

Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине «ОС Linux»

Программирование на SHELL. Использование командных файлов

Студент

Печенкин Д.В.

Группа ПИ-18

Руководитель

Доцент

Кургасов В.В.

Липецк 2020г

Цель работы

Изучение основных возможностей языка программирования Shell с целью автоматизации процесса администрирования системы за счет написания и использования командных файлов.

Задание кафедры

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.
8. Программа запрашивает значение переменной, а затем выводит значение этой переменной.
9. Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной.
10. Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды а) EXPR; б) BC),.
11. Вычислить объем цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран.
12. Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки.

13. Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.
14. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно.
15. Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения.
16. Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран.
17. Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются.
18. В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc.
19. Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение.
20. 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.
21. 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).
22. Если файл запуска программы найден, программа запускается (по выбору).

23. 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.
24. 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.
25. Написать скрипт с использованием функции, например, функции, суммирующей значения двух переменных.

Оглавление

Цель работы	2
Задание кафедры	3
Выполнение работы	9
1. Вывести информационные сообщения на экран.....	9
2. Присвоить переменной целочисленное значение. Просмотреть значение переменной.....	9
3. Присвоить переменной В значение переменной А. Просмотреть значение переменной В	9
4. Присвоить переменной значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной.....	10
5. Присвоить переменной значение “имя команды”, а именно, команды DATE. Выполнить эту команду, используя значение переменной	10
6. Присвоить переменной значение “имя команды”, а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной	11
7. Присвоить переменной значение “имя команды”, а именно сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.....	12
8. Программа запрашивает значение переменной, а затем выводит значение этой переменной	12
9. Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной	13
10. Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды а) EXPR; б) BC	13
11. Вычислить объем цилиндра. Исходные данные запрашиваются	

программой. Результат выводится на экран	14
12. Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки	15
13. Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.....	15
14. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно	16
15. Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения	17
16. Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран	18
17. Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются.....	19
18. В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc	20
19. Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение.....	21
20. 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	22
21. 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).....	23
22. Если файл запуска программы найден, программа запускается (по выбору).....	24
23. 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	25
24. 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.....	27
25. Написать скрипт с использованием функций, например, функции, суммирующей значения двух переменных.....	28
Вывод.....	29

Выполнение работы

1. Вывести информационные сообщения на экран

Команда echo отвечает за вывод сообщения на экран, а “Hello, world” – само сообщение, которое выведет команда. Аналогичная ситуация с командой printf

```
danibrogue@jorgserver:~$ echo Hello, world  
Hello, world  
danibrogue@jorgserver:~$ printf 'Hello, world\n'  
Hello, world  
danibrogue@jorgserver:~$ _
```

Рисунок 1 – Вывод сообщения на экран

2. Присвоить переменной целочисленное значение. Просмотреть значение переменной

Код скрипта:

```
A=5 # Присваиваем переменной A значение 5.  
echo $A # Выводим значение переменной.
```

Знак \$ позволяет обратиться к переменной, которой присвоено значение.

```
root@jorgserver:/home/danibrogue# sh script  
5  
root@jorgserver:/home/danibrogue# _
```

Рисунок 2 – Вывод значения переменной на экран

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

Код скрипта:

```
A=5 #Присваивание переменной А значения 25  
B=$A #Присваивание переменной В значения А  
echo $B #Вывод на экран значение переменной В
```

```
root@jorgserver:/home/danibrogue# sh script  
5  
root@jorgserver:/home/danibrogue# _
```

Рисунок 3 – Вывод значения переменной В на экран

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

Код скрипта:

```
C="/home" #Присваивание переменной пути до каталога в  
который необходимо перейти.
```

```
cd $C #Команда перехода в каталог, который записан в  
переменной C.
```

```
root@jorgserver:/home/danibrogue# . lab  
root@jorgserver:/home#
```

Рисунок 4 – Переход в другой каталог

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

Код скрипта:

```
D= "date" #присваивание переменной D имя команды "date"  
echo `\$D` #Вывод выполнения команды, заложенной в  
переменную D
```

Обратные кавычки `` позволяют присваивать переменным данные вывода системных команд. Символы, заключенные в обратные кавычки, воспринимаются интерпретатором shell как системная команда, которую необходимо выполнить.

```
root@jorgserver:/home/danibrogue# . lab  
Wed 25 Nov 2020 04:39:10 PM UTC  
root@jorgserver:/home/danibrogue# _
```

Рисунок 5 – Вывод даты на консоль

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

Код скрипта:

```
E="cat" #присваивание переменной E имя команды "cat"
```

```
echo $E #вывод на экран имени команды, которая присвоена переменной E
```

```
$E lab #выполнение команды "cat" и вывод на экран
```

```
root@jorgserver:/home/danibrogue# . lab
cat
E="cat"
echo $E
$E lab
root@jorgserver:/home/danibrogue# _
```

Рисунок 6 – Выполнение скрипта №6

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

Код скрипта:

```
F="sort" #присваивание переменной F имя команды "sort"  
$F text #выполнение команды  
cat text #вывод результата
```

```
root@jorgserver:/home/danibrogue# cat text  
Edinburgh  
Phoenix  
New York  
Los Angeles  
root@jorgserver:/home/danibrogue# . lab  
Edinburgh  
Los Angeles  
New York  
Phoenix  
root@jorgserver:/home/danibrogue#
```

Рисунок 7 – Содержимое файла ForNum7.txt

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

Код скрипта:

```
read A #ожидание ввода значения A. После нажатия Enter,  
записанное значение присвоит переменная A.
```

```
echo $A #Вывод введенной переменной
```

```
danibrogue@jorgserver:~$ . lab  
Something  
Something  
danibrogue@jorgserver:~$ _
```

Рисунок 8 – Пример выполнения скрипта

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

Код скрипта:

```
read name  
echo "Hello, $name!"
```

```
danibrogue@jorgserver:~$ . lab  
Daniel  
Hello, Daniel  
danibrogue@jorgserver:~$
```

Рисунок 9 – Пример выполнения скрипта №9

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

а) Код скрипта:

```
echo Enter the 1st number  
read a  
echo Enter second number  
read b  
c=`expr $a + $b` #Присваивание переменной с суммы введенных  
переменных a и a  
echo $c #Вывод сообщения с суммой чисел через значение
```

```
danibrogue@jorgserver:~$ . lab
Enter the 1st number
9
Enter the 2nd number
5
14
danibrogue@jorgserver:~$ _
```

Рисунок 10 – Вычисления с помощью “expr”

6) Код скрипта:

```
echo Enter the 1st
number read a
echo Enter the 2nd number
read b
echo “`bc <<< $a+$b`” #нахождение суммы и её вывод на экран
```

```
danibrogue@jorgserver:~$ . lab
Enter the 1st number
6
Enter the 2nd number
4
10
danibrogue@jorgserver:~$ _
```

Рисунок 11 – Вычисления с помощью “bc”

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

Воспользуемся формулой : $V = \pi * r^2 * h$

```
echo Enter the height of
```

```
cylinder
```

```
read h
```

```
echo Enter the radius of
```

```
cylinder
```

```
read r
```

```
echo “`bc -l <<< 3.14*$r*$r*$h`” #вычисления объёма и вывод на
```

экран.

```
danibrogue@jorgserver:~$ . lab
Enter the height of cylinder
4
Enter the radius of cylinder
4
200.96
```

Рисунок 12 – Вычисление объема цилиндра

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

Код скрипта:

```
echo Name: $0 #вывод имени исполняемого скрипта
echo Amount of arguments: $# #число аргументов
echo Arguments: $* #отображение всех аргументов
```

```
danibrogue@jorgserver:~$ sh lab arg1 arg2 arg3
Name: lab
Amount of arguments: 3
Arguments: arg1 arg2 arg3
danibrogue@jorgserver:~$
```

Рисунок 13 – Пример работы скрипта №12

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

Код скрипта:

```
result= "$(cat $1)" #присваивание переменной result вывод
исполнения команды "cat $1"
echo "${result}" #вывод на экран значения переменной result
sleep 10 - Пауза при выполнении скрипта
clear - очищение экрана
```

Позиционные параметры – это аргументы командной строки или функции в скрипте, доступ к которым осуществляется по номеру. Возможно передавать произвольное число аргументов, но доступными являются только 9 из них. Имеют такой вид: \$1 \$2 \$3 \$4 \$5 \$6 \$7 \$8 \$9.

“\$()” является альтернативным вариантом обратных кавычек ``.

```
danibrogue@jorgserver:~$ cat text
Edinburgh
Phoenix
New York
Los Angeles
danibrogue@jorgserver:~$ . lab text
Edinburgh
Phoenix
New York
Los Angeles
```

Рисунок 14 – Пример работы скрипта №13

После паузы в 10 секунд, экран очищается.

14. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранино.

Код скрипта:

```
for txtFiles in *.txt #циклично – поочередная запись в
переменную txtFiles файлов, имеющих в конце названия текст
“.txt”. От количества файлов зависит количество итераций
цикла.
```

```
do #начало цикла
```

```
cat $txtFiles #Отображение содержимого файла
```

```
done #конец цикла
```

```
danibrogue@jorgserver:~$ ls
file.txt long.txt loop.txt supervisor_script test1.out.log testcron.txt typescript
lab      loop2.txt pipe     test1.err.log    testcron2.txt text
```

Рисунок 15 – имеющиеся txt-файлы

```
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
while true; do true; echo 'Hello'; done
while true; do true; done
Done
Done
danibrogue@jorgserver:~$ _
```

Рисунок 16 – Пример работы скрипта №14

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

```
A=8 #выбираем 8 в качестве допустимого значения
echo Enter the value
read B
if test $B -gt $A #проверка если значение больше
допустимого
then echo Value is greater than the allowed value
else
if test $B -lt $A #проверка если значение меньше
допустимого
then echo Value is lesser than the allowed value
else #если ни одно из условий не выполнено, значение равно
допустимому
```

```
echo Value equals the allowed value
```

```
fi
```

```
fi
```

Опции “gt” и “lt” обозначают больше и меньше соответственно.

```
danibrogue@jorgserver:~$ . lab
Enter the value
7
Value is lesser than the allowed value
danibrogue@jorgserver:~$ . lab
Enter the value
9
Value is greater than the allowed value
danibrogue@jorgserver:~$ . lab
Enter the value
8
Value equals the allowed value
danibrogue@jorgserver:~$
```

Рисунок 17 – Пример работы скрипта №15

16. Программой запрашивается год, определяется, високосный ли он.

Резулььтат выдается на экран

Код скрипта:

```
echo Enter a year
read year
cond=`expr $year % 4` #первое условие – кратность 4
if test $cond -eq 0 #переходим ко второму если истина
then
cond=`expr $year % 100` #второе условие – кратность 100
if test $cond -eq 0 #переходим к третьему если истина
then
cond=`expr $year % 400` #третье условие – кратность 400
if test $cond -eq 0
then echo $year is a leap year
else echo $year is not a leap year
fi #конец третьего условия
else echo $year is a leap year
fi #конец второго условия
else echo $year is not a leap year
fi #конец первого условия
```

Знак “%” находит остаток от деления.

Чтобы определить, является ли год високосным, выполните следующие действия:

- 1) Если год делится на 4 без остатка, перейдите на шаг 2). В противном случае перейдите к выполнению действия 5).
- 2) Если год делится на 100 без остатка, перейдите на шаг 3). В противном случае перейдите к выполнению действия 4).
- 3) Если год делится на 400 без остатка, перейдите на шаг 4). В противном случае перейдите к выполнению действия 5).
- 4) Год високосный (366 дней).
- 5) Год не високосный год (365 дней).

```
danibrogue@jorgserver:~$ . lab
Enter a year
2004
2004 is a leap year
danibrogue@jorgserver:~$ . lab
Enter a year
2003
2003 is not a leap year
danibrogue@jorgserver:~$ . lab
Enter a year
2000
2000 is a leap year
danibrogue@jorgserver:~$ . lab
Enter a year
1900
1900 is not a leap year
danibrogue@jorgserver:~$ _
```

Рисунок 18 – Пример выполнения скрипта №16

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

Код скрипта:

```
echo Enter A
read A
echo Enter B
read B
#введены числа
echo Enter the left bound
read L
echo Enter the right bound
read R
#введены границы
```

```

#итерируем первую переменную
while test $A -lt $R -a $A -gt $L
do
A=`expr $A + 1`
echo Incremented, A = $A
done
#итерируем вторую переменную
while test $B -lt $R -B $B -gt $L
do
B=`expr $B + 1`
echo Incremented, B = $B
done

```

```

danibrogue@jorgserver:~$ . lab
Enter A
2
Enter B
3
Enter the left bound
1
Enter the right bound
5
Incremented, A = 3
Incremented, A = 4
Incremented, A = 5
Incremented, B = 4
Incremented, B = 5
danibrogue@jorgserver:~$ _

```

Рисунок 19 – Пример выполнения скрипта №17

Опция -а работает как логическое «И»

18. В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc
Установим пароль в скрипте “TestPass”

Код скрипта:

```

if test $1 = "TestPass"
then
ls -al /etc |less #команда просмотра файлов, в том числе
скрытных, каталога
/etc и постраничный просмотр
else echo "Incorrect password"
fi

```

```

danibrogue@jorgserver:~$ . lab 1111111
Incorrect password
danibrogue@jorgserver:~$ _

```

Рисунок 20 – Вывод сообщения в случае неправильного пароля

```

total 808
drwxr-xr-x 94 root root          4096 Nov 13 17:02 .
drwxr-xr-x 20 root root          4096 Oct  7 08:46 ..
-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  7 08:52 apport
drwxr-xr-x  7 root root          4096 Oct  7 08:38 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  7 08:52 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          5714 Jul 31 16:29 ca-certificates.conf
-rw-r--r--  1 root root          5713 Jul 31 16:28 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  7 08:49 cloud
drwxr-xr-x  2 root root          4096 Oct  7 09:02 console-setup
drwxr-xr-x  2 root root          4096 Jul 31 16:29 cron.d
drwxr-xr-x  2 root root          4096 Oct  7 08:52 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
-rw-r--r--  1 root root          1042 Feb 13 2020 crontab
drwxr-xr-x  2 root root          4096 Jul 31 16:30 cron.weekly
drwxr-xr-x  2 root root          4096 Oct  7 08:59 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
-rw-r--r--  1 root root          2969 Aug  3 2019 debconf.conf
-rw-r--r--  1 root root          13 Dec  5 2019 debian_version
drwxr-xr-x  3 root root          4096 Nov 13 17:02 default
-rw-r--r--  1 root root          604 Sep 15 2018 deluser.conf
--More--

```

Рисунок 21 – Пример работы скрипта №18

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

Код скрипта:

```

if test -e
$1 then cat
$1 |less
else
echo "File does not
exist"
fi

```

Опция “-e” у “if” проверяет существование файла.

```
danibrogue@jorgserver:~$ ./lab
File doesn't exist
danibrogue@jorgserver:~$ ./lab аоаоао
File doesn't exist
danibrogue@jorgserver:~$ ./lab text
Edinburgh
Phoenix
New York
Los Angeles
(END)
```

Рисунок 22 – Пример работы скрипта №19

20. 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:

```
if test -e $1 #Проверка на существование файла
then
if test -r $1 #Проверка на права чтения файла
then
if test -d $1 #Проверка, является ли файл каталогом
then
ls -l $1 else
cat $1 |less
fi
else
echo "No permission to read the file"
fi
else
mkdir $1
fi
```

```
danibrogue@jorgserver:~$ . lab /home
total 8
drwxr-xr-x 4 danibrogue danibrogue 4096 Nov 27 16:52 danibrogue
drwxr-xr-x 2 user      user        4096 Nov 13 16:41 user
danibrogue@jorgserver:~$ . lab text
Edinburgh
Phoenix
New York
Los Angeles

[3]+  Stopped                  cat $1 | less
danibrogue@jorgserver:~$
```

Рисунок 23 – Пример работы скрипта №20

21. 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:

```
if [ -e $1 ] && [ -f $1 ] #проверка первого файла на
существование и является ли файл обычным
then
if [ -r $1 ] #проверка первого файла на право чтения текущим
пользователем
then
if [ -e $2 ] && [ -f $2 ] #проверка второго файла на
существование и является ли файл обычным
then
if [ -w $2 ] #проверка второго файла на право записи текущим
пользователем
then
cat $1 >>$2 else
echo "User does not have write permission to $2 file"
fi
else
echo "$2 file does not exist or is not a regular file"
fi
else
echo "User does not have read permission to $1 file"
fi
else
echo "$1 file does not exist or is not a regular file"
fi
```

```
danibrogue@jorgserver:~$ sh lab text file.txt
danibrogue@jorgserver:~$ cat text
Edinburgh
Phoenix
New York
Los Angeles
danibrogue@jorgserver:~$ cat file.txt
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10
line 11
Edinburgh
Phoenix
New York
Los Angeles
danibrogue@jorgserver:~$
```

Рисунок 24 – Пример выполнения скрипта №21

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

Код скрипта:

```
if [ -f $1 ] $$ [ -x $1 ] #проверка, является ли файл обычным и
доступен ли файл для исполнения
then
. $1 $2 $3 $4 $5 $6 $7 $8 $9 #запуск исполняемого файла с
позиционными параметрами, которые указали при запуске основного
скрипта.
else
echo "File does not exist or is not executable"
fi
```

```
danibrogue@jorgserver:~$ cat loop2.txt
n=0
while test $n -lt 10
do
echo Done
n=`expr $n + 1`
done
danibrogue@jorgserver:~$ ./lab loop2.txt
Done
danibrogue@jorgserver:~$ _
```

Рисунок 25 – Пример работы скрипта №22

23. 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:

```
if [ -e $1 ] && [ -f $1 ] #проверка на существование файла и
является ли он обычным
then
if [ $(stat -c %s $1) -gt 1 ] #проверка размера файла.
then
sort -k1 $1 >file2.txt #сортировка по 1 столбцу перенаправление
вывода в файл.
cat file2.txt |less
else
echo "$1 is empty"
fi
else
echo "$1 does not exist or is not a text file"
fi
```

```

danibrogue@jorgserver:~$ cat file.txt
total 268
drwxr-xr-x 4 danibrogue danibrogue 4096 Nov 27 19:02 .
drwxr-xr-x 4 root      root      4096 Nov 25 16:36 ..
-rw-r--r-- 1 danibrogue danibrogue 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 danibrogue danibrogue 3771 Feb 25 2020 .bashrc
drwx----- 2 danibrogue danibrogue 4096 Oct  7 09:05 .cache
drwx----- 3 danibrogue danibrogue 4096 Oct 30 14:40 .config
-rw-rw-r-- 1 danibrogue danibrogue 0 Nov 27 19:02 file2.txt
-rw-rw-r-- 1 danibrogue danibrogue 0 Nov 27 18:42 file.txt
-rwxrwxrwx 1 danibrogue danibrogue 187 Nov 27 19:02 lab
-rw-rw-r-- 1 danibrogue danibrogue 111 Oct 31 06:53 long.txt
-rw-rw-r-- 1 danibrogue danibrogue 59 Nov 27 18:11 loop2.txt
-rw-rw-r-- 1 danibrogue danibrogue 26 Oct 29 11:36 loop.txt
prw-rw-r-- 1 danibrogue danibrogue 0 Oct 30 13:39 pipe
-rw-r--r-- 1 danibrogue danibrogue 807 Feb 25 2020 .profile
-rw-r--r-- 1 danibrogue danibrogue 0 Oct  7 09:10 .sudo_as_admin_successful
-rw-r--r-- 1 root      root      48 Nov 22 15:44 supervisor_script
-rw----- 1 root      root      12288 Nov 22 15:42 supervisor_script.swp
-rw-r--r-- 1 root      root      0 Nov 22 15:54 test1.err.log
-rw-r--r-- 1 root      root      30000 Nov 27 19:02 test1.out.log
-rw-r--r-- 1 root      root      0 Nov 22 16:15 testcron2.txt
-rw-r--r-- 1 root      root      10 Nov 22 16:16 testcron.txt
-rw-r--r-- 1 root      root      39 Nov 25 16:49 text
-rw-rw-r-- 1 danibrogue danibrogue 155648 Nov 25 16:49 typescript
-rw----- 1 danibrogue danibrogue 7860 Nov 27 19:02 .viminfo
danibrogue@jorgserver:~$ _

```

Рисунок 26 – Файл для сортировки

```

danibrogue@jorgserver:~$ ./lab file.txt
drwx----- 2 danibrogue danibrogue 4096 Oct  7 09:05 .cache
drwx----- 3 danibrogue danibrogue 4096 Oct 30 14:40 .config
drwxr-xr-x 4 danibrogue danibrogue 4096 Nov 27 19:02 .
drwxr-xr-x 4 root      root      4096 Nov 25 16:36 ..
prw-rw-r-- 1 danibrogue danibrogue 0 Oct 30 13:39 pipe
-rw----- 1 danibrogue danibrogue 7860 Nov 27 19:02 .viminfo
-rw----- 1 root      root      12288 Nov 22 15:42 supervisor_script.swp
-rw-r--r-- 1 danibrogue danibrogue 0 Oct  7 09:10 .sudo_as_admin_successful
-rw-r--r-- 1 danibrogue danibrogue 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 danibrogue danibrogue 3771 Feb 25 2020 .bashrc
-rw-r--r-- 1 danibrogue danibrogue 807 Feb 25 2020 .profile
-rw-r--r-- 1 root      root      0 Nov 22 15:54 test1.err.log
-rw-r--r-- 1 root      root      0 Nov 22 16:15 testcron2.txt
-rw-r--r-- 1 root      root      10 Nov 22 16:16 testcron.txt
-rw-r--r-- 1 root      root      30000 Nov 27 19:02 test1.out.log
-rw-r--r-- 1 root      root      39 Nov 25 16:49 text
-rw-r--r-- 1 root      root      48 Nov 22 15:44 supervisor_script
-rw-rw-r-- 1 danibrogue danibrogue 0 Nov 27 18:42 file.txt
-rw-rw-r-- 1 danibrogue danibrogue 0 Nov 27 19:02 file2.txt
-rw-rw-r-- 1 danibrogue danibrogue 111 Oct 31 06:53 long.txt
-rw-rw-r-- 1 danibrogue danibrogue 155648 Nov 25 16:49 typescript
-rw-rw-r-- 1 danibrogue danibrogue 26 Oct 29 11:36 loop.txt
-rwxrwxrwx 1 danibrogue danibrogue 59 Nov 27 18:11 loop2.txt
-rwxrwxrwx 1 danibrogue danibrogue 187 Nov 27 19:02 lab
total 268
(END)

```

Рисунок 27 – Пример работы скрипта №23

24. 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:

```
tar -cf my.tar *.txt #архивирование всех текстовых файлов
sleep 2 #пауза в 2 секунды
tar -tf my.tar #просмотр содержимого архива tar
gzip my.tar #посредством GZIP сжатия архива tar
```

```
danibrogue@jorgserver:~$ . lab
file2.txt
file.txt
long.txt
loop2.txt
loop.txt
testcron2.txt
testcron.txt
danibrogue@jorgserver:~$ ls
done.tar.gz  lab      loop.txt      test1.err.log  testcron.txt
file2.txt    long.txt  pipe        test1.out.log  text
file.txt     loop2.txt supervisor_script  testcron2.txt  typescript
danibrogue@jorgserver:~$
```

Рисунок 28 – Пример работы скрипта №24

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

Код скрипта:

```
Summ() #инициализация функции с названием Summ
{
    #начало функции
    echo `bc <<< $1+$2` 
} #конец функции
Summ $1 $2 #обращение к функции и передача аргументов
```

Так же, стоит учесть, что в функции используются локальные позиционные параметры, которые не связаны с программой.

```
danibrogue@jorgserver:~$ . lab 5 3
8
danibrogue@jorgserver:~$
```

Рисунок 29 – Пример работы скрипта №25

Вывод

В ходе выполнения лабораторной работы, изучили основные возможности языка программирования Shell с целью автоматизации процесса администрирования системы за счет написания и использования командных файлов.