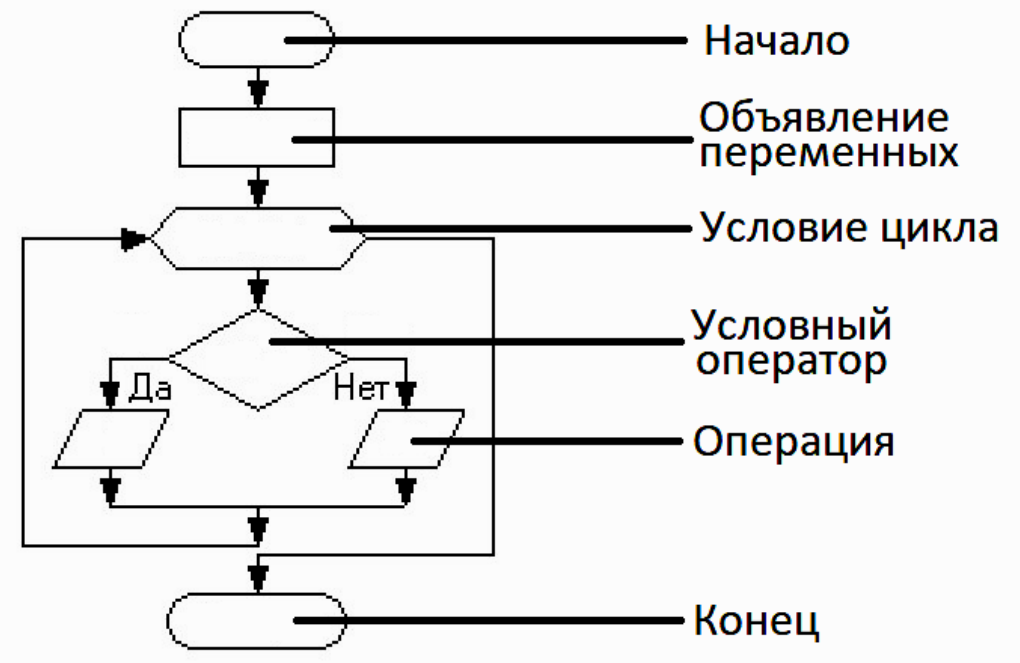
УНИВЕРСИТЕТ ИТМО

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Bash-скрипты

Мусаев Андрей Александрович
amusayev1990@gmail.com

Скрипт



Скрипт — это программа или программный файл сценарий, которые автоматизируют некоторую задачу, которую пользователь делал бы вручную, используя интерфейс программы.

Начало скрипта

which [*options*] [--] *program_name* [...]

#!/bin/bash

После #! указывается путь к интерпретатору. Также, для поиска пути можно использовать команду **whereis**.

При этом комментарии в скрипте начинаются с символа # (кроме первой строки).

Переменные


```
#!/bin/bash
# This is a basic bash script.
a=Hello
b="Good Morning"
c=16

echo $a
echo $b
echo $c
```

В bash переменные не имеют типа.

echo — команда Unix, предназначенная для отображения строки текста. Команда echo выводит текст на стандартное устройство вывода.

Знак доллара в строке означает ссылку на переменную.

Переменные

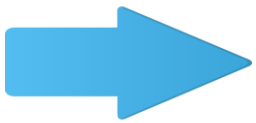
Оболочка bash назначает специальным переменным, называемым позиционными параметрами, введённые при вызове скрипта параметры командной строки:

$0 — имя скрипта.

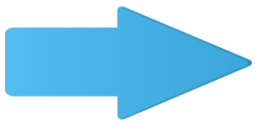
$1 — первый параметр.

$2 — второй параметр — и так далее, вплоть до переменной $9, в которую попадает девятый параметр.

```
#!/bin/bash
echo $0
echo $1
echo $2
echo $3
```



./myscript 5 10 15



```
./myscript
5
10
15
```

Условия

Существует специальная команда - **[** (левая квадратная скобка).
Она является синонимом команды **test**, и является встроенной командой (т.е. более эффективной, в смысле производительности). Эта команда воспринимает свои аргументы как выражение сравнения или как файловую проверку и возвращает код завершения в соответствии с результатами проверки (0 - истина, 1 - ложь).

Начиная с версии 2.02, Bash предоставляет в распоряжение программиста конструкцию **[[...]]** *расширенный вариант команды test*.

Круглые скобки **((...))** и предложение **let ...** так же возвращают код 0, если результатом арифметического выражения является ненулевое значение. Таким образом, арифметические выражения могут участвовать в операциях сравнения.

Условия

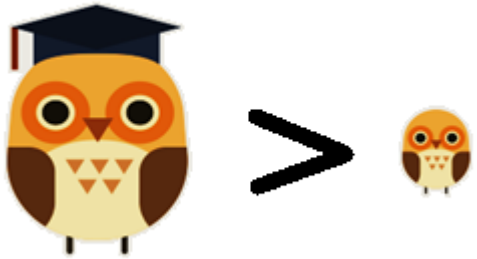
сравнение целых чисел

["$a" -eq "$b"]
-eq равно
-ne не равно
-gt больше
-ge больше или равно
-lt меньше
-le меньше или равно

(("$a" < "$b"))
< меньше
<= меньше или равно
> больше
>= больше или равно

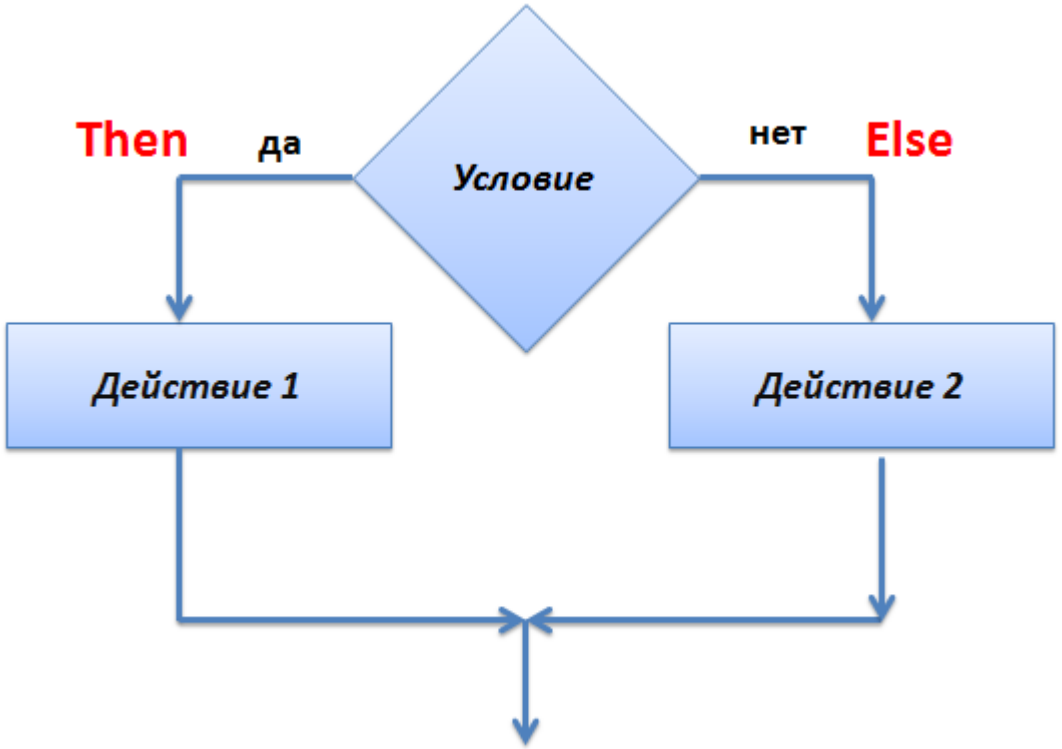
сравнение строк

["$a" = "$b"]
= равно
== равно (синоним)
!=не равно
<меньше
>больше
Символы "<", ">" необходимо экранировать (\) внутри **[ ]**.
-z строка "пустая"
-n строка не "пустая".



Условный оператор if

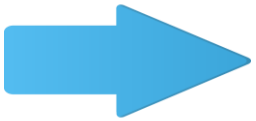
```
if [ <some test> ]
then
<commands>
else
<commands>
fi
```




```
#!/bin/bash
# Basic if statement
if [ $1 -gt 100 ]
then
echo Hey that\'s a large number.
fi
```

Условный оператор case

```
case "$variable“
in
 "$condition1" )
 <commands>
 ;;
 "$condition2" )
 <commands>
 ;;
esac
```



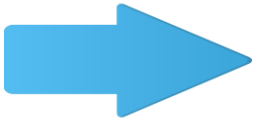
```
#!/bin/bash
echo "Нажмите клавишу"
read Keypress
case "$Keypress" in
[a-z] ) echo "буква в нижнем
регистре“
;;
[A-Z] ) echo "Буква в верхнем
регистре“
;;
[0-9] ) echo "Цифра“
;;
) echo "Знак пунктуации,
пробел или что-то другое“
;;
esac
```

Цикл for

Оболочка bash поддерживает циклы for, которые позволяют организовывать перебор последовательностей значений. Вот какова базовая структура таких циклов:

```
for var in list
do
<commands>
done
```

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



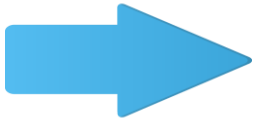
```
The first item
The second item
The third item
The fourth item
The fifth item
```

Цикл for

Если вы знакомы с языком программирования C, синтаксис описания bash-циклов for может показаться вам странным, так как привыкли вы, очевидно, к такому описанию циклов:

for ((начальное значение переменной *; условие окончания цикла; изменение переменной*))

```
#!/bin/bash
for (( i=1; i <= 10; i++ ))
do
echo "number is $i"
done
```



```
number is 1
number is 2
…
number is 9
number is 10
```

Цикл while

Конструкция for — не единственный способ организации циклов в bash-скриптах. Здесь можно пользоваться и циклами **while**. В таком цикле можно задать команду проверки некоего условия и выполнять тело цикла до тех пор, пока проверяемое условие возвращает ноль, или сигнал успешного завершения некоей операции. Когда условие цикла вернёт ненулевое значение, что означает ошибку, цикл остановится.

```
while [ <some test> ]
do
<commands>
done
```