляция и так далее;
- \S — любой непробельный символ;
- \t — символ табуляции;
- \v — символ вертикальной табуляции;
- \w — любой буквенный символ, включая подчёркивание;
- \W — любой буквенный символ, кроме подчёркивания;
- \uXXX — конкретный указанный символ Unicode.

Регулярные выражения и утилиты для работы с регулярными выражениями

grep находит на вводе целые строки, отвечающие заданному регулярному выражению, и выводит их, если вывод не отменён специальным ключом. Grep работает с регулярными выражениями POSIX (BRE), а также все остальные, включая PCRE в разных режимах. У него есть модификация egrep, которая позволяет расширенный синтаксис (ERE).

Если нужно найти конкретную папку или один файл среди сотни других, то мы можем передать вывод команды **ls** в **grep** через вертикальную черту (|), а уже **grep**-у параметром передать нужное слово.

```
$ ls | grep Documents
```

Если же нужно найти не одно слово, а словосочетание или целое предложение, то параметр команды **grep** должно быть выделено кавычками.

```
$ ls | grep 'My Documents'
```

Поиск указанных в кавычках слов в файле Students.txt

```
$ grep 'Class 1' Students.txt
```

Регулярные выражения и утилиты для работы с регулярными выражениями

sed — потоковый текстовый редактор. Позволяет редактировать потоки данных на основе заданных правил. С помощью SED можно провести простые операции по поиску и замене слов в тексте.

Регулярные выражения и утилиты для работы с регулярными выражениями

awk — более мощная, чем SED, утилита для обработки потока данных. С точки зрения AWK данные разбиваются на наборы полей, то есть наборы символов, разделённых разделителем. AWK — это практически полноценный язык программирования, в котором есть свои переменные, операторы выбора и циклы. AWK — родоначальник языка perl.

Часто востребованная задача - выборка полей из стандартного вывода.

По умолчанию **awk** разделяет поля пробелами. Если вы хотите напечатать первое поле, вам нужно просто использовать функцию **echo** и передать ей параметр **\$1**, если функция одна, то скобки можно опустить:

```
$ echo 'one two three four' | awk '{print $1}'
```

Если поля разделены не пробелами, а другим разделителем, просто укажите в параметре **-F** нужный разделитель в кавычках, например ":"

```
$ echo 'one mississippi:two mississippi:three mississippi:four mississippi' | awk -F":" '{print $4}'
```

Иногда нужно обработать данные с неизвестным количеством полей. Если вам нужно выбрать последнее поле можно воспользоваться переменной **\$NF**. Вот так вы можете вывести последнее поле

```
$ echo 'one two three four' | awk '{print $NF}'
```

Домашнее задание:

1. Повторить материал лекции.
2. Изучить дополнительные материалы.