

Лабораторная работа № 3

Циклы и функции в Bash

Цикл WHILE

Цикл с оператором while может принимать одну команду или конвейер команд, выдающих true или false.

Выполните простой цикл:

```
i=0
while (( i < 100 ))
do
    echo $i
    let i++
done
```

Что означают круглые скобки?

В двойных скобках воспроизводится числовая (C/Java/Python) логика повышенного уровня. Любое ненулевое значение считается истинным, и только ноль — обратное значение всем остальным операторам if в bash — ложным. Например:

```
if (( $? )) ; then echo "previous command failed" ; fi
```

сделает то, что вы хотите/ожидаете, — если предыдущая команда завершилась неудачно, то \$? будет содержать ненулевое значение; внутри (()) ненулевое значение будет истинным и ветвь then будет выполнена.

Создайте файл while.sh с помощью echo. Добавьте в него текст программки с помощью nano и запустите ./while.sh

Изменит текст файла на следующий:

```
while ls | grep -q pdf
do
    echo -n 'there is a file with pdf in its name here:'
    pwd
    cd ..
done
```

grep используется для поиска на вводе целые строки, отвечающие заданному регулярному выражению (в нашем случае — pdf), и выводит их, если вывод не отменён специальным ключом.

Ls — вывод каталога

Pwd - выводит полный путь от корневого каталога к текущему рабочему каталогу

Цикл FOR

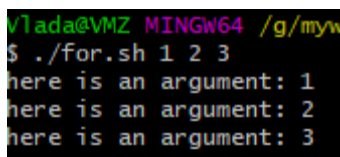
```
for ((i=0; i < 10; i++))
do
    echo $i
done
```

Скрин результата выполнения занесите в файл

Это был знакомый вам вариант использования цикла FOR

Давайте рассмотрим другой способ:

```
for ARG
do
    echo here is an argument: $ARG
done
```



```
Vlada@VMZ MINGW64 /g/мyw
$ ./for.sh 1 2 3
here is an argument: 1
here is an argument: 2
here is an argument: 3
```

Попробуйте ввести разное число аргументов.

Еще один способ использования цикла FOR в случае заранее известного числа переменных:

```
for VAL in there is a simple text
do
    echo $VAL
done
```

ФУНКЦИИ

Синтаксис функции в bash:

```
function func_name ()
{
    # Тело функции
}
```

*Если указанная внутри функции команда не объявлена как local, переменные в видимой области являются глобальными. Цикл for, устанавливающий и увеличивающий значение i, можно использовать в любом месте вашего кода. □

*Скобки — это наиболее популярные символы для группировки в теле функции, но разрешен любой из составных синтаксисов команд оболочки. □

*Перенаправление ввода/вывода (I/O), заключенное в фигурные скобки, распространяется на все операторы внутри функции.

*В определении функции параметры не объявляются. Какие бы аргументы и их количество при вызове функции ни приводились, они передаются этой функции. Функция вызывается (активизируется) так же, как и любая команда в командной оболочке.

Определив `my_function` как функцию, вы можете вызвать ее следующим образом:

```
My_function ke408 ke409
```

Эта команда вызывает функцию `My_function`, предоставляя ей два аргумента.

Аргументы функции

Внутри определения функции аргументы упоминаются так же, как параметры сценария оболочки, то есть как `$1`, `$2` и т. д. Это означает, что аргументы «скрывают» параметры, первоначально переданные в сценарий. Если вы хотите получить доступ к первому параметру скрипта, то перед вызовом функции нужно сохранить `$1` в переменной (или передать его в качестве параметра функции). Другие переменные установлены соответственно: `$#` выдает количество аргументов, переданных функции, хотя обычно мы получаем количество аргументов, переданных самому сценарию. Единственное исключение — `$0`, которая в функции не изменяется. Она сохраняет свое значение как имя скрипта, а не функции.

Шаблон соответствия

Самый простой подстановочный знак — символ звездочки (*), который будет соответствовать любому количеству любых символов. Поэтому, когда используется только подстановочный знак, он сопоставляет все файлы в текущем каталоге. Звездочку также можно указывать вместе с другими символами. Например, `*.txt` соответствует всем файлам в текущем каталоге, имена которых заканчиваются четырьмя символами `.txt`.

Шаблон `/usr/bin/g*` будет соответствовать всем файлам в `/usr/bin`, которые начинаются с буквы `g`. Другой специальный символ для сопоставления — вопросительный знак (?), который соответствует одному символу. Например, `source.?` будет соответствовать `source.c` или `source.o`, но не `source.py` или `source.cpp`.

Последний из трех подстановочных символов для сопоставления — квадратные скобки: []. Сопоставление может быть выполнено с любым из символов, перечисленных в

квадратных скобках. Так, шаблон `x[abc]y` соответствует любому или всем файлам с именами `xaу`, `xby` или `xcy` при условии, что они существуют. Вы можете указать диапазон в квадратных скобках, например: `[0–9]` для всех цифр. Если первый символ в скобках — восклицательный знак (!) или «шляпка» (^), то шаблон определяет все что угодно, кроме оставшихся символов в скобках. Например, `[aeiou]` будет соответствовать гласным буквам, тогда как `^[aeiou]` — любым символам (включая цифры и знаки пунктуации), кроме гласных

Специальные классы символов

`[:alpha:]` — соответствует любому алфавитному символу, записанному в верхнем или нижнем регистре.

`[:alnum:]` — соответствует любому алфавитно-цифровому символу, а именно — символам в диапазонах 0-9, A-Z, a-z.

`[:blank:]` — соответствует пробелу и знаку табуляции.

`[:digit:]` — любой цифровой символ от 0 до 9.

`[:upper:]` — алфавитные символы в верхнем регистре — A-Z.

`[:lower:]` — алфавитные символы в нижнем регистре — a-z.

`[:print:]` — соответствует любому печатаемому символу.

`[:punct:]` — соответствует знакам препинания.

`[:space:]` — пробельные символы, в частности — пробел, знак табуляции

Классы символов указываются как `[:ctrl:]`, но в дополнительных квадратных скобках (поэтому у вас есть два набора скобок). Например, шаблон `*[:punct:].jpg` будет соответствовать любому имени файла, имеющему любое количество любых символов, за которыми следуют знаки пунктуации, а за ними — буквы `jpg`. Таким образом, он будет соответствовать файлам с именами `wow!Jpg`, `some.jpg` или `photo.jpg`, но не файлу `this.is.myjpg`, потому что прямо перед `jpg` нет знака пунктуации

Скачайте (git clone) файл `osdetect` с <https://github.com/VladaZhernova/StudentSpring2021> и откройте его в режиме просмотра в `bash`.

Мы используем встроенную в `bash` команду `type`, чтобы сообщить, к какому виду команд (псевдоним, ключевое слово, функция, встроенная команда или файл) относятся ее аргументы. Опция `-t` говорит, что ничего не нужно выводить, если команда не найдена. В этом случае возвращается значение `false`. Мы перенаправляем весь вывод (и `stdout`, и `stderr`) в `/dev/null`, тем самым отбрасывая его, так как хотим знать только, была ли найдена команда `wevtutil`.

Если wevtutil не найдена, мы снова используем встроенную в bash команду type, но на этот раз ищем команду scutil, которая доступна только в системах MacOS.

Проверочные вопросы и задания:

Создайте скрипт с именем My_argt.sh, который считает и сообщает количество переданных в него аргументов.

Добавьте изменения, чтобы скрипт выводил каждый аргумент в новой строке.

Измените скрипт так, чтобы перед каждым аргументом выводился его номер:

```
$ bash argcnt.sh London is the capital of GB
```

```
6 arguments
```

```
arg1: London
```

```
arg2: is
```

```
arg3: the
```

```
arg4: capital
```

```
arg5: of
```

```
arg6: GB
```

```
$
```