Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Отчет по Лабораторной работе №4

по курсу “ОС Linux”

Студент \_\_\_\_\_\_\_\_\_\_\_ Красиков И.А.

подпись, дата

Группа ПИ-21-1

Руководитель \_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

подпись, дата

Липецк 2023 г.

**Оглавление**

[Цель работы 3](#_Toc1)

[Задание 4](#_Toc2)

[Ход работы 6](#_Toc3)

[I часть 6](#_Toc4)

[II часть 9](#_Toc5)

[Вывод 33](#_Toc6)

[Ответы на контрольные вопросы 34](#_Toc7)

# **Цель работы**

- изучить основные возможности языка программирования высокого уровня Shell;

- получить навыки написания и использования скриптов.

# **Задание**

1. Используя команды ECHO, PRINTF, вывести информационные сообщения на экран.

2. Присвоить переменной А целочисленное значение. Просмотреть значение переменной А.

3. Присвоить переменной В значение переменной А. Просмотреть значение переменной В.

4. Присвоить переменной С значение "путь до своего каталога". Перейти в этот каталог с использованием переменной.

5. Присвоить переменной D значение "имя команды", а именно, команды РАТЕ. Выполнить эту команду, используя значение переменной.

6. Присвоить переменной Е значение "имя команды", а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной.

7. Присвоить переменной F значение "имя команды", а именно, сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.

Написать скрипты, при запуске которых выполняются следующие действия:

1. Программа запрашивает значение переменной, а затем выводит значение этой переменной.

2. Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной.

3. Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды: а) EXPR; б) BC).

4. Вычислить объем цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран.

5. Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки.

6. Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.

7. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно.

8. Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения.

9. Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран.

10. Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются.

11. В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc.

12. Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение.

13. Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог, просматривается содержимое файла.

14. Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются соответствующие сообщения (использовать имена файлов и/или позиционные параметры).

15. Если файл запуска программы найден, программа запускается (по выбору).

16. В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля, содержимое файла сортируется по первому столбцу по возрастанию, отсортированная информация помещается в другой файл, содержимое которого затем отображается на экране.

Для сравнения с другими языками программирования проделайте аналогичные действия на Java, Си и Python.

# **Ход работы**

## **I часть**

1) Используя команды ECHO, PRINTF, вывести информационные сообщения на экран.

Для начала нужно создать файл для этого используем команду vim.

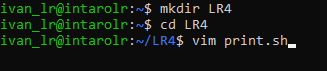


Рисунок 1.1 – Создание каталога и файла для скрипта

После чего можем написать скрипт.

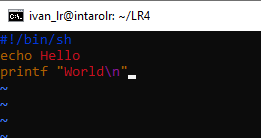


Рисунок 1.2 – Написание скрипта

Для того, чтобы запустить файл как скрипт, нужно выдать ему формат исполняемого файла.

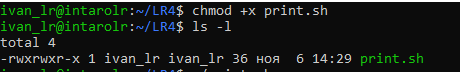


Рисунок 1.3 – Задание файлу формата исполняемого файла

После чего можем запустить скрипт.

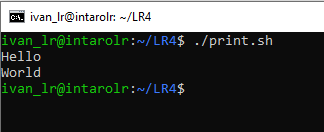


Рисунок 1.4 – Запуск скрипта

2) Присвоить переменной А целочисленное значение. Просмотреть значение переменной А.

Также можно писать на языке Shell не в файле, а в командной строке.

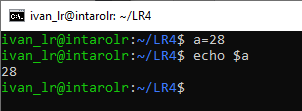


Рисунок 2 – Создание переменной, а также присвоение ей значение и вывод значение на экран

3) Присвоить переменной В значение переменной А. Просмотреть значение переменной В.

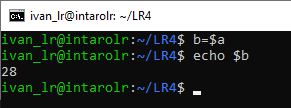


Рисунок 3 – Присвоение значения переменной B и вывод

4) Присвоить переменной С значение "путь до своего каталога". Перейти в этот каталог с использованием переменной.

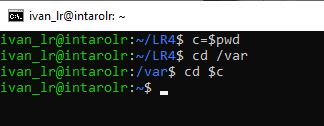


Рисунок 4 – Присвоение значения переменной C и вывод

5) Присвоить переменной D значение "имя команды", а именно, команды DАТЕ. Выполнить эту команду, используя значение переменной.

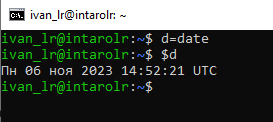


Рисунок 5 – Присвоение значения переменной D и вывод

6) Присвоить переменной Е значение "имя команды", а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной.

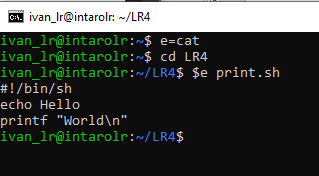


Рисунок 6 – Присвоение значения переменной E и вывод

7) Присвоить переменной F значение "имя команды", а именно, сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.

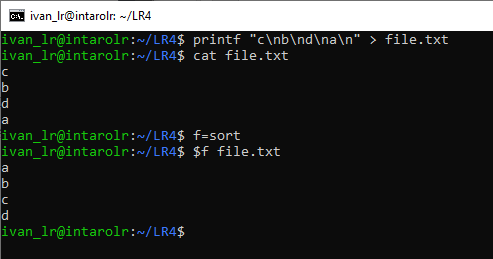


Рисунок 7 – Создание файла, вывод содержимого файла, присвоение команды переменной F, сортировка файла.

## **II часть**

1) Программа запрашивает значение переменной, а затем выводит значение этой переменной.

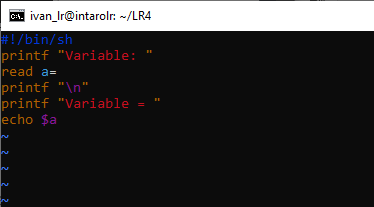


Рисунок 8.1 – Код скрипта input.sh

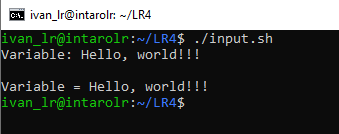


Рисунок 8.2 – Запуск скрипта input.sh

Для сравнения код на Python.

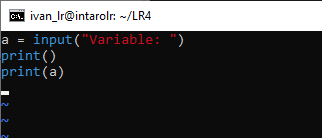


Рисунок 8.3 – Код на Python

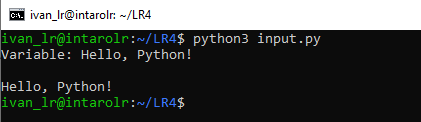


Рисунок 8.4 – Запуск кода на Python

2) Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной.

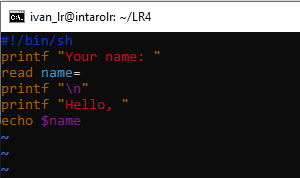


Рисунок 9.1 – Код скрипта hello.sh

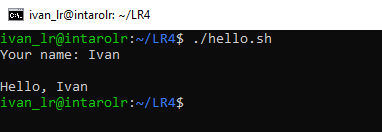


Рисунок 9.2 – Запуск скрипта hello.sh

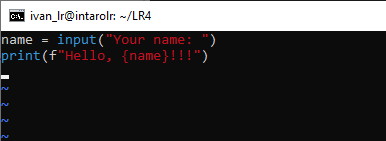


Рисунок 9.3 – Код на Python (hello.py)

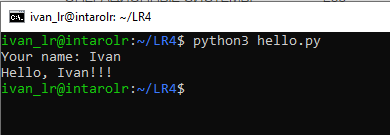


Рисунок 9.4 – Запуск кода на Python (hello.py)

3) Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды: а) EXPR; б) BC).

expr – базовый целочисленный калькулятор. Поддерживает операции сложения/деления/умножения/вычитания

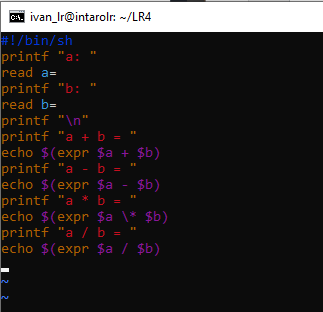


Рисунок 10.1 – Код скрипта expr.sh

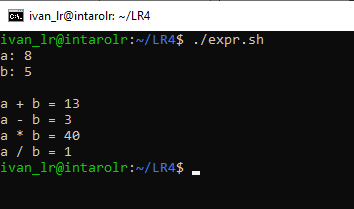


Рисунок 10.2 – Запуск скрипта expr.sh

bc – интерактивный интерпретатор Си-подобного языка, позволяет выполнять вычисления с произвольно заданной точностью.

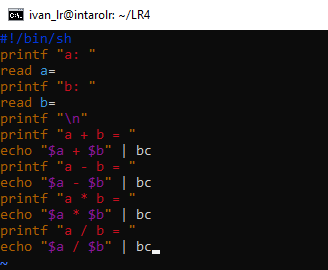


Рисунок 10.3 – Код скрипта bc.sh

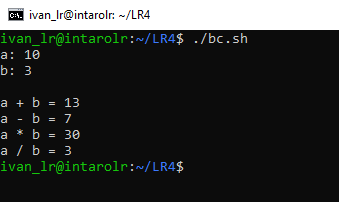


Рисунок 10.4 – Запуск скрипта bc.sh

Программа на Python:

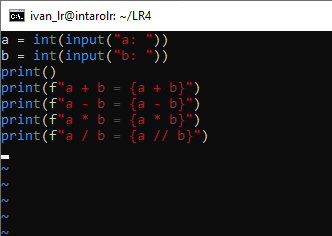


Рисунок 10.5 – Код на Python (calc.py)

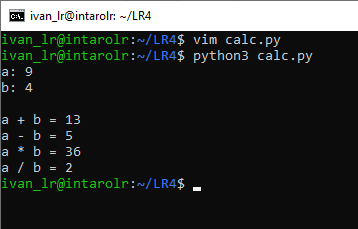


Рисунок 10.6 – Запуск кода на Python (calc.py)

4) Вычислить объем цилиндра. Исходные данные запрашиваются программой Результат выводится на экран.

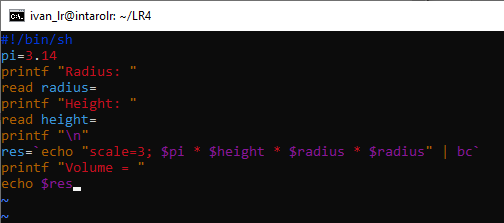


Рисунок 11.1 – Код скрипта vol.sh

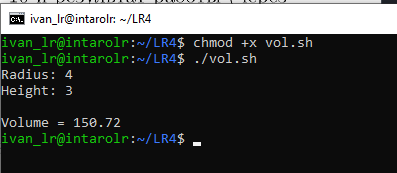


Рисунок 11.2 – Запуск скрипта vol.sh

Код на Python:

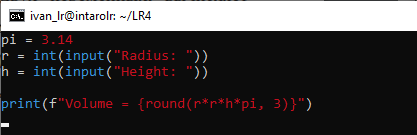


Рисунок 11.3 – Код на Python (vol.py)

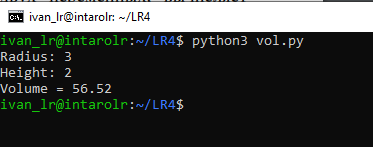


Рисунок 11.4 – Запуск программы на Python

5) Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки.

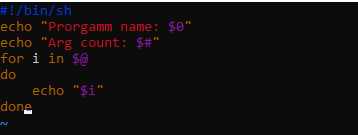


Рисунок 12.1 – Код скрипта arg.sh

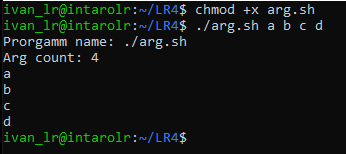


Рисунок 12.2 – Запуск скрипта arg.sh

Код на Python для сравнения:

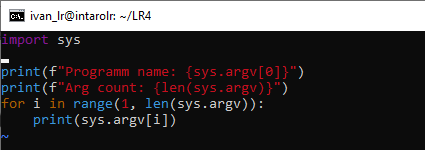


Рисунок 12.3 – Код программы на Python (arg.py)

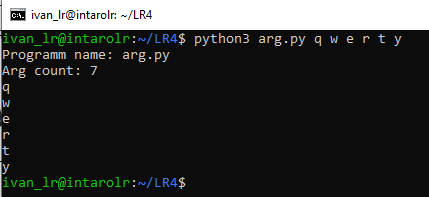


Рисунок 12.4 – Запуск программы на Python (arg.py)

6) Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.

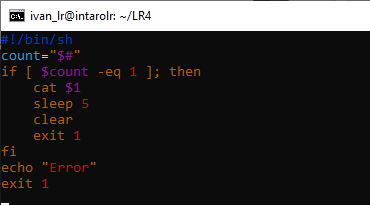


Рисунок 13.2 – Код скрипта farg.sh

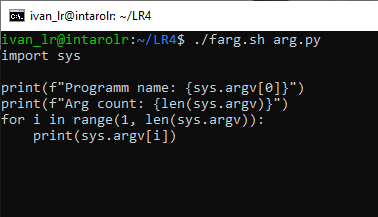


Рисунок 13.3 – Запуск скрипта farg.sh

Код на Python:

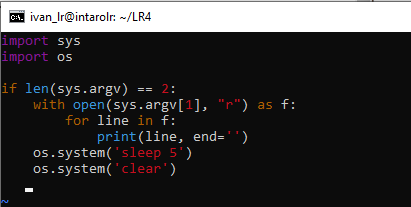


Рисунок 13.4 – Код программы на Python (farg.py)

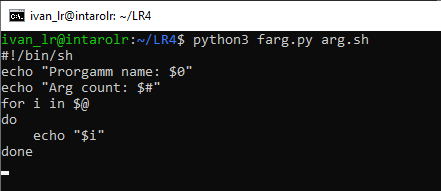


Рисунок 13.5 - Запуск программы на Python (farg.py)

7) Используя FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно.

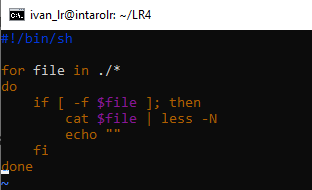
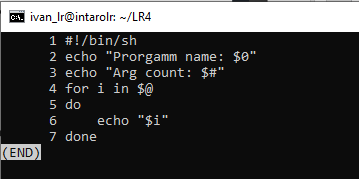
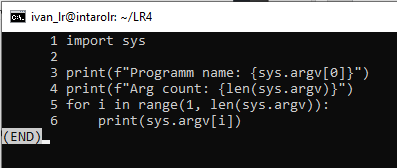


Рисунок 13.6 – Код скрипта filesdir.sh



Рисунок 13.7 – Запуск скрипта filesdir.sh



Рисунки 13.8 и 13.9 – Результат запуска

Код на Python:

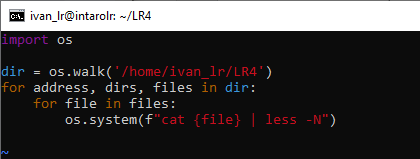


Рисунок 13.10 – Код программы на Python (filesdir.py)

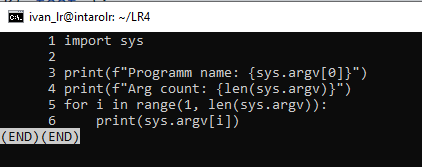


Рисунок 13.11 – Результат запуска программы на Python

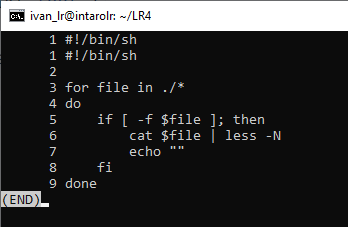


Рисунок 13.12 – Результат запуска программы на Python

8) Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения.

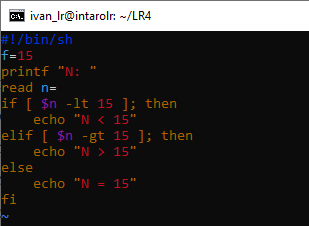


Рисунок 14.1 – Код скрипта compar.sh

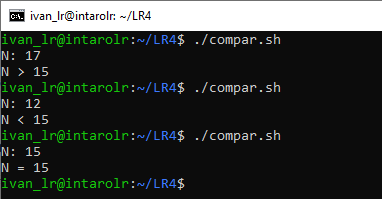


Рисунок 14.2 – Запуск скрипта compar.sh

Код на Python:

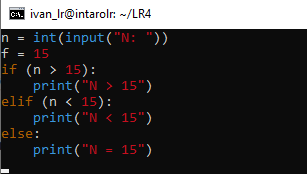


Рисунок 14.3 – Код программы на Python (compar.py)

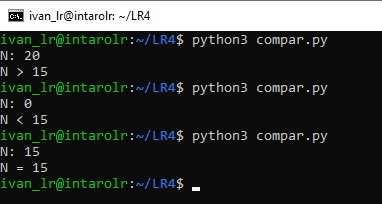


Рисунок 14.4 – Запуск программы на Python (compar.py)

9) Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран.

Рисунок 15.1 – Код скрипта year.sh

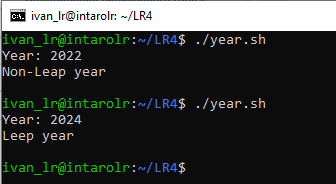
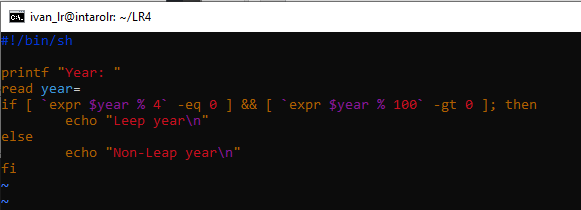


Рисунок 15.2 – Запуск скрипта year.sh

Код на Python:

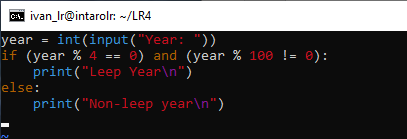


Рисунок 15.3 – Код программы на Python (year.py)

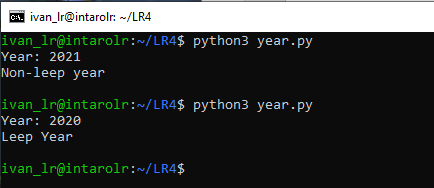


Рисунок 15.4 – Запуск программы на Python (year.py)

10) Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значение инкрементируется.

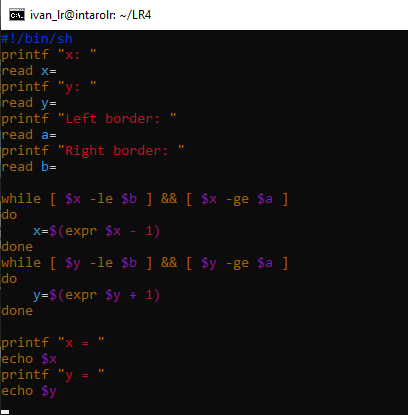


Рисунок 16.1 – Код скрипта diap.sh

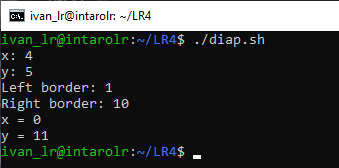


Рисунок 16.2 – Запуск скрипта diap.sh

Код на Python:

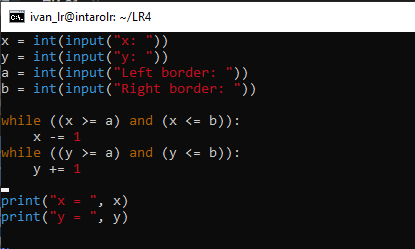


Рисунок 16.3 – Код программы на Python (diap.py)

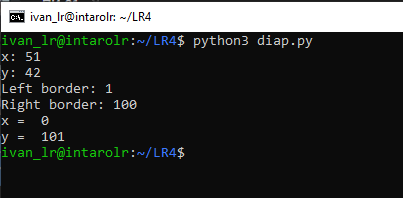


Рисунок 16.4 – Запуск программы на Python (diap.py)

11) В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично в длинном формате с указанием скрытых файлов отображается содержимое каталога /etc.

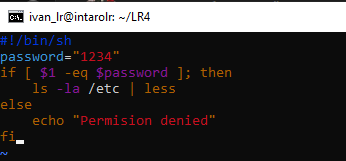


Рисунок 17.1 – Код скрипта pswd.sh

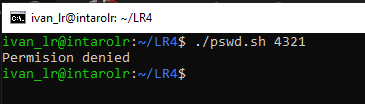


Рисунок 17.2 – Результат запуска скрипта с неверным паролем.

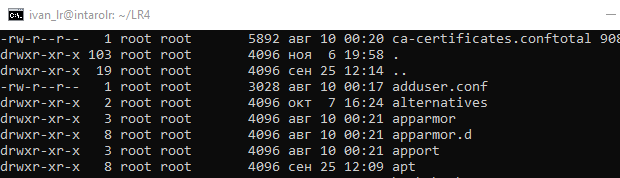


Рисунок 17.3 – Результат запуска скрипта с верным паролем.

Код на Python:

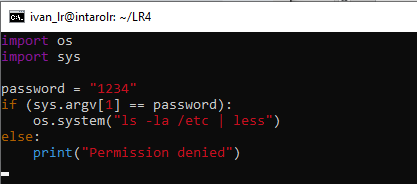


Рисунок 17.4 – Код программы на Python (pswd.py)

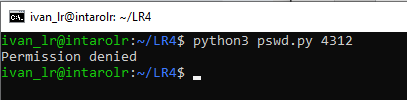


Рисунок 17.5 – Результат запуска программы на Python с неверным паролем

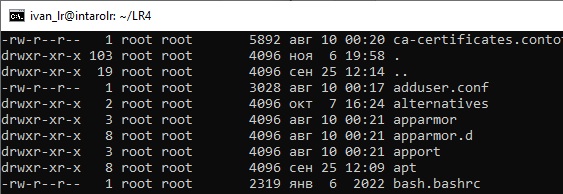


Рисунок 17.6 – Результат запуска программы на Python с верным паролем

12) Проверить существует ли файл. Если да, выводится на экран его содержимое, если нет – выдается соответствующее сообщение.

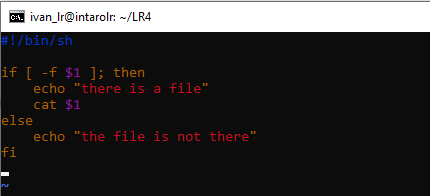


Рисунок 18.1 – Код скрипта checkfile.sh

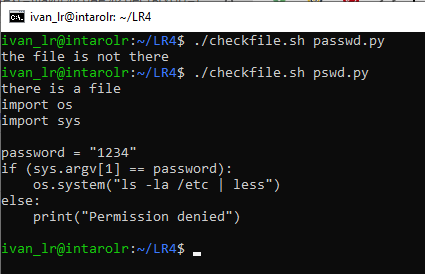


Рисунок 18.2 – Запуск скрипта checkfile.sh

Код на Python:

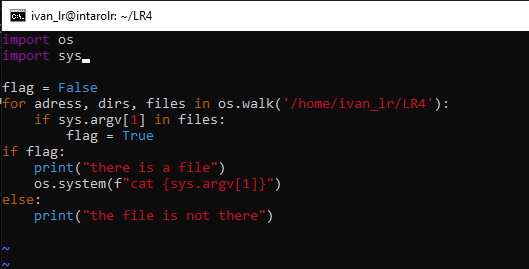


Рисунок 18.3 – Код программы на Python (checkfile.py)

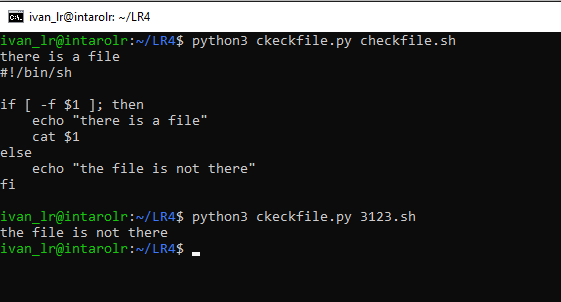


Рисунок 18.4 – Запуск программы на Python (checkfile.py)

13) Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог просматривается содержимое файла.

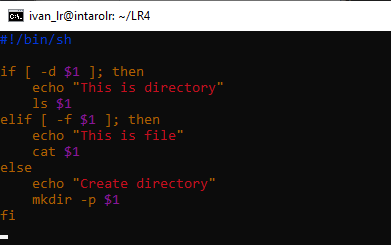


Рисунок 19.1 – Код скрипта checkdir.sh

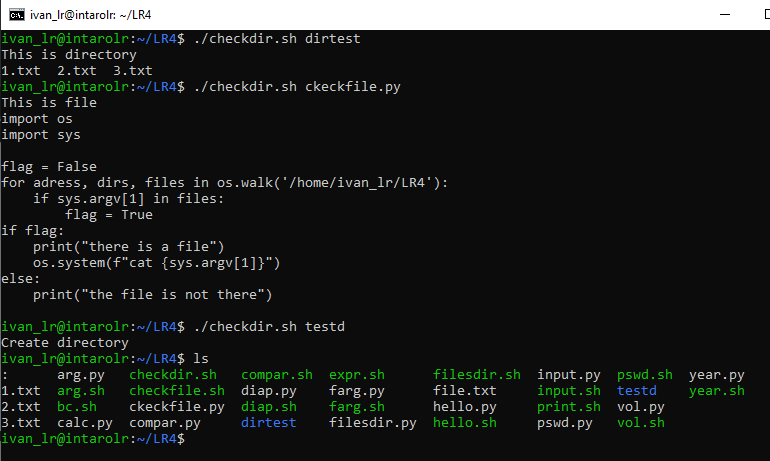


Рисунок 19.2 – Запуск скрипта checkdir.sh

Код на Python:

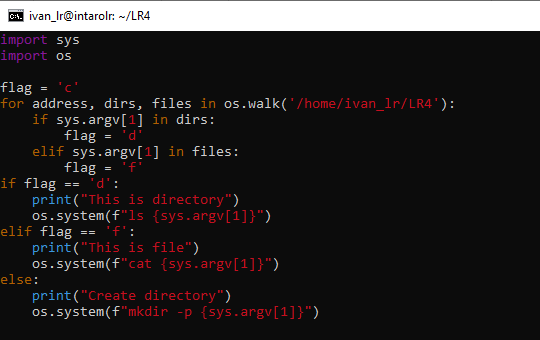


Рисунок 19.3 – Код программы на Python (checkdir.py)

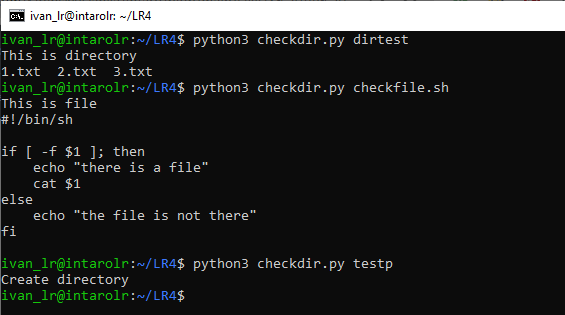


Рисунок 19.4 – Запуск программы на Python (checkdir.py)

14) Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются соответствующие сообщения (использовать имена файлов и/или позиционные параметры).

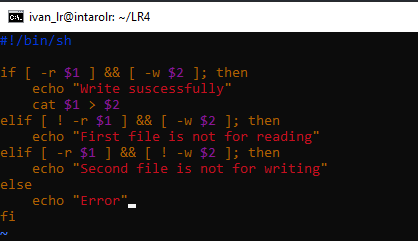


Рисунок 20.1 – Код скрипта analys.sh

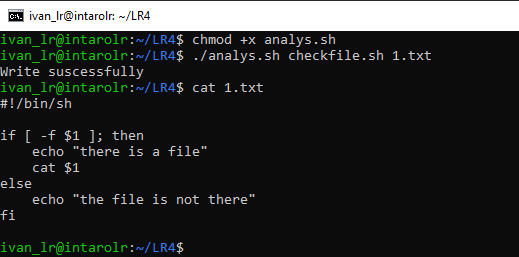


Рисунок 20.2 – Запуск скрипта analys.sh

Код на Python:

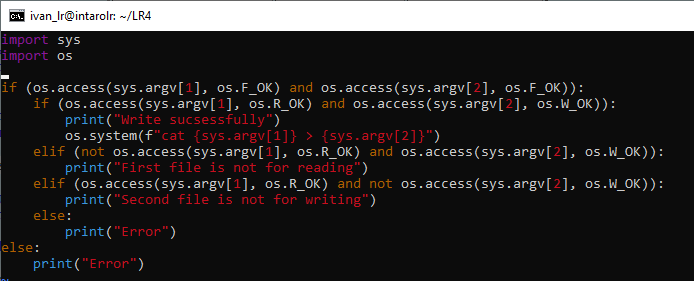


Рисунок 20.3 – Код программы на Python (analys.py)

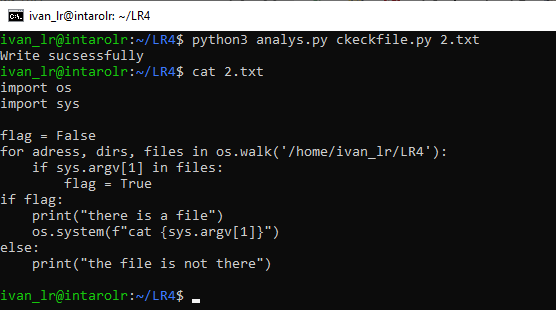


Рисунок 20.4 – Запуск программы на Python (analys.py)

15) Если файл запуска программы найден, программа запускается (по выбору).

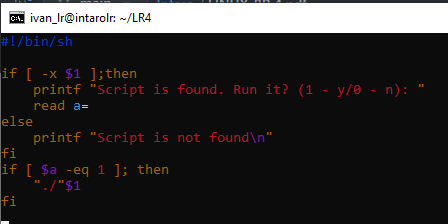


Рисунок 21.1 – Код скрипта xfile.sh

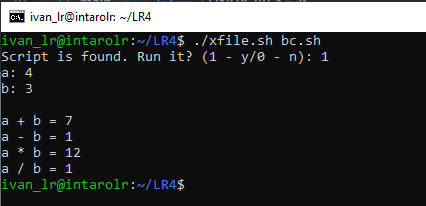


Рисунок 21.2 – Запуск скрипта xfile.sh

Код на Python:

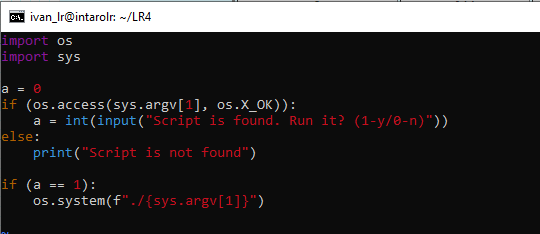


Рисунок 21.3 – Код программы на Python (xfile.py)

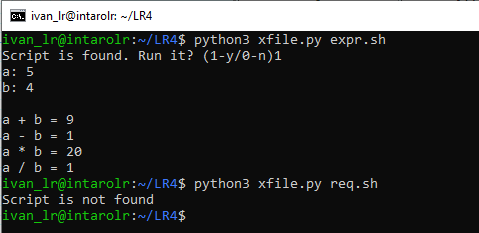


Рисунок 21.4 – Запуск программы на Python (xfile.py)

16) В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля , содержимое файла сортируется по первому столбцу по возрастанию, отсортированная информация помещается в другой файл, содержимое которого затем отображается на экране.

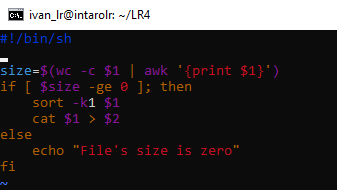


Рисунок 22.1 – Код скрипта sort.sh

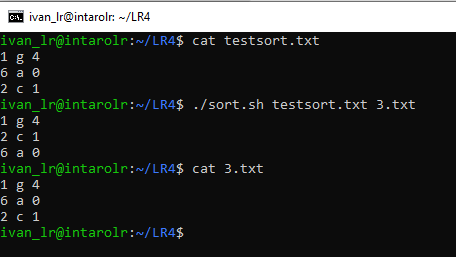


Рисунок 22.2 – Запуск скрипта sort.sh

Код на Python:

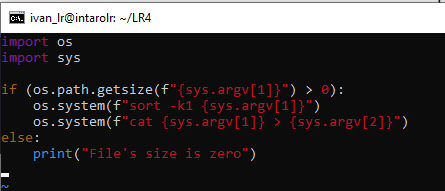


Рисунок 22.3 – Код программы на Python (sort.py)

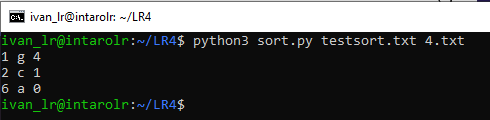


Рисунок 22.4 – Запуск программы на Python (sort.py)

# **Вывод**

По итогу лабораторной работы я изучил основные возможности языка программирования высокого уровня Shell и получил навыки написания и использования скриптов.

# **Ответы на контрольные вопросы**

1) В чем отличие пользовательских переменных от переменных среды?

Пользовательские переменные - это переменные, определенные и использованные конкретным пользователем внутри его рабочей среды.

Переменные среды - это переменные, которые определены на уровне операционной системы и доступны для всех пользователей и приложений, работающих на компьютере.

2) Математические операции в Shell

«+» - сложение (a + b), «-» - вычитание (a - b), «\*» - умножение (a \* b), «/» - деление (a / b), «%» - остаток от деления (a % b), «\*\*» - возведение в степень (a \*\* b).

Пример использования

result=$(($a + $b))

3) Условные операторы Shell

if [ УСЛОВИЕ1 ] ; then

# Блок кода, если условие1 выполнено

elif [ УСЛОВИЕ2 ] ; then

# Блок кода, если условие2 выполнено

else

# Блок кода, если ни одно из описанных условий не выполнено

fi

4) Принципы построения простых и составных условий.

«&&» или «-а» - логическое И; «||» или «-o» - логическое ИЛИ; «=» или «==» - проверка на равенство; «!=» или «-ne» - проверка на неравенство; «-z» - проверяет, что значение переменной пусто; «-n» - проверяет, что значение переменной НЕпустое; также поддерживает все операции сравнения типа больше/меньше/равно: «>», «<», «>=», «<=» или «-gt» (greater than), «-lt» (less than), «-ge» (greater or equal), «-le» (less or equal).

5) Циклы в Shell.

while [ УСЛОВИЕ ] do

# Блок кода, если условие выполнено

done

for ПЕРЕМЕННАЯ in МНОЖЕСТВО\_ЭЛЕМЕНТОВ

# Блок кода, использующий элемент из множества

done

6) Массивы и модули в Shell.

- Определение массива:

myArray=(“value1” “value2” “value3”)

- Доступ к элементу по индексу:

array=(“a” “b” “c”)

echo $(array[0])

Все элементы массива:

echo $(array[@])

Количество элементов в массиве:

echo $(#array[@])

- Удаление элемента массива:

unset array[2]

- Добавление элементов в конец массива:

$array+=(”e” “f”)

7) Чтение параметров командной строки.

Работа с параметрами происходит следующим образом:  
$0 - всегда **имя файла скрипта** (в данном случае script.sh)

$1 - первый позиционный параметр (в данном случае par1)

$2 - второй позиционный параметр (par2) и тд

8) Как различать ключи и параметры?

Перед ключами ставится “-”, а параметры ставятся в конце команды.

Пример: ls –la dir

9) Чтение данных из файлов.

Чтение из файлов происходит с помощью символа “<”

Пример: readarray lines < file.txt

Или с помощью команды cat

10) Стандартные дескрипторы файлов.

- Дескриптор стандартного ввода (stdin) - используется для чтения данных из стандартного входного потока.

- Дескриптор стандартного вывода (stdout) - используется для вывода данных в стандартный поток вывода.

- Дескриптор стандартной ошибки (stderr) - используется для вывода сообщений об ошибках и диагностической информации.

числовые значения:

- stdin: 0

- stdout: 1

- stderr: 2

11) Перенаправление вывода.

ls > result.txt - Перенаправит вывод команды ls в файл result.txt. Если файл существует, его содержимое будет перезаписано.

- ls >> result.txt - Сделает то же, что и «>», но если файл уже существует, результат будет записан в конец файла.

- ls | grep “test” - Перенаправит результат вывода команды ls в команду grep, которая отфильтрует полученный результат и выведет только те результаты, которые содержат слово «test».

12) Подавление вывода.

Команда будет подавлять вывод ошибок и стандартного вывод команды ls:

ls > /dev/null 2>$1

13) Отправка сигналов скриптам.

Скриптам в процессе выполнения можно отправлять сигналы так же, как и другим процессам, например kill, killall и другие.

14) Использование функций.

Синтаксис функций в shell:

function myfunc {

# do smth

}

15) Отправка сообщений в терминал пользователя.

Это происходит с помощью команды echo:

echo “Hello, it is script file $0”

16) BASH и SHELL – синонимы?

SHELL (интерпретатор командной оболочки) - это общий термин, который описывает программу, предоставляющую пользователю интерфейс для взаимодействия с операционной системой. Она обрабатывает команды, вводимые пользователем, и выполняет их.

BASH (Bourne Again SHell) - это одна из наиболее распространенных командных оболочек в UNIX-подобных системах. BASH является расширением исходного кода оригинальной командной оболочки Unix - Bourne shell (sh), и предоставляет дополнительные функции и улучшения.

Таким образом, BASH - это конкретная реализация командной оболочки, в то время как SHELL - это более широкое понятие, описывающее класс программ, осуществляющих взаимодействие пользователя с операционной системой.

17) PowerShell в операционных системах семейства Windows: назначение и особенности.

PowerShell - это мощный интерактивный оболочечный язык и среда командной строки, разработанные компанией Microsoft для операционных систем Windows. Он был выпущен в 2006 году и является продолжением более старой командной оболочки cmd.exe.

Назначение PowerShell заключается в автоматизации задач администрирования и управления компьютером. Он предоставляет мощные средства для создания и выполнения сценариев, управления файлами и папками, настройки и управления службами, установки и удаления программных пакетов, редактирования реестра и многого другого. PowerShell также поддерживает управление удаленными компьютерами и Active Directory.

Особенности:

- Объектно-ориентированность: PowerShell представляет результаты команд в виде объектов, которые можно легко фильтровать, сортировать и передавать на вход других команд.

- Расширяемость: PowerShell поддерживает создание пользовательских модулей, которые добавляют новые команды и функциональность.

- Интеграция с другими технологиями Microsoft: PowerShell может использоваться для автоматизации работы с продуктами Microsoft, такими как SQL Server, Exchange, SharePoint и другими.

- Кросс-платформенность: начиная с версии PowerShell 6.0, Microsoft выпустила кросс-платформенную версию PowerShell, которая работает также на Linux и macOS.