Bash-скрипты: функции

Оболочка bash предоставляет такую возможность, позволяя создавать функции. Функции bash — это именованные блоки кода, которые можно повторно использовать в скриптах.

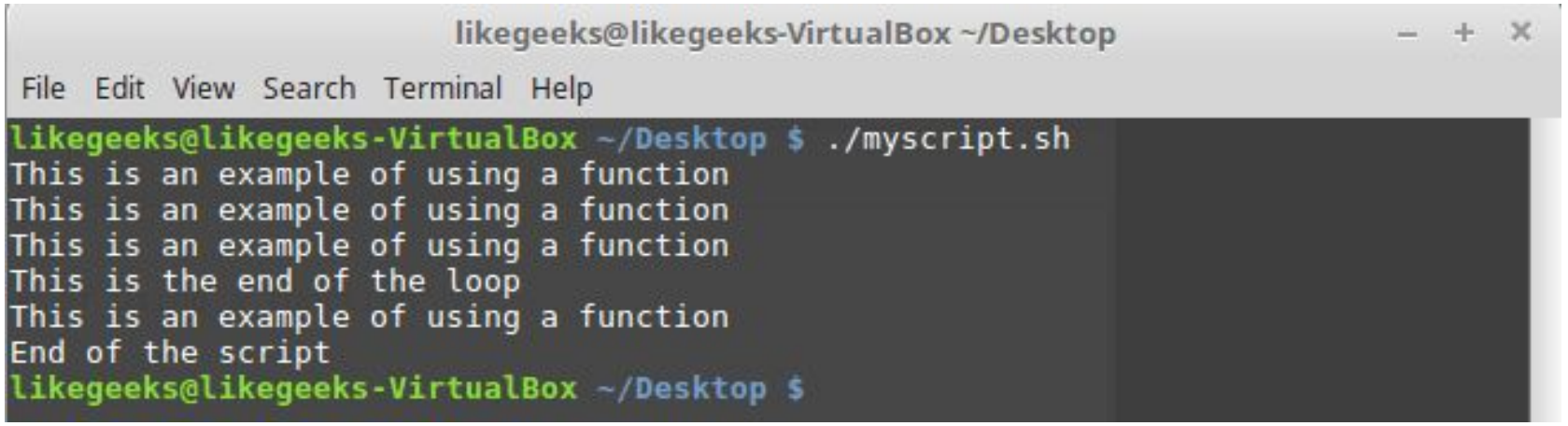
Структура объявления функций:

— functionName { } – функция без параметров;

— functionName() { } – функция с параметрами.

Напишем с вами скрипт, содержащий объявление функции и использующий её:

```
#!/bin/bash
function myfunc {
echo "This is an example of using a function"
}
count=1
while [ $count -le 3 ]
do
myfunc
count=$(( $count + 1 ))
done
echo "This is the end of the loop"
myfunc
echo "End of the script"
```

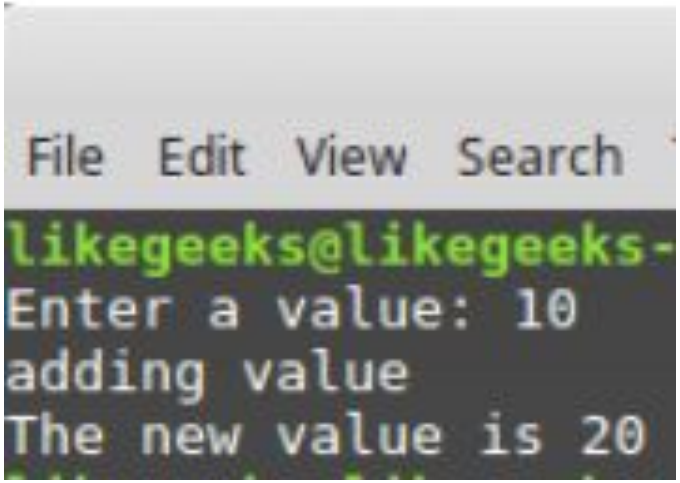

likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript.sh
This is an example of using a function
This is an example of using a function
This is an example of using a function
This is the end of the loop
This is an example of using a function
End of the script
likegeeks@likegeeks-VirtualBox ~/Desktop $

Использование команды return

Команды **return** позволяет задавать возвращаемый функцией целочисленный код завершения. Есть два способа работы с тем, что является результатом вызова функции.

Первый способ:

```
#!/bin/bash
function myfunc {
read -p "Enter a value: " value
echo "adding value"
return $(( $value + 10 ))
}
myfunc
echo "The new value is $?"
```



Второй способ заключается в записи данных, выводимых функцией, в переменную. Рассмотрим пример:

```
#!/bin/bash
function myfunc {
read -p "Enter a value: " value
echo $(( $value + 10 ))
}
result=$( myfunc)
echo "The value is $result"
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop $
Enter a value: 10
The value is 20
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Аргументы функций

Функции могут использовать стандартные позиционные параметры, в которые записывается то, что передаётся им при вызове. Например, имя функции хранится в параметре $0, первый переданный ей аргумент — в $1, второй — в $2, и так далее.

Количество переданных функции аргументов можно узнать, обратившись к переменной $#.

Аргументы передают функции, записывая их после её имени:

myfunc $val1 10 20

Глобальные переменные

Глобальные переменные — это переменные, которые видны из любого места bash-скрипта. Если вы объявили глобальную переменную в основном коде скрипта, к такой переменной можно обратиться из функции.

Почти то же самое справедливо и для глобальных переменных, объявленных в функциях. Обращаться к ним можно и в основном коде скрипта после вызова функций.

```
#!/bin/bash
function myfunc {
value=$(( $value + 10 ))
}
read -p "Enter a value: " value
myfunc
echo "The new value is: $value"
```

Локальные переменные

Переменные, которые объявляют и используют внутри функции, могут быть объявлены локальными. Для того, чтобы это сделать, используется ключевое слово local перед именем переменной:

local temp=$(($temp + 5))

Если за пределами функции есть переменная с таким же именем, это на неё не повлияет. Ключевое слово **local** позволяет отделить переменные, используемые внутри функции, от остальных переменных.

```
#!/bin/bash
function myfunc {
local temp=$[  $temp  + 5 ]
echo "The Temp from inside function is $temp"
}
temp=4
myfunc
echo "The temp from outside is $temp"
```

Bash-скрипты: Регулярные выражения

Регулярные выражения — это очень мощный инструмент для поиска текста по шаблону, обработки и изменения строк, который можно применять для решения множества задач.

Вот основные из них:

— Проверка ввода текста;

— Поиск и замена текста в файле;

— Пакетное переименование файлов;

— Проверка строки на соответствие шаблону.

Bash-скрипты: Регулярные выражения

Регулярные выражения — это очень мощный инструмент для поиска

В регулярных выражениях могут использоваться два типа символов:

— Обычные буквы;

— Метасимволы.

Обычные символы — это буквы, цифры и знаки препинания, из которых состоят любые строки. Все тексты состоят из букв и вы можете использовать их в регулярных выражениях для поиска нужной позиции в тексте.

Метасимволы — это кое-что другое, именно они дают силу регулярным выражениям. С помощью метасимволов вы можете сделать намного больше чем поиск одного символа. Вы можете искать комбинации символов, использовать динамическое их количество и выбирать диапазоны.

Все спецсимволы можно разделить на два типа, это символы замены, которые заменяют собой обычные символы, или операторы, которые указывают сколько раз может повторяться символ. Синтаксис регулярного выражения будет выглядеть таким образом:

обычный_символ спецсимвол_оператор

спецсимвол_замены спецсимвол_оператор

- \ — с обратной косой черты начинаются буквенные спецсимволы, а также он используется если нужно использовать спецсимвол в виде какого-либо знака препинания;
- ^ — указывает на начало строки;
- $ — указывает на конец строки;
- * — указывает, что предыдущий символ может повторяться 0 или больше раз;
- + — указывает, что предыдущий символ должен повторится больше один или больше раз;
- ? — предыдущий символ может встречаться ноль или один 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 — обозначает что символ должен быть внутри слова, например, ux совпадет с uxb или tuxedo, но не совпадет с Linux;
- \d — означает, что символ — цифра;
- \D — нецифровой символ;
- \n — символ перевода строки;
- \s — один из символов пробела, пробел, табуляция и так далее;
- \S — любой символ кроме пробела;
- \t — символ табуляции;
- \v — символ вертикальной табуляции;
- \w — любой буквенный символ, включая подчеркивание;
- \W — любой буквенный символ, кроме подчеркивания;
- \uXXX — символ Unicdoe.