

# Липецкий государственный технический университет

Кафедра прикладной математики

## Отчет по лабораторной работе №5 «Программирование в ОС семейства Linux.»

Студент

\_\_\_\_\_

подпись, дата

Комолых Т.О.  
фамилия, инициалы

Группа

ПМ-18

Руководитель

доц., к.п.н. кафедры АСУ  
ученая степень, ученое звание

\_\_\_\_\_

подпись, дата

Кургасов В. В.  
фамилия, инициалы

Липецк 2020 г.

## Содержание

Цель работы	3
Практическое задание	3
Выполнение практического задания.	5
Простые скрипты. . . . .	5
Написание скриптов. . . . .	7
Вывод	26
Список литературы	27

## Цель работы

Изучить основные возможности языка программирования Shell с целью автоматизации процесса администрирования системы за счет написания и использования командных файлов.

## Практическое задание

- 1) Используя команды ECHO, PRINTF, вывести информационные сообщения на экран.
- 2) Присвоить переменной A целочисленное значение. Просмотреть значение переменной A.
- 3) Присвоить переменной B значение переменной A. Просмотреть значение переменной B.
- 4) Присвоить переменной C значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной.
- 5) Присвоить переменной D значение “имя команды”, а именно команды DATE. Выполнить эту команду, используя значение переменной.
- 6) Присвоить переменной E 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 а) имена файлов; б) 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.
- 17) Командой TAR осуществляется сборка всех текстовых файлов текущего каталога в один архивный файл my.tar, после паузы просматривается содержимое файла my.tar, затем командой GZIP архивный файл my.tar сжимается.
- 18) Написать скрипт с использованием функции, например, функции, суммирующей значения двух переменных.

## Выполнение практического задания.

Код скриптов описывается в файле task\_shell.sh, права доступа которого были предварительно приведены к виду 777.

### Простые скрипты.

Используя команды ECHO, PRINTF вывести информационные сообщения на экран.

Код скрипта приведён на рисунке 1.

```
echo "Информационное сообщение!"  
printf "%s \n" "Новое информационное сообщение_"
```

Рисунок 1.

Результат скрипта показан на рисунке 2.

```
tatyana@arianrod:~$ sh task_shell.sh  
Информационное сообщение!  
Новое информационное сообщение
```

Рисунок 2.

Присвоить переменной А целочисленное значение. Просмотреть значение переменной А.

Код скрипта приведён на рисунке 3.

```
A=10;  
echo A = $A
```

Рисунок 3.

Результат скрипта показан на рисунке 4.

```
tatyana@arianrod:~$ sh task_shell.sh  
Информационное сообщение!  
Новое информационное сообщение  
A = 10
```

Рисунок 4.

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

Код скрипта приведён на рисунке 5.

```
B=$A  
echo B = $B_
```

Рисунок 5.

Результат скрипта показан на рисунке 6.

```
tatyana@arianrod:~$ sh task_shell.sh  
Информационное сообщение!  
Новое информационное сообщение  
A = 10  
B = 10
```

Рисунок 6.

Присвоить переменной C значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной.

Код скрипта приведён на рисунке 7.

```
C=$HOME
echo C = $C
echo $PWD

cd $C
echo $PWD
```

Рисунок 7.

Результат скрипта показан на рисунке 8.

```
tatyana@arianrod:~$ sh task_shell.sh
Информационное сообщение!
Новое информационное сообщение
A = 10
B = 10
C = /home/tatyana
/home/tatyana
/home/tatyana
tatyana@arianrod:~$ cd ../
tatyana@arianrod:/home$ ./tatyana/task_shell.sh
Информационное сообщение!
Новое информационное сообщение
A = 10
B = 10
C = /home/tatyana
/home
/home/tatyana
```

Рисунок 8.

Присвоить переменной D значение “имя команды”, а именно команды DATE. Выполнить эту команду, используя значение переменной.

Код скрипта приведён на рисунке 9.

```
D='date'
$D
~
```

Рисунок 9.

Результат скрипта показан на рисунке 10.

```
tatyana@arianrod:~$ sh task_shell.sh
Информационное сообщение!
Новое информационное сообщение
A = 10
B = 10
C = /home/tatyana
/home/tatyana
/home/tatyana
Fri Nov 27 17:28:47 UTC 2020
```

Рисунок 10.

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

Код скрипта приведён на рисунке 11.

```
E='less loop1'
echo Comand = $E
$E
```

Рисунок 11.

Результат скрипта показан на рисунке 12.

```
tatyana@arianrod:~$ sh task_shell.sh
Информационное сообщение!
Новое информационное сообщение
A = 10
B = 10
C = /home/tatyana
/home/tatyana
/home/tatyana
Fri Nov 27 17:36:42 UTC 2020
Comand = less loop1
while true;
do true;
done
```

Рисунок 12.

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

Код скрипта приведён на рисунке 13.

```
F='sort loop1'
$F
```

Рисунок 13.

Результат скрипта показан на рисунке 14.

```
tatyana@arianrod:~$ sh task_shell.sh
Информационное сообщение!
Новое информационное сообщение
A = 10
B = 10
C = /home/tatyana
/home/tatyana
/home/tatyana
Fri Nov 27 17:41:31 UTC 2020
Comand = less loop1
while true;
do true;
done
-----
do true;
done
while true;
```

Рисунок 14.

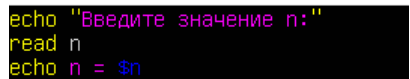
## Написание скриптов.

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

Код скрипта приведён на рисунке 15.

```
echo "Введите значение n:"
```

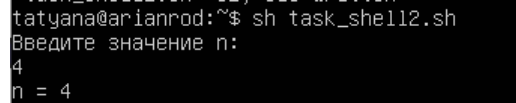
```
read n
echo n = $n
```



```
echo "Введите значение n:"
read n
echo n = $n
```

Рисунок 15.

Результат скрипта показан на рисунке 16.



```
tatyana@arianrod:~$ sh task_shell2.sh
Введите значение n:
4
n = 4
```

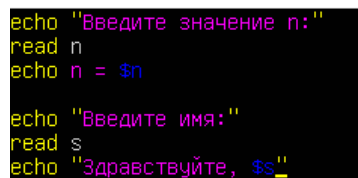
Рисунок 16.

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

Код скрипта приведён на рисунке 17.

```
echo "Введите значение n:"
read n
echo n = $n
```

```
echo "Введите имя:"
read s
echo "Здравствуйте, $s"
```

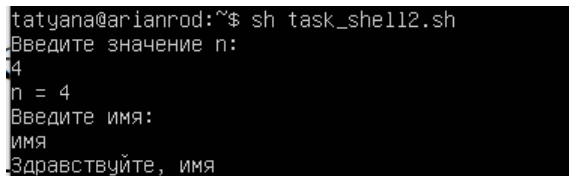


```
echo "Введите значение n:"
read n
echo n = $n

echo "Введите имя:"
read s
echo "Здравствуйте, $s"
```

Рисунок 17.

Результат скрипта показан на рисунке 18.



```
tatyana@arianrod:~$ sh task_shell2.sh
Введите значение n:
4
n = 4
Введите имя:
Имя
Здравствуйте, Имя
```

Рисунок 18.

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

Код скрипта приведён на рисунке 19.

```
printf "Введите значение x = "; read x
printf "Введите значение y = "; read y
```



```

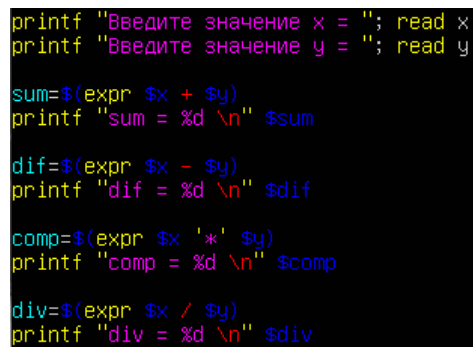
sum=$(expr $x + $y)
printf "sum = %d \n"$sum

dif=$(expr $x - $y)
printf "dif = %d \n"$dif

comp=$(expr $x '*' $y)
printf "comp = %d \n"$comp

div=$(expr $x / $y)
printf "div = %d \n"$div

```



```

printf "Введите значение x = "; read x
printf "Введите значение y = "; read y

sum=$(expr $x + $y)
printf "sum = %d \n" $sum

dif=$(expr $x - $y)
printf "dif = %d \n" $dif

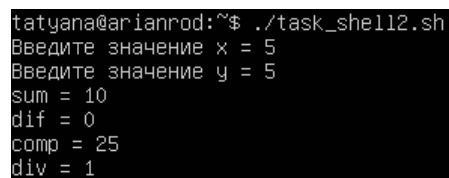
comp=$(expr $x '*' $y)
printf "comp = %d \n" $comp

div=$(expr $x / $y)
printf "div = %d \n" $div

```

Рисунок 19.

Результат скрипта показан на рисунке 20.



```

tatyana@arianrod:~$ ./task_shell2.sh
Введите значение x = 5
Введите значение y = 5
sum = 10
dif = 0
comp = 25
div = 1

```

Рисунок 20.

Код скрипта приведён на рисунке 21.

```

printf "Введите значение x = "; read x
printf "Введите значение y = "; read y

echo "sum = "
echo $x + $y | bc

echo "dif = "
echo $x - $y | bc

echo "comp = "
echo $x '*' $y | bc

echo "div = "
echo $x / $y | bc

```

```

printf "Введите значение x = "; read x
printf "Введите значение y = "; read y

echo "sum = "
echo $x + $y | bc

echo "dif = "
echo $x - $y | bc

echo "comp = "
echo $x '*' $y | bc

echo "div = "
echo $x / $y | bc

```

Рисунок 21.

Результат скрипта показан на рисунке 22.

```

tatyana@arianrod:~$ ./task_shell2.sh
Введите значение x = 5
Введите значение y = 5
sum =
10
dif =
0
comp =
25
div =
1

```

Рисунок 22.

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

Код скрипта приведён на рисунке 23.

```

printf "Введите высоту h = "; read h
printf "Введите радиус r = "; read radius

echo "V = "
echo $h '*' 3.1415926 '*' $radius '*' $radius | bc

```

```

printf "Введите высоту h = "; read h
printf "Введите радиус r = "; read radius

echo "V = "
echo $h '*' 3.1415926 '*' $radius '*' $radius | bc

```

Рисунок 23.

Результат скрипта показан на рисунке 24.

```

tatyana@arianrod:~$ ./task_shell2.sh
Введите высоту h = 5
Введите радиус r = 1
V =
15.7079630

```

Рисунок 24.

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

Код скрипта приведён на рисунке 25.

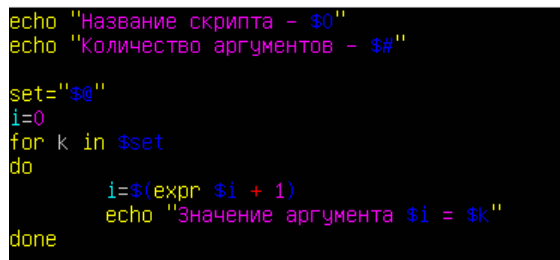
```

echo "Название скрипта - $0"
echo "Количество аргументов - $#"
```

```

set="$@"
i=0
for k in $set
do
i=$(expr $i + 1)
echo "Значение аргумента $i = $k"
done
```



```

echo "Название скрипта - $0"
echo "Количество аргументов - $#"
```

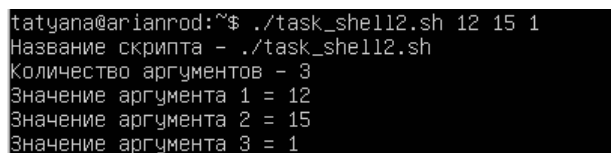
  

```

set="$@"
i=0
for k in $set
do
    i=$(expr $i + 1)
    echo "Значение аргумента $i = $k"
done
```

Рисунок 25.

Результат скрипта показан на рисунке 26.



```

tatyana@arianrod:~$ ./task_shell12.sh 12 15 1
Название скрипта - ./task_shell12.sh
Количество аргументов - 3
Значение аргумента 1 = 12
Значение аргумента 2 = 15
Значение аргумента 3 = 1
```

Рисунок 26.

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

Код скрипта приведён на рисунке 27.

```

less $1
clear
```

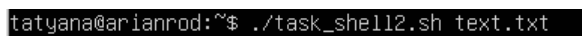


```

less $1
clear
```

Рисунок 27.

На рисунке 28 осуществляется вызов скрипта с указанным параметром.



```

tatyana@arianrod:~$ ./task_shell12.sh text.txt
```

Рисунок 28.

Результат скрипта показан на рисунке 29.

```
Выдра –  
крупный  
зверь  
с  
вытянутым  
гибким  
телом  
обтекаемой  
формы.  
Длина  
тела  
– 55-95  
см,  
хвоста  
26-55  
см,  
масса  
– 6-10  
кг.  
Лапы  
короткие,  
с  
плавательными  
перепонками.  
Хвост  
мускулистый,  
не  
пушистый.  
Половой  
диформизм  
слабо  
выражен:  
около  
50%  
самцов  
крупнее  
text.txt
```

Рисунок 29.

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

Код скрипта приведён на рисунке 30.

```
set=$(ls -l $PWD | awk 'print $9' | grep txt)
for k in $set
do
echo "Название файла - $k"
less $k
done
```

```
set=$(ls -l $PWD | awk '{print $9}' | grep txt)
for k in $set
do
    echo "Название файла - $k"
    less $k
done
```

Рисунок 30.

На рисунке 31 осуществляется вызов скрипта.

```
tatyana@arianrod:~$ ./task_shell2.sh_
```

Рисунок 31.

Результат скрипта на первом экране показан на рисунке 32.

```

Выдра -
крупный
зверь
с
вытянутым
гибким
телом
обтекаемой
формы.
Длина
тела
- 55-95
см,
хвоста
- 26-55
см,
масса
- 6-10
кг.
Лапы
короткие,
с
плавательными
перепонками.
Хвост
мускулистый,

```

Рисунок 32.

Результат скрипта на втором экране показан на рисунке 33.

```

top - 19:20:14 up 3:16, 1 user, load average: 0.03, 0.38, 0.31
Tasks: 97 total, 1 running, 95 sleeping, 1 stopped, 0 zombie
%Cpu(s): 6.2 us, 0.0 sy, 0.0 ni, 93.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 981.3 total, 416.2 free, 137.7 used, 427.4 buff/cache
MiB Swap: 1207.0 total, 1207.0 free, 0.0 used. 691.3 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
    1 root        20   0 167684 11804 8572  S   0.0   1.2   0:01.09 systemd
    2 root        20   0      0      0      0  S   0.0   0.0   0:00.00 kthreadd
    3 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 rcu_gp
    4 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 rcu_par_gp
    6 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 kworker/0:0H-kblockd
    9 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 mm_percpu_wq
   10 root        20   0      0      0      0  S   0.0   0.0   0:00.14 ksoftirqd/0
   11 root        20   0      0      0      0  I   0.0   0.0   0:01.24 rcu_sched
   12 root        rt    0      0      0      0  S   0.0   0.0   0:00.03 migration/0
   13 root       -51   0      0      0      0  S   0.0   0.0   0:00.00 idle_inject/0
   14 root        20   0      0      0      0  S   0.0   0.0   0:00.00 cpuhp/0
   15 root        20   0      0      0      0  S   0.0   0.0   0:00.00 kdevtmpfs
   16 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 netns
   17 root        20   0      0      0      0  S   0.0   0.0   0:00.00 rcu_tasks_kthre
   18 root        20   0      0      0      0  S   0.0   0.0   0:00.00 kauditd
   19 root        20   0      0      0      0  S   0.0   0.0   0:00.00 khungtaskd
   20 root        20   0      0      0      0  S   0.0   0.0   0:00.00 oom_reaper
   21 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 writeback
   22 root        20   0      0      0      0  S   0.0   0.0   0:00.00 kcompactd0
   23 root        25   5      0      0      0  S   0.0   0.0   0:00.00 ksm
   24 root        39  19      0      0      0  S   0.0   0.0   0:00.00 khugepaged
   70 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 kintegrityd
   71 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 kblockd
   72 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 blkcg_punt_bio
   73 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 tpm_dev_wq
   74 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 ata_sff
   75 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 md
   76 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 edac-poller
   77 root         0 -20      0      0      0  I   0.0   0.0   0:00.00 devfreq_wq

```

Рисунок 33.

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

Код скрипта приведён на рисунке 34.

```

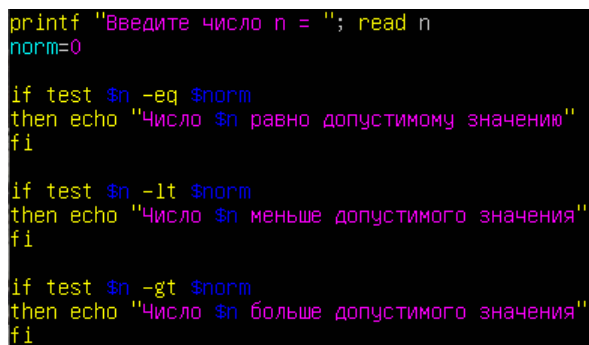
printf "Введите число n = "; read n
norm=0

if test $n -eq $norm
then echo "Число $n равно допустимому значению"
fi

if test $n -lt $norm
then echo "Число $n меньше допустимого значения"
fi

if test $n -gt $norm
then echo "Число $n больше допустимого значения"
fi

```



```

printf "Введите число n = "; read n
norm=0

if test $n -eq $norm
then echo "Число $n равно допустимому значению"
fi

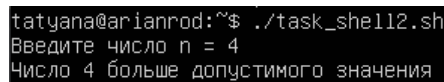
if test $n -lt $norm
then echo "Число $n меньше допустимого значения"
fi

if test $n -gt $norm
then echo "Число $n больше допустимого значения"
fi

```

Рисунок 34.

Результат скрипта показан на рисунке 35.



```

tatyana@arianrod:~$ ./task_shell12.sh
Введите число n = 4
Число 4 больше допустимого значения

```

Рисунок 35.

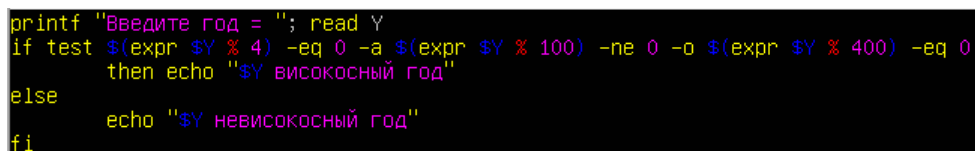
Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран.

Код скрипта приведён на рисунке 36.

```

printf "Введите год = "; read Y
if test $(expr $Y % 4) -eq 0 -a $(expr $Y % 100) -ne 0 -o $(expr $Y % 400) -eq 0
then echo "$Y високосный год"
else
echo "$Y невисокосный год"
fi

```



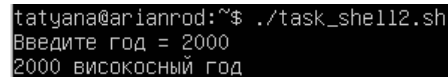
```

printf "Введите год = "; read Y
if test $(expr $Y % 4) -eq 0 -a $(expr $Y % 100) -ne 0 -o $(expr $Y % 400) -eq 0
then echo "$Y високосный год"
else
echo "$Y невисокосный год"
fi

```

Рисунок 36.

Результат скрипта показан на рисунке 37.



```
tatyana@arianrod:~$ ./task_shell12.sh
Введите год = 2000
2000 високосный год
```

Рисунок 37.

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

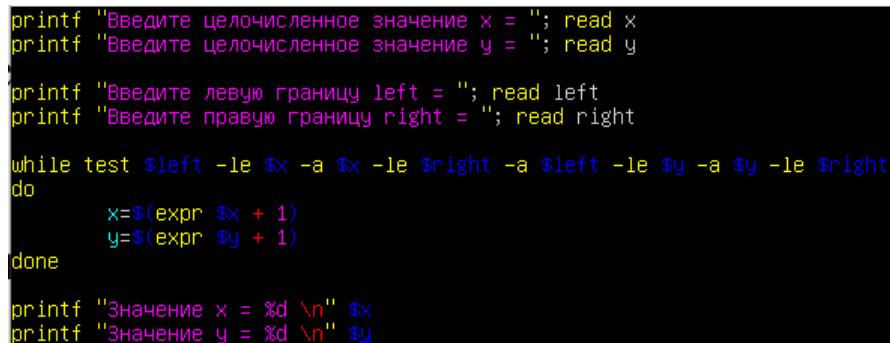
Код скрипта приведён на рисунке 38.

```
printf "Введите целочисленное значение x = "; read x
printf "Введите целочисленное значение y = "; read y

printf "Введите левую границу left = "; read left
printf "Введите правую границу right = "; read right

while test $left -le $x -a $x -le $right -a $left -le $y -a $y -le $right
do
x=$(expr $x + 1)
y=$(expr $y + 1)
done

printf "Значение x = $d \n" $x
printf "Значение y = $d \n" $y
```



```
printf "Введите целочисленное значение x = "; read x
printf "Введите целочисленное значение y = "; read y

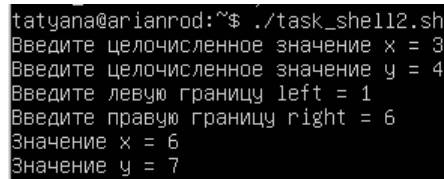
printf "Введите левую границу left = "; read left
printf "Введите правую границу right = "; read right

while test $left -le $x -a $x -le $right -a $left -le $y -a $y -le $right
do
    x=$(expr $x + 1)
    y=$(expr $y + 1)
done

printf "Значение x = %d \n" $x
printf "Значение y = %d \n" $y
```

Рисунок 38.

Результат скрипта показан на рисунке 39.



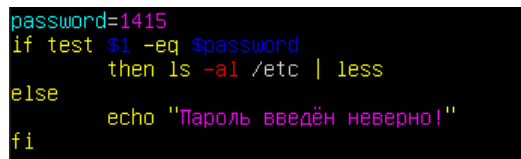
```
tatyana@arianrod:~$ ./task_shell12.sh
Введите целочисленное значение x = 3
Введите целочисленное значение y = 4
Введите левую границу left = 1
Введите правую границу right = 6
Значение x = 6
Значение y = 7
```

Рисунок 39.

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

Код скрипта приведён на рисунке 40.

```
password=1415
if test $1 -eq $password
then ls -al /etc | less
else
echo "Пароль введён неверно!"
fi
```



```
password=1415
if test $1 -eq $password
then ls -al /etc | less
else
echo "Пароль введён неверно!"
fi
```

Рисунок 40.

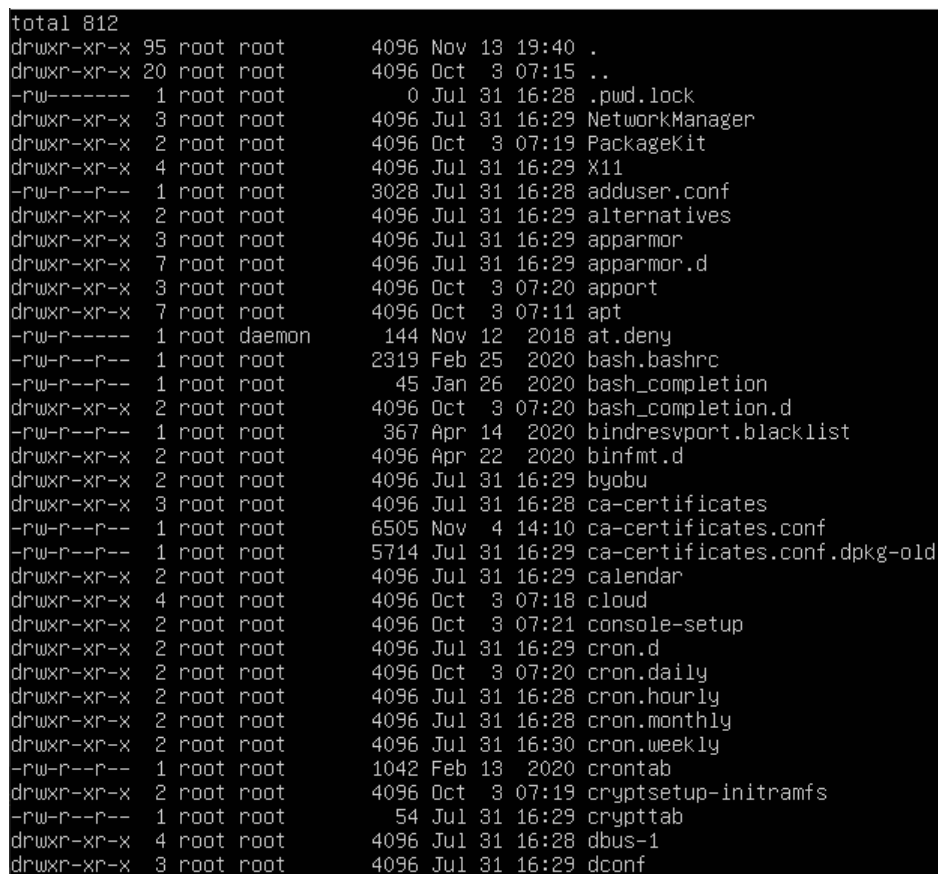
На рисунке 41 осуществляется вызов скрипта с указанным параметром пароля.



```
tatyana@arianrod:~$ ./task_shell2.sh 1415
```

Рисунок 41.

В случае совпадения пароля, введённого пользователем, производится построчное отображение в длинном формате содержимое каталога (рисунок 42).



```
total 812
drwxr-xr-x 95 root root      4096 Nov 13 19:40 .
drwxr-xr-x 20 root root      4096 Oct  3 07:15 ..
-rw----- 1 root root         0 Jul 31 16:28 .pwd.lock
drwxr-xr-x 3 root root      4096 Jul 31 16:29 NetworkManager
drwxr-xr-x 2 root root      4096 Oct  3 07:19 PackageKit
drwxr-xr-x 4 root root      4096 Jul 31 16:29 X11
-rw-r--r-- 1 root root     3028 Jul 31 16:28 adduser.conf
drwxr-xr-x 2 root root      4096 Jul 31 16:29 alternatives
drwxr-xr-x 3 root root      4096 Jul 31 16:29 apparmor
drwxr-xr-x 7 root root      4096 Jul 31 16:29 apparmor.d
drwxr-xr-x 3 root root      4096 Oct  3 07:20 appport
drwxr-xr-x 7 root root      4096 Oct  3 07:11 apt
-rw-r----- 1 root daemon    144 Nov 12 2018 at.deny
-rw-r--r-- 1 root root     2319 Feb 25 2020 bash.bashrc
-rw-r--r-- 1 root root        45 Jan 26 2020 bash_completion
drwxr-xr-x 2 root root      4096 Oct  3 07:20 bash_completion.d
-rw-r--r-- 1 root root       367 Apr 14 2020 bindresvport.blacklist
drwxr-xr-x 2 root root      4096 Apr 22 2020 binfmt.d
drwxr-xr-x 2 root root      4096 Jul 31 16:29 byobu
drwxr-xr-x 3 root root      4096 Jul 31 16:28 ca-certificates
-rw-r--r-- 1 root root     6505 Nov  4 14:10 ca-certificates.conf
-rw-r--r-- 1 root root     5714 Jul 31 16:29 ca-certificates.conf.dpkg-old
drwxr-xr-x 2 root root      4096 Jul 31 16:29 calendar
drwxr-xr-x 4 root root      4096 Oct  3 07:18 cloud
drwxr-xr-x 2 root root      4096 Oct  3 07:21 console-setup
drwxr-xr-x 2 root root      4096 Jul 31 16:29 cron.d
drwxr-xr-x 2 root root      4096 Oct  3 07:20 cron.daily
drwxr-xr-x 2 root root      4096 Jul 31 16:28 cron.hourly
drwxr-xr-x 2 root root      4096 Jul 31 16:28 cron.monthly
drwxr-xr-x 2 root root      4096 Jul 31 16:30 cron.weekly
-rw-r--r-- 1 root root     1042 Feb 13 2020 crontab
drwxr-xr-x 2 root root      4096 Oct  3 07:19 cryptsetup-initramfs
-rw-r--r-- 1 root root        54 Jul 31 16:29 crypttab
drwxr-xr-x 4 root root      4096 Jul 31 16:28 dbus-1
drwxr-xr-x 3 root root      4096 Jul 31 16:29 dconf
```

Рисунок 42.

Иначе выводится сообщение о том, что пароль был введён не верно (рисунок 43).



```
tatyana@arianrod:~$ ./task_shell12.sh 1416
Пароль введён неверно!
```

Рисунок 43.

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

Код скрипта приведён на рисунке 44.

```
file=$PWD'/'$1

if test -e $file
then less $file
else
echo "Такой файл не существует!"
fi
```

```
file=$PWD'/'$1
if test -e $file
then less $file
else
    echo "Такой файл не существует!"
fi
```

Рисунок 44.

На рисунке 45 осуществляется вывод дерева для демонстрации существующих файлов.

```
tatyana@arianrod:~$ tree
.
├── !
├── loop1
├── snap
│   └── tree
│       ├── 18
│       ├── common
│       └── current -> 18
├── task_shell.sh
├── task_shell12.sh
├── text.txt
└── top-output.txt
```

Рисунок 45.

На рисунке 46 переданное в качестве параметра название файла не было найдено.

```
tatyana@arianrod:~$ ./task_shell12.sh text1.txt
Такой файл не существует!
```

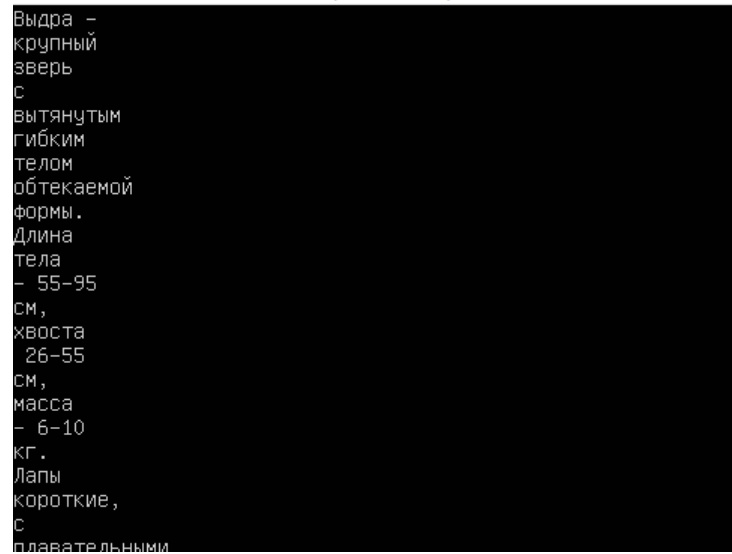
Рисунок 46.

На рисунке 47 переданное в качестве параметра название файла было найдено.

```
tatyana@arianrod:~$ ./task_shell12.sh text.txt
```

Рисунок 47.

Содержимое данного файла представлено на рисунке 48.



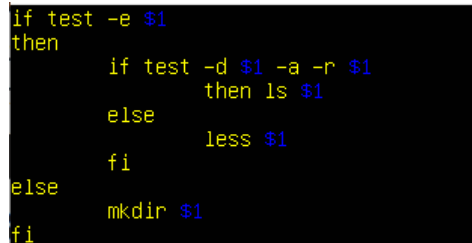
Выдра –  
крупный  
зверь  
с  
вытянутым  
гибким  
телом  
обтекаемой  
формы.  
Длина  
тела  
– 55-95  
см,  
хвоста  
– 26-55  
см,  
масса  
– 6-10  
кг.  
Лапы  
короткие,  
с  
плавательными

Рисунок 48.

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.

Код скрипта приведён на рисунке 49.

```
if test -e $1
then
if test -d $1 -a -r $1
then ls $1
else
less $1
fi
else
mkdir $1
fi
```



```
if test -e $1
then
    if test -d $1 -a -r $1
    then ls $1
    else
        less $1
    fi
else
    mkdir $1
fi
```

Рисунок 49.

```
tatyana@arianrod:~$ tree
.
├── !
├── loop1
├── snap
│   └── tree
│       ├── 18
│       ├── common
│       └── current -> 18
├── task_shell1.sh
├── task_shell12.sh
├── text.txt
└── top-output.txt

5 directories, 6 files
```

Рисунок 50.

Так как файл отсутствовал в (рисунок 50) результатом работы скрипта он был создан, что можно пронаблюдать на рисунке 51.

```
tatyana@arianrod:~$ ./task_shell12.sh text1.txt
tatyana@arianrod:~$ tree
.
├── !
├── loop1
├── snap
│   └── tree
│       ├── 18
│       ├── common
│       └── current -> 18
├── task_shell1.sh
├── task_shell12.sh
├── text.txt
├── text1.txt
└── top-output.txt

6 directories, 6 files
```

Рисунок 51.

На рисунке 52 осуществляется вызов скрипта.

```
tatyana@arianrod:~$ ./task_shell12.sh text.txt
```

Рисунок 52.

Так как файл существует, то производится просмотр его содержимого (рисунок 53).

```

Выдра -
крупный
зверь
с
вытянутым
гибким
телом
обтекаемой
формы.
Длина
тела
- 55-95
см,
хвоста
- 26-55
см,
масса
- 6-10
кг.
Лапы
короткие,
с
плавательными
перепонками.
Хвост
мускулистый,
не
пушистый.
Половой
диформизм
слабо

```

Рисунок 53.

На рисунке 54 производится просмотр содержимого каталога snap.

```

tatyana@arianrod:~$ ./task_shell2.sh snap
tree

```

Рисунок 54.

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 а) имена файлов; б) позиционные параметры).

Код скрипта приведён на рисунке 55.

```

if test -e $1 -a -r $1
then
if test -e $2 -a -r $2
then
cat $1 > $2
echo "$1 был записан в $2"
else
if test -e $2
then
echo "$2 недоступен для записи"
else
echo "$2 не существует"
fi
fi
else

```

```

if test -e $1
then
echo "$1 недоступен для чтения"
else
echo "$1 не существует"
fi
fi

```

```

if test -e $1 -a -r $1
then
    if test -e $2 -a -w $2
    then
        cat $1 >> $2
        echo "$1 был записан в $2"
    else
        if test -e $2
        then
            echo "$2 недоступен для записи"
        else
            echo "$2 не существует"
        fi
    fi
fi
else
    if test -e $1
    then
        echo "$1 недоступен для чтения"
    else
        echo "$1 не существует"
    fi
fi

```

Рисунок 55.

На рисунке 56 представлен результат работы скрипта. Текстовый файл не был обнаружен.

```

tatyana@arianrod:~$ ./task_shell2.sh text2 text3
text2 не существует

```

Рисунок 56.

Чтобы убедиться в этом производится вывод дерева с файлами (рисунок 57).

```

tatyana@arianrod:~$ tree
.
├── !
├── loop1
├── snap
│   └── tree
│       ├── 18
│       ├── common
│       └── current -> 18
├── task_shell.sh
├── task_shell2.sh
├── text.txt
├── text1
└── top-output.txt

5 directories, 7 files

```

Рисунок 57.

На рисунке 58 представлен результат работы скрипта. Текстовый файл не был обнаружен.

```

tatyana@arianrod:~$ ./task_shell2.sh text1 text3
text3 не существует

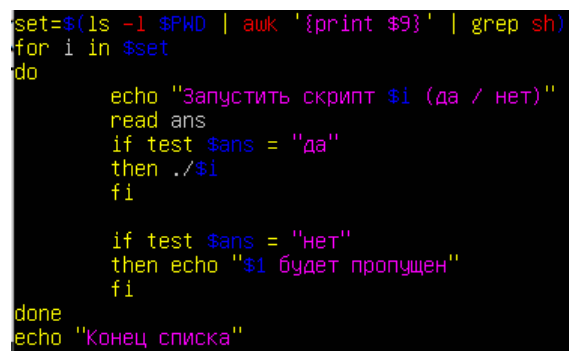
```

Рисунок 58.

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

Код скрипта приведён на рисунке 59.

```
set=$(ls -l $PWD | awk '{print $9}' | grep sh)
for i in $set
do
echo "Запустить скрипт $i (да/нет)"
read ans
if test $ans = "да"
then ./$i
fi
if test $ans = "нет"
then echo "$i будет пропущен"
fi
done
echo "Конец списка"
```

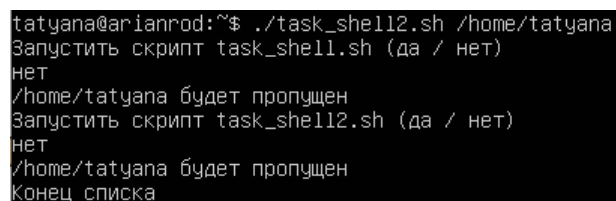


```
set=$(ls -l $PWD | awk '{print $9}' | grep sh)
for i in $set
do
    echo "Запустить скрипт $i (да / нет)"
    read ans
    if test $ans = "да"
    then ./$i
    fi

    if test $ans = "нет"
    then echo "$i будет пропущен"
    fi
done
echo "Конец списка"
```

Рисунок 59.

На рисунках 60, 61 представлены результаты работы скрипта. Пользователю циклично производится обращение с вопросом на запуск найденного файла запуска программы.



```
tatyana@arianrod:~$ ./task_shell2.sh /home/tatyana
Запустить скрипт task_shell.sh (да / нет)
нет
/home/tatyana будет пропущен
Запустить скрипт task_shell2.sh (да / нет)
нет
/home/tatyana будет пропущен
Конец списка
```

Рисунок 60.

```
tatyana@arianrod:~$ ./task_shell12.sh /home/tatyana
Запустить скрипт task_shell.sh (да / нет)
да
Информационное сообщение!
Новое информационное сообщение
A = 10
B = 10
C = /home/tatyana
/home/tatyana
/home/tatyana
Fri Nov 27 21:54:17 UTC 2020
Comand = less loop1
while true;
do true;
done
-----
do true;
done
while true;
Запустить скрипт task_shell12.sh (да / нет)
нет
/home/tatyana будет пропущен
Конец списка
```

Рисунок 61.

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.

Код скрипта приведён на рисунке 62.

```
file=$PWD'/'$1
if test -s $file
then
sort -k1 $file > $PWD'/'prom.txt
less $PWD'/'prom.txt
fi
```

```
file=$PWD'/'$1
if test -s $file
then
    sort -k1 $file_> $PWD'/'prom.txt
    less $PWD'/'prom.txt
fi
```

Рисунок 62.

Результат скрипта показан на рисунке 63.

```
tatyana@arianrod:~$ ./task_shell2.sh task_shell.sh

$D
$E
$F
A=10
B=$A
C=$HOME
D='date'
E='less loop1'
F='sort loop1'
cd $C
echo "-----"
echo "Информационное сообщение!"
echo $PWD
echo $PWD
echo A = $A
echo B = $B
echo C = $C
echo Command = $E
printf "%s \n" "Новое информационное сообщение"
/home/tatyana/prom.txt (END)
```

Рисунок 63.

На рисунке 64 можно заметить, что появился файл prom.txt.

```
tatyana@arianrod:~$ ls
'!'      loop1      prom.txt   task_shell.sh  text.txt      top-output.txt
arch.gz  my.tar     snap       task_shell2.sh text1
```

Рисунок 64.

Командой TAR осуществляется сборка всех текстовых файлов текущего каталога в один архивный файл my.tar, после паузы просматривается содержимое файла my.tar, затем командой GZIP архивный файл my.tar сжимается.

Код скрипта приведён на рисунке 65.

```
s=$(ls -l $PWD | awk '{ print "$PWD"/"$9" ' | grep txt)
tar -cvf my.tar $s
tar -tvf my.tar
echo "Архив заполнен"
gzip my.tar > my.tar.gz
echo "Архив Сжат"
```

```
s=$(ls -l $PWD | awk '{ print "$PWD"/"$9" ' | grep txt)
tar -cvf my.tar $s

tar -tvf my.tar
echo "Архив заполнен"

gzip my.tar > my.tar.gz
echo "Архив сжат"
```

Рисунок 65.

Результат скрипта показан на рисунке 66.



```
tatyana@arianrod:~$ ./task_shell12.sh
tar: Removing leading '/' from member names
/home/tatyana/prom.txt
tar: Removing leading '/' from hard link targets
/home/tatyana/text.txt
/home/tatyana/top-output.txt
-rw-rw-r-- tatyana/tatyana 313 2020-11-27 23:13 home/tatyana/prom.txt
-rwsrwxrwx user/tatyana 683 2020-11-11 17:12 home/tatyana/text.txt
-rw-r--r-- root/root 8405 2020-11-13 19:28 home/tatyana/top-output.txt
Архив заполнен
gzip: my.tar.gz already exists; do you wish to overwrite (y or n)? y
Архив сжат
```

Рисунок 66.

Написать скрипт с использованием функции, например, функции, суммирующей значения двух переменных.

Код скрипта приведён на рисунке 67.

```
multiply(){
echo $1 '*' $2 '*' $3 | bc
}
result=$(multiply $1 $2 2)
echo "Удвоенное произведение чисел равно $result"
```

```
multiply() {
    echo $1 '*' $2 '*' $3 | bc
}
result=$(multiply $1 $2 2)
echo "Удвоенное произведение чисел равно $result"
```

Рисунок 67.

Результат скрипта показан на рисунке 68.

```
tatyana@arianrod:~$ ./task_shell12.sh 3 6
Удвоенное произведение чисел равно 36
```

Рисунок 68.

## Вывод

В ходе лабораторной работы были изучены основные возможности языка программирования Shell с целью автоматизации процесса администрирования системы.

## Список литературы

- [1] Львовский, С.М. Набор и верстка в системе  $\text{\LaTeX}$  [Текст] / С.М. Львовский. М.: МЦНМО, 2006. — 448 с.