**Лабораторная работа №5. Командные файлы и язык Shell**

**Цель:** **изучение языка сценариев Unix Shell.**

**ЗАДАНИЕ**

1. Изучите главу 6 теоретического курса «Командные файлы и язык shell».
2. Выполните на рабочем компьютере все практические примеры из данной лабораторной работы.
3. Разработайте скрипт, выполняющий следующие действия. Через параметры командной строки ему передаются от одного до пяти чисел. Требуется вывести максимальное значение и сумму квадратов введенных чисел.
4. Пусть существует файл, содержащий строки вида «имя=значение». Разработайте скрипт, который будет по заданному имени находить соответствующее значение из правой части строки. Имя файла и имя-ключ передаются через параметры командной строки. Если один или оба параметра не заданы, то они запрашиваются у пользователя в интерактивном режиме.
5. Разработайте скрипт, который просматривает все файлы в заданном каталоге. Если первой строкой файла является «#!/bin/bash» и файл не обладает правами на исполнение владельцем, то информация о таком файле выводится на экран и файлу устанавливается право на исполнение владельцем. Каталог передается через первый параметр командной строки. Действия по анализу и изменению прав доступа для некоторого файла оформите в виде функции.

```
guzovskiy@debian:~/lab_shell$ nano max_and_squares.sh
guzovskiy@debian:~/lab_shell$ chmod +x max_and_squares.sh
```

Рисунок 1 - Создание файла1

```
  GNU nano 5.4                    max_and_squares.sh *
#!/bin/bash

# Проверка количества аргументов
if [ $# -lt 1] || [ $# -gt 5 ]; then
 echo "Введите от 1 до 5 чисел через пробел"
 exit 1
fi

max=$1
sum=0

for num in "$@"; do
 #Проверка на число
 if ! [[ "$num" =~ ^-?[0-9]+$ ]]; then
  echo "Ошибка: '$num' не является целым числом"
  exit 1
 fi

 #Максимум
 if [ "$num" -gt "$max" ]; then
  max=$num
 fi

 #Сумма квадратов
 sum=$((sum + num * num))
done

echo "Максимум: $max"
echo "Сумма квадратов: $sum"

^G Help       ^O Write Out  ^W Where Is   ^K Cut       ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste     ^J Justify    ^_ Go To Line
```

Рисунок 2 - Файл1

```
./max_and_squares.sh: строка 9: sum-0: команда не найдена
Максимум: 9
Сумма крадратов: 101
```

Рисунок 3 - Результат выполнения

```
guzovskiy@debian:~/lab_shell$ nano find_value.sh
```

Рисунок 4 - Создание файла2

```
  GNU nano 5.4                          find_value.sh *
#!/bin/bash

filename=$1
key=$2

# Запрос интерактивно, если не передано
if [ -z "$filename" ]; then
 read -p "Введите имя файла: " filename
fi

if [ -z "$key" ]; then
 read -p "Введите ключ: " key
fi

# Проверка файла
if [ ! -f "$filename" ]; then
 echo "Файл '$filename' не найден"
 exit 1
fi

# Поиск значения
value=$(grep "^$key=" "$filename" | cut -d'=' -f2)

if [ -z "$value" ]; then
 echo "Ключ '$key' не найден"
else
 echo "Значение: $value"
fi
```

Рисунок 5 - Файл2

```
guzovskiy@debian:~/lab_shell$ echo -e "name=Женя\nage=20\ncity=Гомель" > data.txt
guzovskiy@debian:~/lab_shell$ ./find_value.sh data.txt name
bash: ./find_value.sh: Отказано в доступе
guzovskiy@debian:~/lab_shell$ chmod +x find_value.sh
guzovskiy@debian:~/lab_shell$ ./find_value.sh data.txt name
Значение: Женя
```

Рисунок 6 - Создание тестового файла и результат выполнения

```
guzovskiy@debian:~/lab_shell$ nano check_exec.sh
guzovskiy@debian:~/lab_shell$ chmod +x check_exec.sh
```

Рисунок 7 - Создание файла3

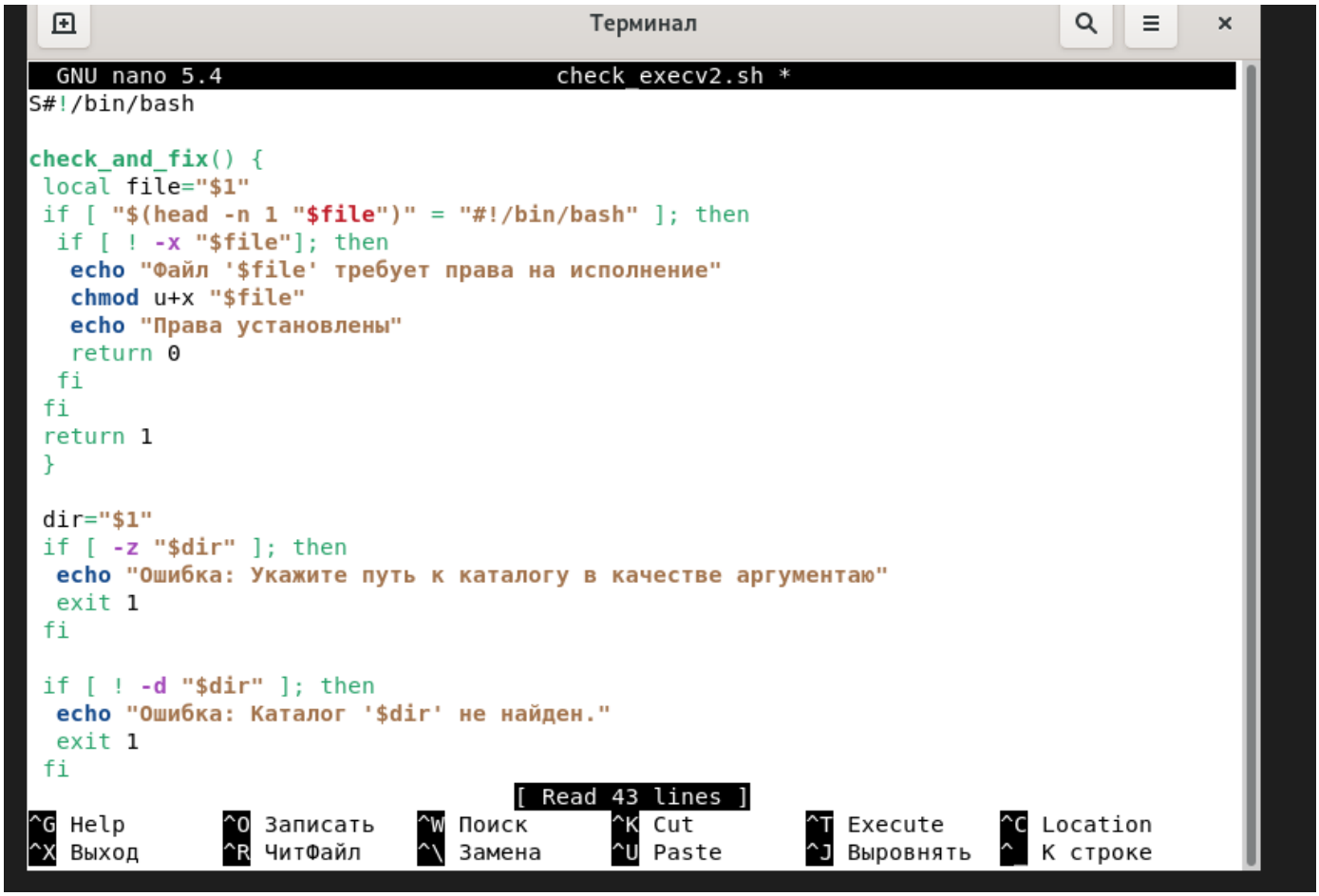


Рисунок 8 - Файл3

```
guzovskiy@debian:~/lab_shell$ ./check_execv2.sh ~/lab_shell
Проверка файлов в каталоге: /home/guzovskiy/lab_shell
```

Рисунок 9 - Результат выполнения

## Контрольные вопросы

1.Программа на языке Shell представляет собой текстовый файл, содержащий последовательность команд, которые интерпретируются оболочкой (например, bash). Такие скрипты позволяют автоматизировать задачи, управлять файлами, выполнять системные операции и обрабатывать данные. Shell удобен для простых сценариев, системного администрирования и автоматизации, но не подходит для сложных вычислений, графических интерфейсов или задач, требующих высокой производительности — в таких случаях лучше использовать языки программирования общего назначения, такие как Python, C++ или Java.

2.Переменные в Shell не требуют предварительного объявления типа — они автоматически интерпретируются как строки. Переменная создаётся простым присваиванием: имя=значение, без пробелов вокруг знака =. Для обращения к

значению переменной используется $имя. Ввод осуществляется через команду read, а вывод — через echo. Например, read name считывает ввод пользователя, а echo "Привет, $name" выводит результат.

3.Позиционные параметры — это аргументы, переданные скрипту при запуске, доступные через $1, $2, ..., $# (количество аргументов), $@ (все аргументы). Специальные параметры включают $0 (имя скрипта), $? (код возврата последней команды), $$ (PID текущего процесса). Переменные окружения — это глобальные переменные, доступные всем процессам, например PATH, HOME, USER, и они задаются через export.

4.В Shell используются разные типы скобок и кавычек: одинарные кавычки ' ' сохраняют текст буквально, двойные кавычки " " позволяют интерполяцию переменных, а обратные кавычки `команда` или конструкция $(команда) выполняют команду и возвращают результат. Для арифметических выражений используется ((выражение)) или let, а также expr и $((выражение)). Например, sum=$((a + b)) вычисляет сумму.

5.Команда if используется для условного выполнения команд, а test — для проверки условий, таких как сравнение чисел, строк или существование файлов. Основные параметры test: -f (файл существует), -d (каталог существует), -x (файл исполняемый), -z (строка пуста), = (строки равны).

6.Циклы for, while и until позволяют выполнять повторяющиеся действия. for перебирает список значений, while выполняется, пока условие истинно, until — пока условие ложно. Особенность Shell — простота синтаксиса и тесная интеграция с командами оболочки, но ограниченные возможности по сравнению с языками с полноценными структурами данных.

7.Функции в Shell определяются через имя_функции() { команды }. Параметры передаются аналогично позиционным аргументам: $1, $2 и т.д. Функции позволяют структурировать код, переиспользовать логику и упрощать скрипты. Возврат значения осуществляется через return или echo, если нужно передать результат в переменную.