**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

A logo of a triangle

Description automatically generated**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

|  |  |
| --- | --- |
| Институт | компьютерных наук |
| Кафедра | автоматизированных систем управления |

ЛАБОРАТОРНАЯ РАБОТА №4

По дисциплине «Операционные системы Linux»

На тему «Создание и использование сценариев (скриптов) в Linux»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ПИ-22-1 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Пахомов А.А. |
| Руководитель | |  |  |  |
| канд.техн.наук, доцент  ученая степень, ученое звание | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Кургасов В.В. |

Липецк, 2024 г.

**Оглавление**

[**Цель работы** 3](#_Toc182264978)

[**Ход работы** 4](#_Toc182264979)

[**1. Часть I** 4](#_Toc182264980)

[**2. Часть II** 7](#_Toc182264981)

[**Вывод** 17](#_Toc182264982)

[**Контрольные вопросы** 18](#_Toc182264983)

**Цель работы**

Изучить основные возможности высокого уровня Shell и получить навыки написания и использования скриптов.

**Ход работы**

**1. Часть I**

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

На рисунке 1 представлен код программы на Bash для текущего задания. На рисунке 2 показан пример выполнения программы.

**

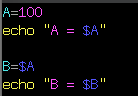
*Рисунок 1*

**

*Рисунок 2*

1.2. Присвоить переменной A целочисленное значение. Просмотреть значение переменной A. Присвоить переменной B значение переменной A. Просмотреть значение переменной B.

На рисунке 3 представлен код программы на Bash для текущего задания. На рисунке 4 показан пример выполнения программы.

**

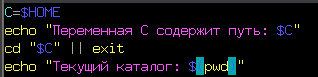
*Рисунок 3*

**

*Рисунок 4*

1.3. Присвоить переменной C значение «путь до своего каталога». Перейти в этот каталог с использованием переменной.

На рисунке 5 представлен код программы на Bash для текущего задания. На рисунке 6 показан пример выполнения программы.

**

*Рисунок 5*

**

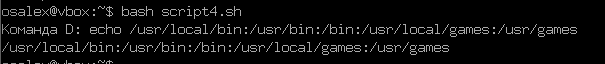
*Рисунок 6*

1.4. Присвоить переменной D значение «имя команды», а именно, команды PATH. Выполнить эту команду, используя значение переменной.

На рисунке 7 представлен код программы на Bash для текущего задания. На рисунке 8 показан пример выполнения программы.

**

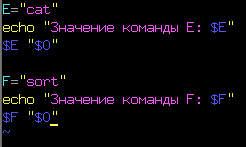
*Рисунок 7*

**

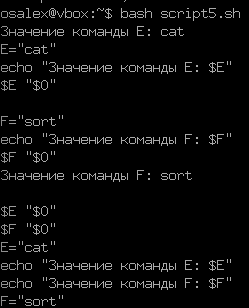
*Рисунок 8*

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

На рисунке 9 представлен код программы на Bash для текущего задания. На рисунке 10 показан пример выполнения программы.

**

*Рисунок 9*

**

*Рисунок 10*

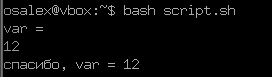
**2. Часть II**

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

На рисунке 11 представлен код программы на Bash для текущего задания. На рисунке 12 показан пример выполнения программы.

**

*Рисунок 11*

**

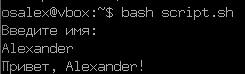
*Рисунок 12*

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

На рисунке 13 представлен код программы на Bash для текущего задания. На рисунке 14 показан пример выполнения программы.

**

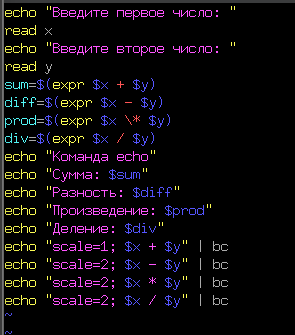
*Рисунок 13*

**

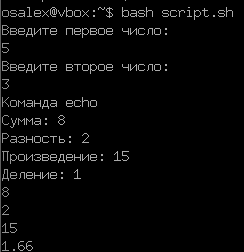
*Рисунок 14*

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

На рисунке 15 представлен код программы на Bash для текущего задания. На рисунке 16 показан пример выполнения программы.

**

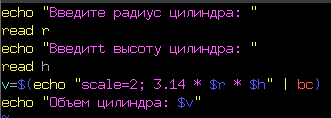
*Рисунок 15*

**

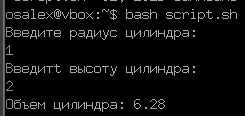
*Рисунок 16*

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

На рисунке 17 представлен код программы на Bash для текущего задания. На рисунке 18 показан пример выполнения программы.

**

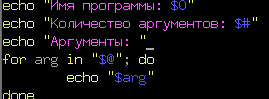
*Рисунок 17*

**

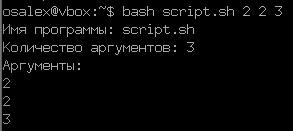
*Рисунок 18*

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

На рисунке 19 представлен код программы на Bash для текущего задания. На рисунке 20 показан пример выполнения программы.

**

*Рисунок 19*

**

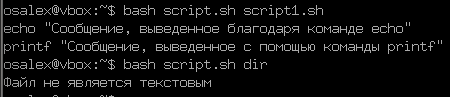
*Рисунок 20*

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

На рисунке 21 представлен код программы на Bash для текущего задания. На рисунке 22 показан пример выполнения программы.

**

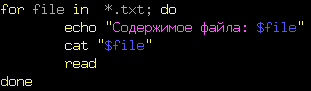
*Рисунок 21*

**

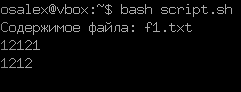
*Рисунок 22*

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

На рисунке 23 представлен код программы на Bash для текущего задания. На рисунке 24 показан пример выполнения программы.

**

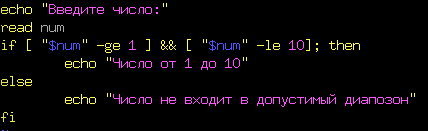
*Рисунок 23*

**

*Рисунок 24*

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

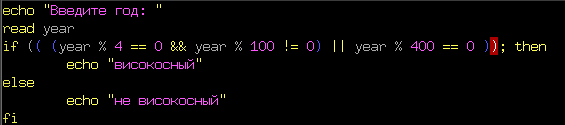
На рисунке 25 представлен код программы на Bash для текущего задания.

**

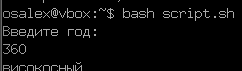
*Рисунок 25*

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

На рисунке 26 представлен код программы на Bash для текущего задания. На рисунке 27 показан пример выполнения программы.

**

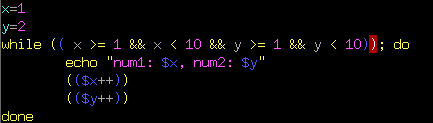
*Рисунок 26*

**

*Рисунок 27*

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

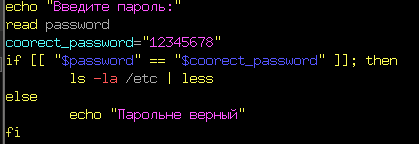
На рисунке 28 представлен код программы на Bash для текущего задания.

**

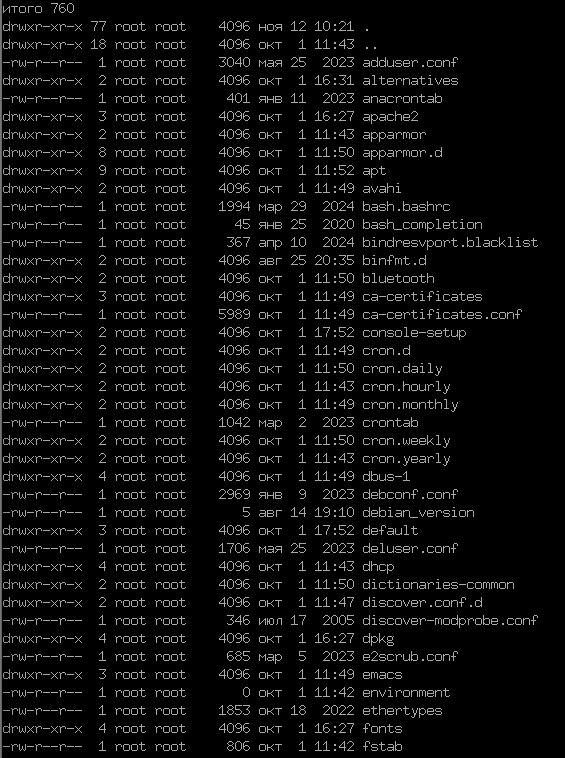
*Рисунок 28*

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

На рисунке 29 представлен код программы на Bash для текущего задания. На рисунке 30 показан пример выполнения программы.

**

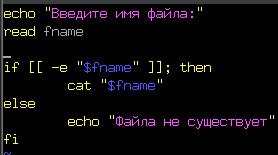
*Рисунок 29*

**

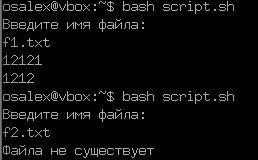
*Рисунок 30*

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

На рисунке 31 представлен код программы на Bash для текущего задания. На рисунке 32 показан пример выполнения программы.

**

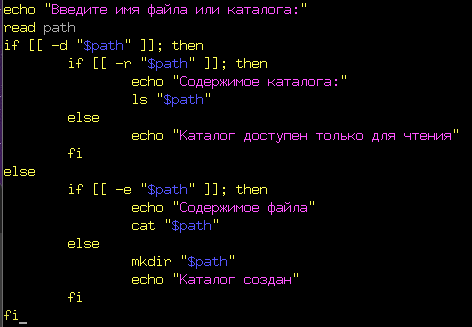
*Рисунок 31*

**

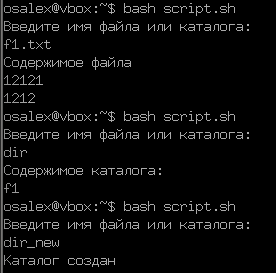
*Рисунок 32*

2.13. 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.

На рисунке 33 представлен код программы на Bash для текущего задания. На рисунке 34 показан пример выполнения программы.

**

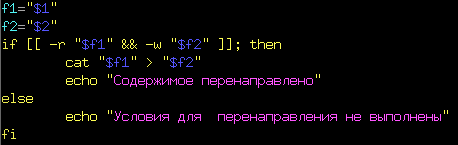
*Рисунок 33*

**

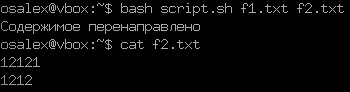
*Рисунок 34*

2.14. 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).

На рисунке 35 представлен код программы на Bash для текущего задания. На рисунке 36 показан пример выполнения программы.

**

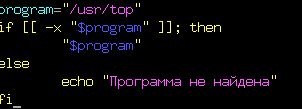
*Рисунок 35*

**

*Рисунок 36*

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

На рисунке 37 представлен код программы на Bash для текущего задания. На рисунке 38 показан пример выполнения программы.

**

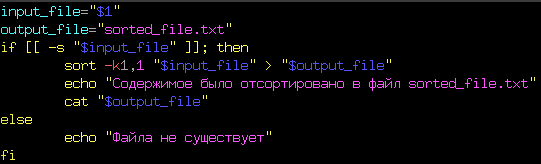
*Рисунок 37*

**

*Рисунок 38*

2.16. 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.

На рисунке 39 представлен код программы на Bash для текущего задания. На рисунке 40 показан пример выполнения программы.

**

*Рисунок 39*

**

*Рисунок 40*

**Вывод**

В ходе выполнения данной лабораторной работы были изучены основные возможности высокого уровня Shell и получены навыки написания и использования скриптов.

**Контрольные вопросы**

1. **В чём отличие пользовательских переменных от переменных среды?**

**Пользовательские переменные** — это переменные, которые задаются в рамках текущей сессии командной оболочки (Shell) и не передаются автоматически другим процессам или скриптам. Они доступны только в текущем Shell или в процессе, где они были созданы.

**Переменные среды** — это переменные, которые задаются в операционной системе и могут быть доступны для всех запущенных процессов и дочерних процессов. Они часто задаются глобально (например, PATH, HOME, USER) и передаются при запуске команд или скриптов. Чтобы пользовательская переменная стала переменной среды, её нужно "экспортировать" командой export.

**2. Математические операции в SHELL.**

В Shell (обычно Bash) для математических операций можно использовать встроенные механизмы, например:

* **Арифметические операции**: выполняются с помощью команды expr, встроенной арифметики $((...)), или команды let.

result=$((5 + 3))

* **Модуль bc**: для работы с числами с плавающей точкой и более сложных операций можно использовать команду bc.

echo "scale=2; 5 / 3" | bc

**3. Условные операторы в SHELL.**

В Shell есть несколько условных операторов:

* if ... then ... fi: основной оператор для выполнения условий.

if [ "$a" -gt "$b" ]; then

echo "a больше b"

fi

* if ... elif ... else ... fi: многоуровневое ветвление.

if [ "$a" -gt "$b" ]; then

echo "a больше b"

elif [ "$a" -lt "$b" ]; then

echo "a меньше b"

else

echo "a равно b"

fi

* case ... in ... esac: используется для проверки нескольких значений переменной.

case "$value" in

"1") echo "Значение один" ;;

"2") echo "Значение два" ;;

\*) echo "Неизвестное значение" ;;

esac

**4. Принципы построения простых и составных условий.**

**Простые условия**: проверка одного условия, как в if [ "$a" -eq 1 ];.

**Составные условия**: используются логические операторы && (и) и || (или).

if [ "$a" -eq 1 ] && [ "$b" -eq 2 ]; then

echo "Оба условия верны"

fi

**Скобки** [[ ... ]] позволяют использовать более сложные выражения и упрощают работу с составными условиями.

**5. Циклы в SHELL.**

Основные типы циклов:

* **for**: проходит по списку значений.

for i in 1 2 3; do

echo "$i"

done

* **while**: выполняется, пока условие истинно.

while [ "$a" -lt 5 ]; do

echo "$a"

a=$((a + 1))

done

* **until**: выполняется, пока условие ложно.

until [ "$a" -ge 5 ]; do

echo "$a"

a=$((a + 1))

done

**6. Массивы и модули в SHELL.**

**Массивы** можно объявить и затем заполнить его элементами.

array=(one two three)

echo ${array[0]} # выводит "one"

Доступ к элементам массива осуществляется с использованием индекса. Все элементы массива выводятся с помощью ${array[@]}.

**Модули**: в Shell можно подключать другие скрипты с помощью source (или .) для повторного использования кода.

source ./script.sh

**7. Чтение параметров командной строки.**

Для работы с параметрами командной строки используются специальные переменные:

* $1, $2, ... — это параметры, переданные скрипту. Например, $1 — первый параметр.
* $# — количество параметров.
* $@ — все параметры в виде списка.

**8. Как различать ключи и параметры?**

Обычно **ключи** начинаются с - или --, чтобы их можно было отличить от других параметров. Например, -f — ключ, а file.txt — параметр.

**9. Чтение данных из файлов.**

Для чтения данных из файла можно использовать следующие методы:

* **Команда cat**: выведет содержимое файла.

cat filename.txt

* **Перенаправление ввода**: через while можно построчно читать файл.

while read line; do

echo "$line"

done < filename.txt

**10. Стандартные дескрипторы файлов.**

В Unix-подобных системах существуют три стандартных дескриптора:

1. **STDIN (стандартный ввод)**, дескриптор 0 – для ввода данных.
2. **STDOUT (стандартный вывод)**, дескриптор 1 – для вывода данных.
3. **STDERR (стандартный вывод ошибок)**, дескриптор 2 – для вывода сообщений об ошибках.

Пример перенаправления:

* command > output.txt — перенаправление вывода STDOUT в файл.
* command 2> error.txt — перенаправление вывода ошибок в файл.
* command > output.txt 2>&1 — перенаправление и STDOUT, и STDERR в один файл.

**11. Перенаправление вывода.**

Перенаправление вывода позволяет направить результат команды в файл, другой процесс или переменную:

* > — перенаправляет стандартный вывод (STDOUT) в файл, перезаписывая его.

echo "Hello" > output.txt

* >> — перенаправляет стандартный вывод в файл, добавляя текст в конец файла.

echo "World" >> output.txt

* 2> — перенаправляет стандартный вывод ошибок (STDERR) в файл.

ls non\_existent\_file 2> error.txt

* &> или >& — перенаправляет оба вывода (STDOUT и STDERR) в файл.

command &> output\_and\_errors.txt

* < — перенаправляет стандартный ввод (STDIN) из файла.

wc -l < input.txt

**12. Подавление вывода.**

Для того чтобы подавить вывод команды, можно перенаправить его в null, что в Unix-системах означает "черная дыра":

* Подавление стандартного вывода:

command > /dev/null

* Подавление стандартного вывода и ошибок:

command &> /dev/null

**13. Отправка сигналов скриптам.**

Для взаимодействия с процессами можно использовать **сигналы**:

* SIGINT (Ctrl+C) — прерывает выполнение процесса.
* SIGTERM — завершает процесс.
* SIGHUP — часто используется для перезапуска процессов.
* SIGKILL — завершает процесс немедленно.

Отправить сигнал можно с помощью команды kill или pkill:

kill -TERM <PID> # отправка SIGTERM процессу

pkill -HUP my\_script.sh # отправка SIGHUP процессу с именем

В скриптах сигналы можно перехватывать через команду trap:

trap 'echo "Прервано!"' SIGINT

**14. Использование функций.**

Функции в Shell позволяют структурировать и переиспользовать код:

function greet() {

echo "Hello, $1!"

}

greet "User"

Функции позволяют передавать аргументы и возвращать значения. Например:

function add() {

result=$(( $1 + $2 ))

echo $result

}

sum=$(add 3 5)

echo "Sum: $sum"

**15. Обработка текстов (чтение, выбор, вставка, замена данных).**

Для обработки текста используются команды:

* **cat** — отображает содержимое файла.
* **grep** — ищет строки, содержащие определенное выражение.

grep "pattern" file.txt

* **sed** — редактирует текст по шаблонам. Например, замена:

sed 's/old/new/g' file.txt

* **awk** — выбирает и обрабатывает текст по строкам и полям. Например, вывод второго столбца:

awk '{print $2}' file.txt

* **cut** — выбирает столбцы или символы.

cut -d ':' -f 1 /etc/passwd

**16. Отправка сообщений в терминал пользователя.**

Команда echo может выводить сообщения в текущий терминал. Для отправки сообщения конкретному пользователю в Unix используют команду write:

write username < message.txt

**17. BASH и SHELL – синонимы?**

**Shell** — это общее название для интерфейса командной строки в Unix-подобных системах, где пользователи могут запускать команды и скрипты. Существует несколько различных оболочек (интерпретаторов команд), таких как:

* **Bash** — одна из наиболее популярных оболочек, используемая в большинстве Linux-систем по умолчанию.
* **Zsh**, **Ksh**, **Tcsh** и другие — альтернативные оболочки с различными возможностями.

Таким образом, **Bash — это конкретная реализация Shell** с расширенными возможностями, а **Shell** — это более общее понятие.

**18. PowerShell в операционных системах семейства Windows: назначение и особенности.**

**PowerShell** — это мощная оболочка командной строки и язык сценариев, разработанный Microsoft. Основные особенности PowerShell:

* **Объектно-ориентированность**: в отличие от Bash, PowerShell оперирует не текстовыми строками, а **объектами** .NET, что позволяет передавать сложные данные между командами.
* **Кроссплатформенность**: PowerShell Core работает не только на Windows, но и на Linux и macOS.
* **Модульная структура**: PowerShell использует модули для расширения возможностей, например, модули для администрирования Active Directory, Azure и других систем.
* **Сильная интеграция с Windows**: PowerShell позволяет управлять системными настройками, реестром и службами Windows.