

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ



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

Клименко Н.Д.

Руководитель

канд.техн.наук, доцент

ученая степень, ученое звание

подпись, дата

Кургасов В.В.

Липецк, 2024 г.

Оглавление

Цель работы.....	3
Ход работы.....	4
1. Часть I.....	4
2. Часть II.....	8
Контрольные вопросы	19
Вывод.....	24

Цель работы

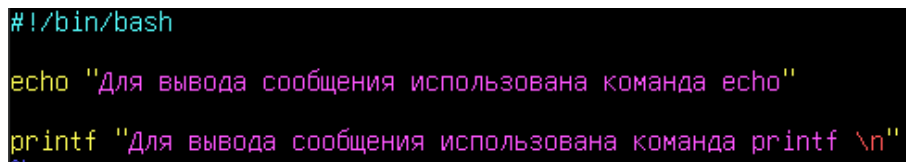
Изучить основные возможности языка программирования высокого уровня Shell. Получить навыки написания и использования скриптов.

Ход работы

1. Часть I

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

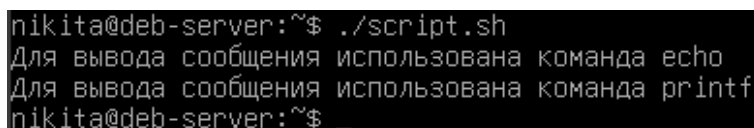
Пример скрипта для задания 1 представлен на рисунке 1.



```
#!/bin/bash
echo "Для вывода сообщения использована команда echo"
printf "Для вывода сообщения использована команда printf \\n"
```

Рисунок 1 – Пример скрипта №1

Пример выполнения скрипта для задания 1 приведен на рисунке 2.

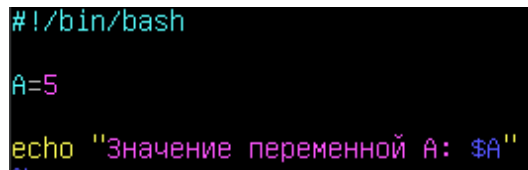


```
nikita@deb-server:~$ ./script.sh
Для вывода сообщения использована команда echo
Для вывода сообщения использована команда printf
nikita@deb-server:~$ _
```

Рисунок 2 – Пример выполнения скрипта №1

1.2. Присвоить переменной А целочисленное значение. Просмотреть значение переменной А.

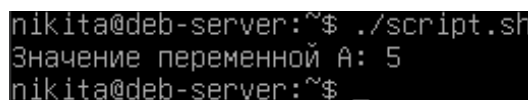
Пример скрипта для задания 2 представлен на рисунке 3.



```
#!/bin/bash
A=5
echo "Значение переменной А: $A"
~
```

Рисунок 3 – Пример скрипта №2

Пример выполнения скрипта для задания 2 приведен на рисунке 4.



```
nikita@deb-server:~$ ./script.sh
Значение переменной А: 5
nikita@deb-server:~$ _
```

Рисунок 4 – Пример выполнения скрипта №2

1.3. Присвоить переменной В значение переменной А. Просмотреть значение переменной В.

Пример скрипта для задания 3 представлен на рисунке 5.

```
#!/bin/bash

A=5
B=$A

echo "Значение переменной B: $B"
~
```

Рисунок 5 – Пример скрипта №3

Пример выполнения скрипта для задания 3 приведен на рисунке 6.

```
nikita@deb-server:~$ ./script.sh
Значение переменной B: 5
nikita@deb-server:~$ _
```

Рисунок 6 – Пример выполнения скрипта №3

1.4. Присвоить переменной C значение "путь до своего каталога". Перейти в этот каталог с использованием переменной.

Пример скрипта для задания 4 представлен на рисунке 7.

```
#!/bin/bash

C=$HOME
echo "Домашний каталог: $C"
cd $C
pwd
~
```

Рисунок 7 – Пример скрипта №4

Пример выполнения скрипта для задания 4 приведен на рисунке 8.

```
nikita@deb-server:~$ ./script.sh
Домашний каталог: /home/nikita
/home/nikita
nikita@deb-server:~$ _
```

Рисунок 8 – Пример выполнения скрипта №4

1.5. Присвоить переменной D значение "имя команды", а именно, команды DATE. Выполнить эту команду, используя значение переменной.

Пример скрипта для задания 5 представлен на рисунке 9.

```
#!/bin/bash

D=date

echo "Команда в переменной D: $D"
$D
~
```

Рисунок 9 – Пример скрипта №5

Пример выполнения скрипта для задания 5 приведен на рисунке 10.

```
nikita@deb-server:~$ ./script.sh
Команда в переменной D: date
Ср 30 окт 2024 18:37:07 MSK
nikita@deb-server:~$ _
```

Рисунок 10 – Пример выполнения скрипта №5

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

Пример скрипта для задания 6 представлен на рисунке 11.

```
#!/bin/bash

E=cat

echo "Команда в переменной E: $E"
echo "Текст временного файла" > file.txt

$E file.txt

rm file.txt
~
```

Рисунок 11 – Пример скрипта №6

Пример выполнения скрипта для задания 6 приведен на рисунке 12.

```
nikita@deb-server:~$ ./script.sh
Команда в переменной E: cat
Текст временного файла
nikita@deb-server:~$ _
```

Рисунок 12 – Пример выполнения скрипта №6

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

Пример скрипта для задания 7 представлен на рисунке 13.

```
#!/bin/bash

F=sort

echo "Команда в переменной F: $F"
echo -e "123\n987\n765\n001" > file.txt

echo "Файл до сортировки: "
cat file.txt

echo "Файл после сортировки: "
$F file.txt
~
```

Рисунок 13 – Пример скрипта №7

Пример выполнения скрипта для задания 7 приведен на рисунке 14.

```
nikita@deb-server:~$ ./script.sh
Команда в переменной F: sort
Файл до сортировки:
123
987
765
001
Файл после сортировки:
001
123
765
987
nikita@deb-server:~$ _
```

Рисунок 14 – Пример выполнения скрипта №7

2. Часть II

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

Пример скрипта для задания 1 представлен на рисунке 15.

```
#!/bin/bash
echo "Введите значение переменной: "
read A
echo "Значение переменной A: $A"
```

Рисунок 15 – Пример скрипта №1

Пример выполнения скрипта для задания 1 приведен на рисунке 16.

```
nikita@deb-server:~$ ./script.sh
Введите значение переменной:
100
Значение переменной A: 100
nikita@deb-server:~$ _
```

Рисунок 16 – Пример выполнения скрипта №1

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

Пример скрипта для задания 2 представлен на рисунке 17.

```
#!/bin/bash
echo "Представьтесь, пожалуйста:"
read name
echo "Здравствуйтесь, $name"
```

Рисунок 17 – Пример скрипта №2

Пример выполнения скрипта для задания 2 приведен на рисунке 18.

```
nikita@deb-server:~$ ./script.sh
Представьтесь, пожалуйста:
nikita
Здравствуйтесь, nikita
nikita@deb-server:~$ _
```

Рисунок 18 – Пример выполнения скрипта №2

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

Пример скрипта для задания 3 представлен на рисунке 19.


```
#!/bin/bash

echo "Введите первое число:"
read a

echo "Введите второе число:"
read b

sum=$(expr $a + $b)
echo "Сумма $a и $b = $sum"

dif=$(expr $a - $b)
echo "Разность $a и $b = $dif"

comp=$(expr $a \* $b)
echo "Произведение $a и $b = $comp"

div=$(echo "scale=4; $a / $b" | bc)
echo "Деление $a на $b = $div"
```

Рисунок 19 – Пример скрипта №3

Пример выполнения скрипта для задания 3 приведен на рисунке 20.

```
nikita@deb-server:~$ ./script.sh
Введите первое число:
12
Введите второе число:
5
Сумма 12 и 5 = 17
Разность 12 и 5 = 7
Произведение 12 и 5 = 60
Деление 12 на 5 = 2.4000
nikita@deb-server:~$
```

Рисунок 20 – Пример выполнения скрипта №3

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

Пример скрипта для задания 4 представлен на рисунке 21.

```
#!/bin/bash

echo "Введите радиус цилиндра:"
read r

echo "Введите высоту цилиндра:"
read h

volume=$(echo "scale=4; 3.14 * $r * $r * $h" | bc)
echo "Объем цилиндра радиусом $r и высотой $h = $volume"
```

Рисунок 21 – Пример скрипта №4

Пример выполнения скрипта для задания 4 приведен на рисунке 22.

```

nikita@deb-server:~$ ./script.sh
Введите радиус цилиндра:
4
Введите высоту цилиндра:
16
Объем цилиндра радиусом 4 и высотой 16 = 803.84
nikita@deb-server:~$

```

Рисунок 22 – Пример выполнения скрипта №4

2.5. Используя позиционный параметр, отобразить имя программы, количество аргументов командой строки, значение каждого аргумента командной строки.

Пример скрипта для задания 5 представлен на рисунке 23.

```

#!/bin/bash

echo "Имя программы: $0"
echo "Количество аргументов: $#"
```



```

i=1
for arg in $@
do
    echo "Аргумент $i: $arg"
    ((i++))
done
~

```

Рисунок 23 – Пример скрипта №5

Пример выполнения скрипта для задания 5 приведен на рисунке 24.

```

nikita@deb-server:~$ ./script.sh arg-1 arg-2 arg-3
Имя программы: ./script.sh
Количество аргументов: 3
Аргумент 1: arg-1
Аргумент 2: arg-2
Аргумент 3: arg-3
nikita@deb-server:~$ _

```

Рисунок 24 – Пример выполнения скрипта №5

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

Пример скрипта для задания 6 представлен на рисунке 25.

```
#!/bin/bash

echo "На вход получен файл: $1"
echo "Содержимое:"
cat $1

sleep 15
clear
```

Рисунок 25 – Пример скрипта №6

Пример выполнения скрипта для задания 6 приведен на рисунке 26.

```
nikita@deb-server:~$ ./script.sh file.txt
На вход получен файл: file.txt
Содержимое:
Текстовый файл для скрипта №6
```

Рисунок 26 – Пример выполнения скрипта №6

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

Пример скрипта для задания 7 представлен на рисунке 27.

```
#!/bin/bash

for file in *.txt
do
    if [ -f $file ] && [ -s $file ]
    then
        echo "Содержимое $file:"
        cat $file | less
    fi
done
```

Рисунок 27 – Пример скрипта №7

Пример выполнения скрипта для задания 7 приведен на рисунке 28.

```
nikita@deb-server:~$ ./script.sh
Содержимое file-1.txt:
Файл №1 для скрипта №7
Содержимое file-2.txt:
Файл №2 для скрипта №7
Содержимое file-3.txt:
Файл №3 для скрипта №7
(END)
```

Рисунок 28 – Пример выполнения скрипта №7

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

Пример скрипта для задания 8 представлен на рисунке 29.

```
#!/bin/bash
number=100
echo "Введите число:"
read number_user

if (($number_user > $number))
then
    echo "указанное число больше допустимого"
elif (($number_user < $number))
then
    echo "указанное число меньше допустимого"
else
    echo "указанное число равно допустимому"
fi
~
```

Рисунок 29 – Пример скрипта №8

Пример выполнения скрипта для задания 8 приведен на рисунке 30.

```
nikita@deb-server:~$ ./script.sh
Введите число:
50
Указанное число меньше допустимого
nikita@deb-server:~$ ./script.sh
Введите число:
150
Указанное число больше допустимого
nikita@deb-server:~$ ./script.sh
Введите число:
100
Указанное число равно допустимому
nikita@deb-server:~$ _
```

Рисунок 30 – Пример выполнения скрипта №8

2.9. Программой запрашивается год, определятся, високосный ли он. Результат выводится на экран.

Пример скрипта для задания 9 представлен на рисунке 31.

```
#!/bin/bash
echo "Введите год:"
read year

if (($year % 4 == 0 && $year % 100 != 0)) || (($year % 100 == 0 && $year % 400 == 0))
then
    echo "Год високосный"
else
    echo "Год не является високосным"
fi
~
```

Рисунок 31 – Пример скрипта №9

Пример выполнения скрипта для задания 9 приведен на рисунке 32.

```

nikita@deb-server:~$ ./script.sh
Введите год:
2024
Год високосный
nikita@deb-server:~$ ./script.sh
Введите год:
2023
Год не является високосным
nikita@deb-server:~$

```

Рисунок 32 – Пример выполнения скрипта №9

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

Пример скрипта для задания 10 представлен на рисунке 33.

```

#!/bin/bash

echo "Введите первое число:"
read a

echo "Введите второе число:"
read b

echo "укажите левую границу"
read left

echo "укажите правую границу"
read right

while true
do
    if (($a < $right && $b < $right && $a > $left && $b > $left))
    then
        ((a++))
        ((b++))
    else
        break
    fi
done
echo "Диапазон [$left; $right]"
echo "a = $a, b = $b"
~

```

Рисунок 33 – Пример скрипта №10

Пример выполнения скрипта для задания 10 приведен на рисунке 34.

```

nikita@deb-server:~$ ./script.sh
Введите первое число:
3
Введите второе число:
10
Укажите левую границу:
2
Укажите правую границу:
15
Диапазон [2; 15]
a = 8, b = 15
nikita@deb-server:~$ _

```

Рисунок 34 – Пример выполнения скрипта №10

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

Пример скрипта для задания 11 представлен на рисунке 35.

```

#!/bin/bash
password="0000"
if [ $1 == $password ]
then
    ls -la /etc | less
else
    echo "Пароли не совпадают"
fi
~

```

Рисунок 35 – Пример скрипта №11

Пример выполнения скрипта для задания 11 приведен на рисунках 36 и 37.

```

nikita@deb-server:~$ ./script.sh 000
Пароли не совпадают
nikita@deb-server:~$ _

```

Рисунок 36– Пример выполнения удачного условия скрипта №11

```

drwxr-xr-x  4 root root    4096 окт  8 10:56 X11
-rw-r--r--  1 root root    681 янв 18 2023 xattr.conf
drwxr-xr-x  4 root root    4096 окт  8 10:56 xdg
(END)

```

Рисунок 37 – Пример выполнения неудачного условия скрипта №11

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

Пример скрипта для задания 12 представлен на рисунке 38.

```
#!/bin/bash
echo "Укажите файл:"
read file

if [ -e $file ]
then
    cat $file
else
    echo "Файл не найден"
fi
~
```

Рисунок 38 – Пример скрипта №12

Пример выполнения скрипта для задания 12 приведен на рисунке 39.

```
nikita@deb-server:~$ ./script.sh
Укажите файл:
file.txt
Текстовый файл для скрипта №12 (Вывод содержимого)
nikita@deb-server:~$ _
```

Рисунок 39 – Пример выполнения скрипта №12

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.

Пример скрипта для задания 13 представлен на рисунке 40.

```
#!/bin/bash
echo "Укажите файл:"
read file

if [ -d $file ]
then
    if [ -r $file ]
    then
        ls -la $file
    else
        echo "Нет прав на чтение каталога"
    fi
elif [ -e $file ]
then
    cat $file
else
    mkdir $file
    echo "Создан каталог: $file"
fi
~
```

Рисунок 40 – Пример скрипта №13

Пример выполнения скрипта для задания 13 приведен на рисунке 41.

```

nikita@deb-server:~$ ./script.sh
Укажите файл:
test-dir
Создан каталог: test-dir
nikita@deb-server:~$ touch /home/nikita/test-dir/file-1
nikita@deb-server:~$ touch /home/nikita/test-dir/file-2
nikita@deb-server:~$ ./script.sh
Укажите файл:
test-dir
итого 8
drwxr-xr-x  2 nikita nikita 4096 ноя  1 11:02 .
drwx----- 4 nikita nikita 4096 ноя  1 11:01 ..
-rw-r--r--  1 nikita nikita   0 ноя  1 11:02 file-1
-rw-r--r--  1 nikita nikita   0 ноя  1 11:02 file-2
nikita@deb-server:~$
nikita@deb-server:~$ ./script.sh
Укажите файл:
/home/nikita/test-dir/file-1
nikita@deb-server:~$
nikita@deb-server:~$ file-1 был пуст, поэтому вывод отсутствует_

```

Рисунок 41 – Пример выполнения скрипта №13

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).

Пример скрипта для задания 14 представлен на рисунке 42.

```

#!/bin/bash

if [ -f $1 ] && [ -r $1 ]
then
    if [ -f $2 ] && [ -w $2 ]
    then
        cat $1 > $2
        echo "Содержимое $1 перенаправлено в $2"
    else
        echo "Второй файл недоступен для записи или не существует"
    fi
else
    echo "Первый файл недоступен для чтения или не существует"
fi

```

Рисунок 42 – Пример скрипта №14

Пример выполнения скрипта для задания 14 приведен на рисунке 43.


```

nikita@deb-server:~$ ./script.sh file-r.txt file-w.txt
Содержимое file-r.txt перенаправлено в file-w.txt
nikita@deb-server:~$ ./script.sh file-r.txt file-none.txt
Второй файл недоступен для записи или не существует
nikita@deb-server:~$ chmod u-r file-r.txt
nikita@deb-server:~$ ./script.sh file-r.txt file-none.txt
Первый файл недоступен для чтения или не существует
nikita@deb-server:~$ _

```

Рисунок 43 – Пример выполнения скрипта №14

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

Пример скрипта для задания 15 представлен на рисунке 44.

```

#!/bin/bash

if [ -f $1 ] && [ -x $1 ]
then
    $1
else
    echo "Файл программы не найден или не является исполняемым"
fi
~

```

Рисунок 44 – Пример скрипта №15

Пример выполнения скрипта для задания 15 приведен на рисунке 45.

```

nikita@deb-server:~$ ./script.sh ./dop-script.sh
Файл запуска программы найден. Скрипт №15 работает
nikita@deb-server:~$ _

```

Рисунок 45 – Пример выполнения скрипта №15

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.

Пример скрипта для задания 16 представлен на рисунке 46.

```

#!/bin/bash

if [ -s $1 ]
then
    echo "Файл до сортировки:"
    cat $1

    sort -k1 $1 > file-out.txt

    echo "Файл после сортировки:"
    cat file-out.txt
else
    echo "Файл пусть или не существует"
fi
~

```

Рисунок 46 – Пример скрипта №16

Пример выполнения скрипта для задания 16 приведен на рисунке 47.

```

nikita@deb-server:~$ ./script.sh file-sort.txt
Файл до сортировки:
Яблоки: 1
Бананы: 2
Апельсины: 4
Виноград: 3
Файл после сортировки:
Апельсины: 4
Бананы: 2
Виноград: 3
Яблоки: 1
nikita@deb-server:~$ _

```

Рисунок 47 – Пример выполнения скрипта №16

Контрольные вопросы

1. В чем отличие пользовательских переменных от переменных среды?

Пользовательские переменные устанавливаются пользователем для текущей оболочки, временно или постоянно. Они определяются для конкретного пользователя и загружаются каждый раз, когда он входит в систему или подключается удалённо. Такие переменные, как правило, хранятся в файлах конфигурации: `.bashrc`, `.bash_profile`, `.bash_login`, `.profile` или в других файлах, размещённых в директории пользователя.

Переменные среды – это переменные, определённые для текущей оболочки и наследуемые любым потомком оболочки и запущенного процесса. Они используются, чтобы пропускать информацию в процессы, которые образуются от самой оболочки. Такие переменные доступны во всей системе, для всех пользователей. Они загружаются при старте системы из системных файлов конфигурации каталога `/etc`.

2. Математические операции SHELL.

Математические операции Shell выполняются с помощью команд:

- `let: res=2+2;`
- оператора `$()`: `res=$((2 + 2))`
- `expr`: `res=$(expr 2 + 2)`
- `bc` (используется для операций с числами с плавающей точкой): `echo "scale=4; 10 / 3" | bc.`

3. Условные операторы SHELL.

Оператор `if` выполняет определённый блок указаний в зависимости от условия. Условие помещают в двойные скобки или квадратные, которые `bash` рассматривает как один элемент с кодом выхода.

Оператор `case` используется, когда необходимо сделать выбор из нескольких альтернатив. Он проверяет значение переменной на совпадение с одним из шаблонов и в случае совпадения выполняет соответствующий блок кода

4. Принципы построения простых и составных условий.

Простые условия проверяют одно утверждение, например `[-f file.txt]`.

Составные условия используют операторы `&&` (логическое "И") и `||` (логическое "ИЛИ") для объединения условий, например `[-f file.txt] && [-r file.txt]`.

5. Циклы в SHELL.

Bash поддерживает циклы `for`, `while` и `until`.

- `for`: позволяет выполнить набор команд для каждого элемента в списке или диапазоне значений;

- `while`: выполняет блок команд до тех пор, пока условие истинно;

- `until`: выполняет блок команд до тех пор, пока условие ложно.

6. Массивы и модули в SHELL.

В `bash` массивы можно объявить как `array=(value1 value2 value3)`. Доступ к элементам массива осуществляется с помощью синтаксиса `${array[index]}`.

Модули можно подключать с помощью команды `source`, что позволяет использовать функции их внешних файлов.

7. Чтение параметров в командной строке.

- `$0`: имя программы;

- `$1`, `$2`, `$3`, `$n`: позиционные параметры, которые соответствуют аргументам командной строки;

- `$@`: список всех аргументов командной строки;

- `$#`: количество аргументов командной строки.

8. Как различать ключи и параметры?

Ключи представляют собой специальные опции, которые изменяют поведение команды или программы. Обычно они указываются с помощью одиночного или двойного дефиса.

Параметры представляют собой значения, передаваемые команде или программе. Они могут быть обязательными или необязательными и указываются после ключей.

9. Чтение данных из файлов.

Для чтения данных из файлов в ОС Linux можно использовать следующие команды:

- `head`: считывает строки из начала файла;

- tail: считывает строки с конца файла;
- cat: выводит содержимое одного или нескольких файлов на экран;
- Также для построчного чтения файла можно использовать команду read в цикле while;
- grep: выполняет поиск строк, соответствующих заданному шаблону;
- awk: позволяет выполнять мощную обработку текста и извлекать нужные данные из файлов.

10. Стандартные дескрипторы файлов.

STDIN – стандартный поток ввода, используется для чтения данных из консоли или другого устройства ввода.

STDOUT – стандартный поток вывода, используется для записи данных в консоль или другое устройство вывода.

STDERR – стандартный вывод ошибок, используется для записи сообщений об ошибках в консоль или другое устройство вывода.

11. Перенаправление вывода.

> – перенаправляет стандартный вывод в файл, создавая или перезаписывая файл.

>> – перенаправляет стандартный вывод в файл, добавляя данные в конец файла.

< – перенаправляет стандартный ввод из файла.

2> – перенаправляет стандартный вывод ошибок в файл.

2>&1 – перенаправляет стандартный вывод ошибок в стандартный вывод.

12. Подавление вывода.

В Bash подавление вывода осуществляется с помощью перенаправления в специальное устройство /dev/null, которое отбрасывает все переданные данные. Чтобы подавить стандартный вывод, команду можно следующим образом: `command > /dev/null`. Для подавления как стандартного вывода, так и вывода ошибок используется: `command &> /dev/null`.

13. Отправка сигналов скриптам.

Для отправки сигналов скриптам используются команды `kill`, `killall`, `pkill`.

- kill: отправляет сигнал конкретному процессу по его идентификатору (PID);

- killall: отправляет сигнал всем процессам с указанным именем;

- pkill: отправляет сигнал процессам, имя которых соответствует шаблону.

14. Использование функций.

В Shell можно определять и использовать собственные функции для группировки и повторного использования блоков кода. Функции могут принимать аргументы и возвращать значения, их можно использовать для выполнения специфических задач в скриптах Shell.

```
# Определение функции
```

```
my_function() {  
    echo "Hello, World!"  
}
```

```
# Вызов функции
```

```
my_function
```

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

Для работы с текстом применяются следующие команды:

- cat file.txt – выводит весь текст из файла;

- grep "шаблон" file.txt – ищет строки, содержащие указанный шаблон;

- awk '{print \$1}' file.txt – извлекает первый столбец из файла.

- cut -d "," -f2 file.csv – выбирает второй столбец из CSV-файла, разделённого запятыми.

- sort file.txt | uniq – выводит уникальные строки из отсортированных данных.

16. Отправка сообщений в терминал пользователя.

Отправить сообщение в терминал пользователя можно через команду echo. Также write и wall позволяют отправлять сообщения другим пользователям, которые авторизованы в системе:

- echo "Сообщение" > /dev/tty1;

- write username;

- wall "Сообщение для всех пользователей"

17. BASH и SHELL – синонимы?

Shell – это программа, которая предоставляет интерфейс между пользователем и операционной системой, позволяя выполнять команды и управлять процессами. Shell запускается, когда пользователь входит в систему, и может быть интерактивным (когда пользователь вводит команды вручную) или использоваться для автоматизации задач с помощью скриптов.

Bash – это одна из реализаций shell, основанная на Bourne Shell и добавляющая ряд улучшений и новых возможностей. Название Bash расшифровывается как Bourne Again Shell, был разработан в проекте GNU и стал стандартом оболочки в большинстве Linux-систем.

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

Windows PowerShell – это кроссплатформенное решение, предназначенное для автоматизации задач. Оно включает оболочку командной строки, скриптовый язык и платформу управления конфигурацией. Позволяет:

- изменять настройки операционной системы;
- управлять службами и процессами;
- настраивать роли и компоненты сервера;
- устанавливать программное обеспечение;
- управлять уже установленным программным обеспечением при помощи специальных интерфейсов;
- встраивать исполняемые компоненты в сторонние программные продукты;
- создавать сценарии для автоматизации задач администрирования;
- работать с файловой системой, реестром Windows, хранилищем сертификатов и тому подобное.

Вывод

В ходе выполнения данной лабораторной работы изучил основные возможности языка программирования высокого уровня Shell. Получил навыки написания и использования скриптов.