

Лабораторная работа №10

Дисциплина - операционные системы

Волгин Иван Алексеевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	11
5	Выводы	20

Список иллюстраций

4.1	Создание файла	11
4.2	Код	11
4.3	Исполнение программы	12
4.4	Проверка правильности выполнения	12
4.5	Создание файла	13
4.6	Код	13
4.7	Выполнение программы	14
4.8	Создание файла	15
4.9	Код	16
4.10	Выполнение программы	17
4.11	Создание файла	17
4.12	Код	18
4.13	Выполнение программы	19

Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

2 Задание

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

3 Теоретическое введение

- Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

- Оболочка `bash` поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (*term*), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате типа `radix#number`, где `radix` (основание системы счисления) — любое число не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%). Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7. Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда `let` не ограничена простыми арифметическими выражениями. Табл. 10.1 показывает полный набор `let`-операций. Подобно `C` оболочка `bash` может присваивать переменной любое значение, а произвольное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина». Для облегчения программирования можно записывать условия оболочки `bash` в двойные скобки — (()).

Арифметические операторы оболочки `bash`

Оператор	Синтаксис	Результат
!	<code>!expr</code>	Если <code>expr</code> равно 0, то возвращает 1; иначе 0
!=	<code>expr1 != expr2</code>	Если <code>expr1</code> не равно <code>expr2</code> , то возвращает 1; иначе 0
%	<code>expr1 % expr2</code>	Возвращает остаток от деления <code>expr1</code> на <code>expr2</code>
%=	<code>var=%expr</code>	Присваивает остаток от деления <code>var</code> на <code>expr</code> переменной <code>var</code>
&	<code>expr1 & expr2</code>	Возвращает побитовое AND выражений <code>expr1</code> и <code>expr2</code>

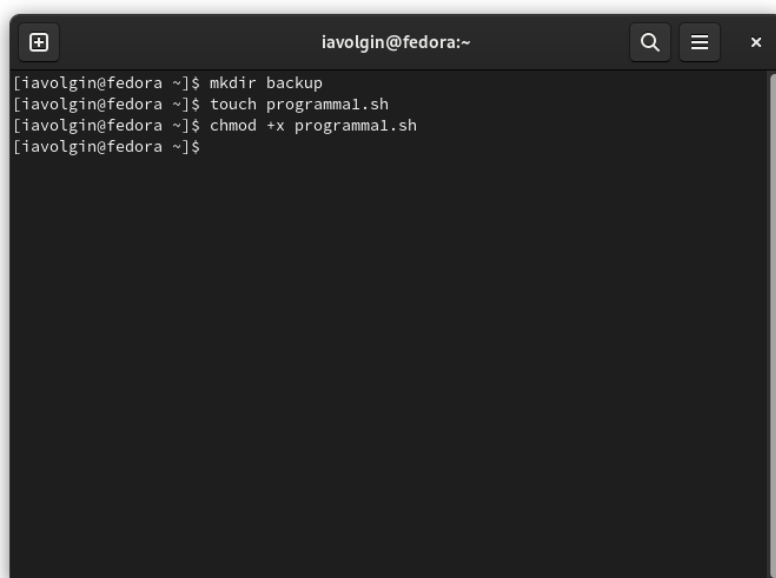
$\text{exp1} \& \text{exp2}$ Если и exp1 и exp2 не равны нулю, то возвращает 1; иначе 0
 $\text{var} \&= \text{exp}$ Присваивает переменной var побитовое AND var и exp
 $\text{exp1} * \text{exp2}$ Умножает exp1 на exp2
 $\text{var} = \text{exp}$ Присваивает переменной var значение exp
 $\text{var} *= \text{exp}$ Умножает exp на значение переменной var и присваивает результат переменной
 $\text{var} + \text{exp1} + \text{exp2}$ Складывает exp1 и exp2
 $\text{var} += \text{exp}$ Складывает exp со значением переменной var и результат присваивает переменной
 $\text{var} - \text{exp}$ Операция отрицания exp (унарный минус)
 $\text{exp1} - \text{exp2}$ Вычитает exp2 из exp1
 $\text{var} -= \text{exp}$ Вычитает exp из значения переменной var и присваивает результат переменной
 $\text{var} / \text{exp} / \text{exp2}$ Делит exp1 на exp2
 $\text{var} /= \text{exp}$ Делит значение переменной var на exp и присваивает результат переменной
 $\text{var} < \text{exp1} < \text{exp2}$ Если exp1 меньше, чем exp2 , то возвращает 1, иначе возвращает 0
 $\text{exp1} \ll \text{exp2}$ Сдвигает exp1 влево на exp2 бит
 $\text{var} \ll \text{exp}$ Побитовый сдвиг влево значения переменной var на exp бит
 $\text{exp1} \leq \text{exp2}$ Если exp1 меньше или равно exp2 , то возвращает 1; иначе возвращает 0
 $\text{var} = \text{exp}$ Присваивает значение exp переменной var
 $\text{exp1} == \text{exp2}$ Если exp1 равно exp2 , то возвращает 1; иначе возвращает 0
 $\text{exp1} > \text{exp2}$ 1, если exp1 больше, чем exp2 ; иначе 0
 $\text{exp1} \geq \text{exp2}$ 1, если exp1 больше или равно exp2 ; иначе 0
 $\text{exp1} \gg \text{exp2}$ Сдвигает exp1 вправо на exp2 бит
 $\text{var} \gg \text{exp}$ Побитовый сдвиг вправо значения переменной var на exp бит
 $\text{exp1} \wedge \text{exp2}$ Исключающее OR выражений exp1 и exp2
 $\text{var} \wedge= \text{exp}$ Присваивает переменной var побитовое XOR var и exp
 $\text{exp1} | \text{exp2}$ Побитовое OR выражений exp1 и exp2
 $\text{var} |= \text{exp}$ Присваивает переменной var результат операции XOR var и exp
 $\text{exp1} || \text{exp2}$ 1, если или exp1 или exp2 являются ненулевыми значениями; иначе 0
 $\sim \text{exp}$ Побитовое дополнение до exp

- При перечислении имён файлов текущего каталога можно использовать следующие символы:
 - $*$ — соответствует произвольной, в том числе и пустой строке;
 - $?$ — соответствует любому одинарному символу;
 - $[c1-c2]$ — соответствует любому символу, лексикографически находящемуся между символами $c1$ и $c2$. Например, `echo *` — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `ls .c` — выведет все файлы с последними двумя символами, совпадающими с `.c`.

– *echo prog.?* — выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются *prog.* – *[a-z]* — соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита. Такие символы, как *' < > * ? | " &*, являются метасимволами и имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме *\$, ', , "*. Например, – *echo ** выведет на экран символ, – *echo ab'|'cd* выведет на экран строку *ab|*cd*.

4 Выполнение лабораторной работы

Я приступил к первому заданию и для начала создал файл и изменил для него права доступа (рис. 4.1). После этого написал код, который выполняет поставленную задачу (рис. 4.2). Далее запустил программу (рис. 4.3) и проверил правильность ее выполнения (рис. 4.4).



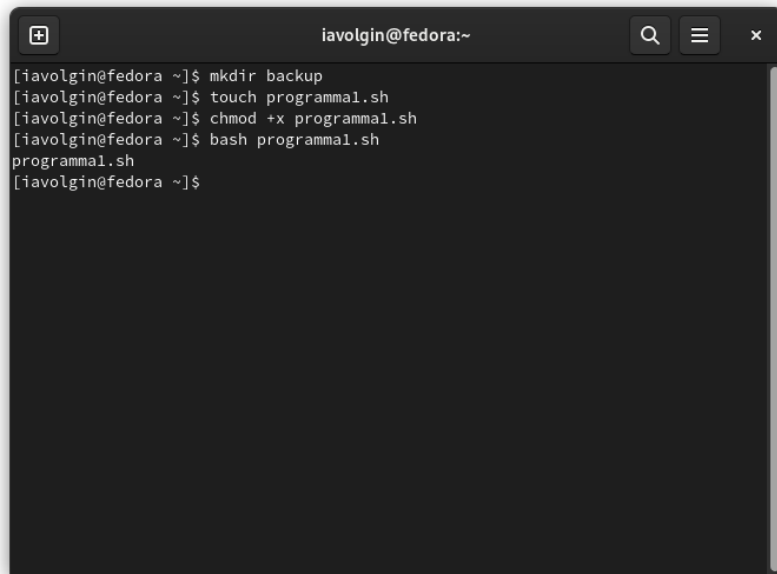
```
iavolgin@fedora:~  
[iavolgin@fedora ~]$ mkdir backup  
[iavolgin@fedora ~]$ touch programma1.sh  
[iavolgin@fedora ~]$ chmod +x programma1.sh  
[iavolgin@fedora ~]$
```

Рис. 4.1: Создание файла



```
Открыть ▾ + programma1.sh  
~/  
tar -cvf ~/backup/backup.tar programma1.sh
```

Рис. 4.2: Код



```
iavolgin@fedora:~  
[iavolgin@fedora ~]$ mkdir backup  
[iavolgin@fedora ~]$ touch program1.sh  
[iavolgin@fedora ~]$ chmod +x program1.sh  
[iavolgin@fedora ~]$ bash program1.sh  
program1.sh  
[iavolgin@fedora ~]$
```

Рис. 4.3: Исполнение программы

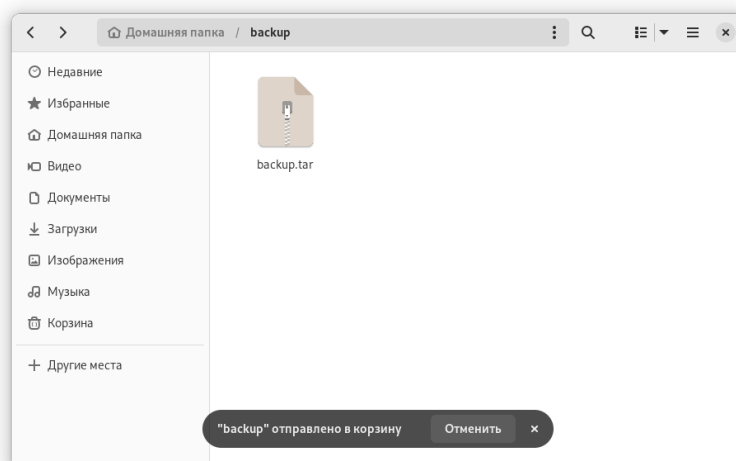
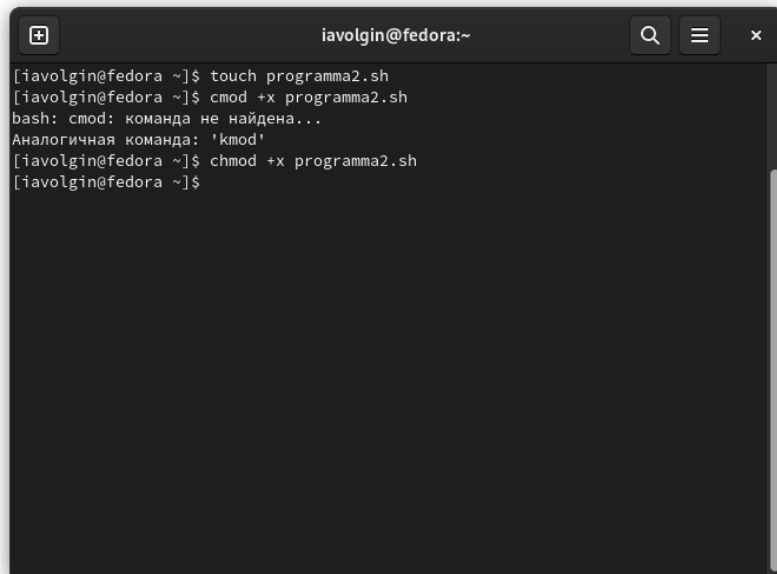


Рис. 4.4: Проверка правильности выполнения

После этого я приступил к выполнению второго задания. Все идентично: создал файл и поменял права доступа (рис. 4.5), написал код (рис. 4.6), используя цикл `for`, и выполнил его (рис. 4.7).



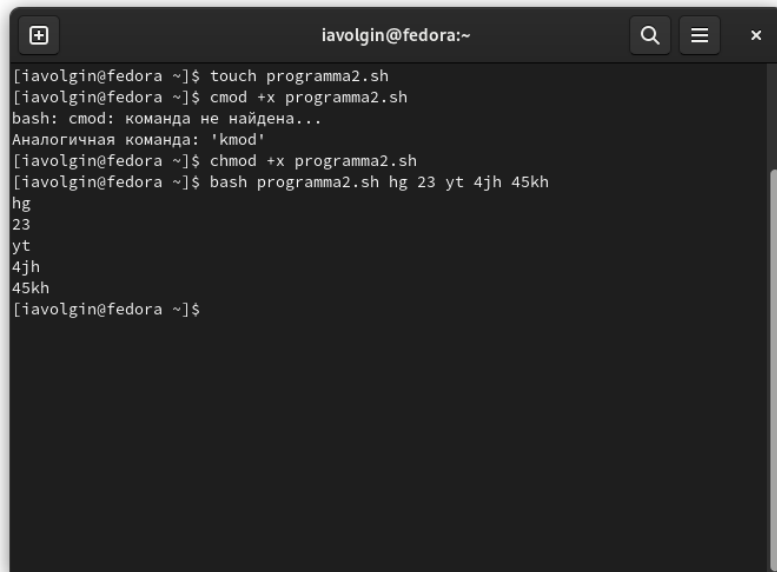
```
iavolgin@fedora:~  
[iavolgin@fedora ~]$ touch programma2.sh  
[iavolgin@fedora ~]$ cmod +x programma2.sh  
bash: cmod: команда не найдена...  
Аналогичная команда: 'kmod'  
[iavolgin@fedora ~]$ chmod +x programma2.sh  
[iavolgin@fedora ~]$
```

Рис. 4.5: Создание файла



```
Открыть ▾ + programma2.sh  
~/  
for A in $*  
do echo $A  
done|
```

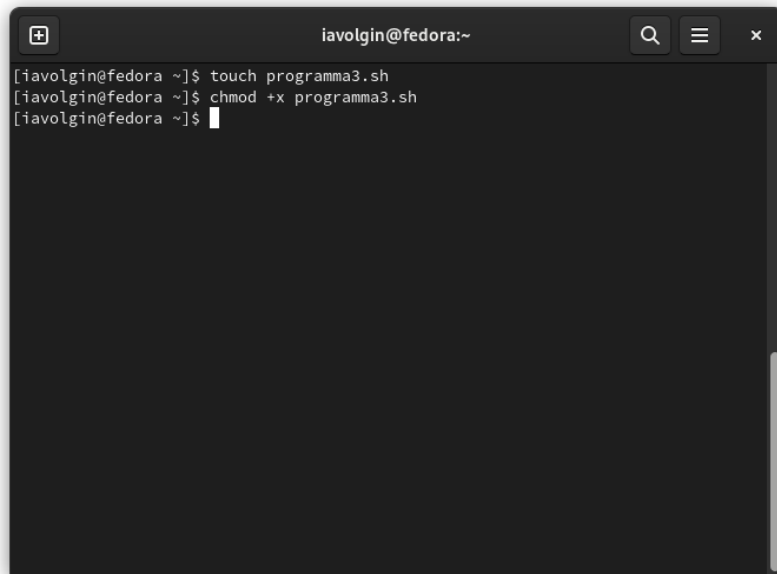
Рис. 4.6: Код

A terminal window titled 'iavolgin@fedora:~' with search, menu, and close buttons. It shows the following commands and output:

```
[iavolgin@fedora ~]$ touch programma2.sh
[iavolgin@fedora ~]$ chmod +x programma2.sh
bash: chmod: команда не найдена...
Аналогичная команда: 'kmod'
[iavolgin@fedora ~]$ chmod +x programma2.sh
[iavolgin@fedora ~]$ bash programma2.sh hg 23 yt 4jh 45kh
hg
23
yt
4jh
45kh
[iavolgin@fedora ~]$
```


Рис. 4.7: Выполнение программы

Далее задание номер 3. Все по тому же сценарию: создаю файл (рис. 4.8), пишу код (рис. 4.9). В коде так же используется цикл `for`, но уже с оператором `if`, с помощью которого мы проверяем, чем является наш элемент (директорией или файлом) и если файлом, то тоже через оператор `if` проверяем какие у него есть права доступа. Далее выполняю его (рис. 4.10)

A terminal window with a dark background and light text. The title bar at the top shows a plus icon, the text 'iavolgin@fedora:~', a search icon, a menu icon, and a close icon. The terminal content shows three lines of commands and their prompts: '[iavolgin@fedora ~]\$ touch programma3.sh', '[iavolgin@fedora ~]\$ chmod +x programma3.sh', and '[iavolgin@fedora ~]\$' followed by a cursor. A vertical scrollbar is visible on the right side of the terminal window.

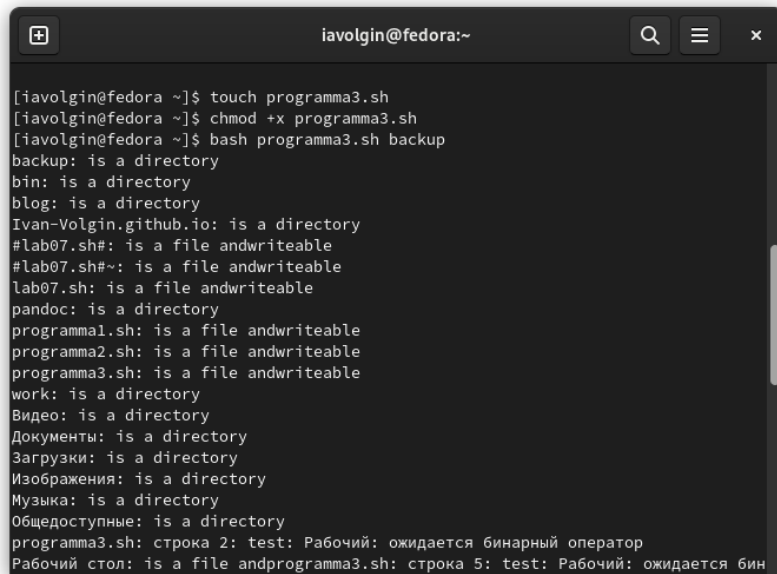
```
iavolgin@fedora:~  
[iavolgin@fedora ~]$ touch programma3.sh  
[iavolgin@fedora ~]$ chmod +x programma3.sh  
[iavolgin@fedora ~]$
```

Рис. 4.8: Создание файла



```
for A in *
do if test -d $A
then echo $A: is a directory
else echo -n $A: is a file and
if test -w $A
then echo writeable
elif test -r $A
then echo readable
else echo neither readable nor writeable
fi
fi
done
```

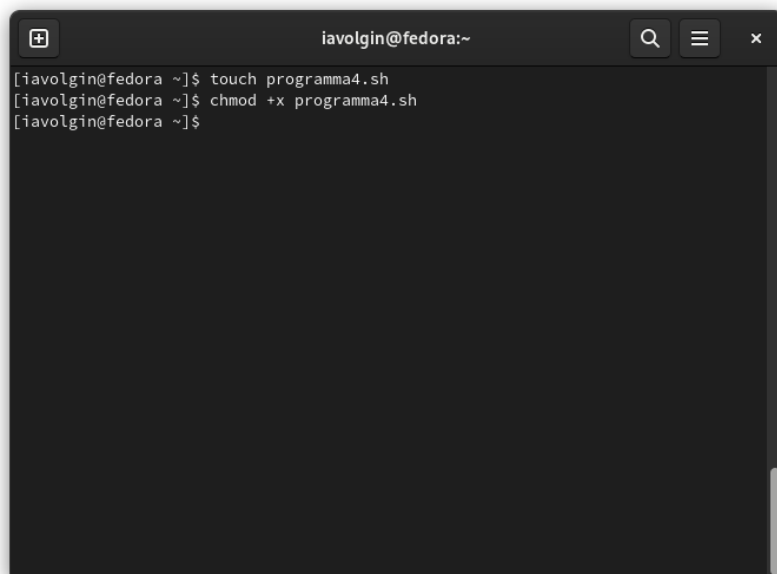
Рис. 4.9: Код



```
iavolgin@fedora:~  
[iavolgin@fedora ~]$ touch programma3.sh  
[iavolgin@fedora ~]$ chmod +x programma3.sh  
[iavolgin@fedora ~]$ bash programma3.sh backup  
backup: is a directory  
bin: is a directory  
blog: is a directory  
Ivan-Volgin.github.io: is a directory  
#lab07.sh#: is a file andwriteable  
#lab07.sh~: is a file andwriteable  
lab07.sh: is a file andwriteable  
pandoc: is a directory  
programma1.sh: is a file andwriteable  
programma2.sh: is a file andwriteable  
programma3.sh: is a file andwriteable  
work: is a directory  
Видео: is a directory  
Документы: is a directory  
Загрузки: is a directory  
Изображения: is a directory  
Музыка: is a directory  
Общедоступные: is a directory  
programma3.sh: строка 2: test: Рабочий: ожидается бинарный оператор  
Рабочий стол: is a file andprogramma3.sh: строка 5: test: Рабочий: ожидается бин
```

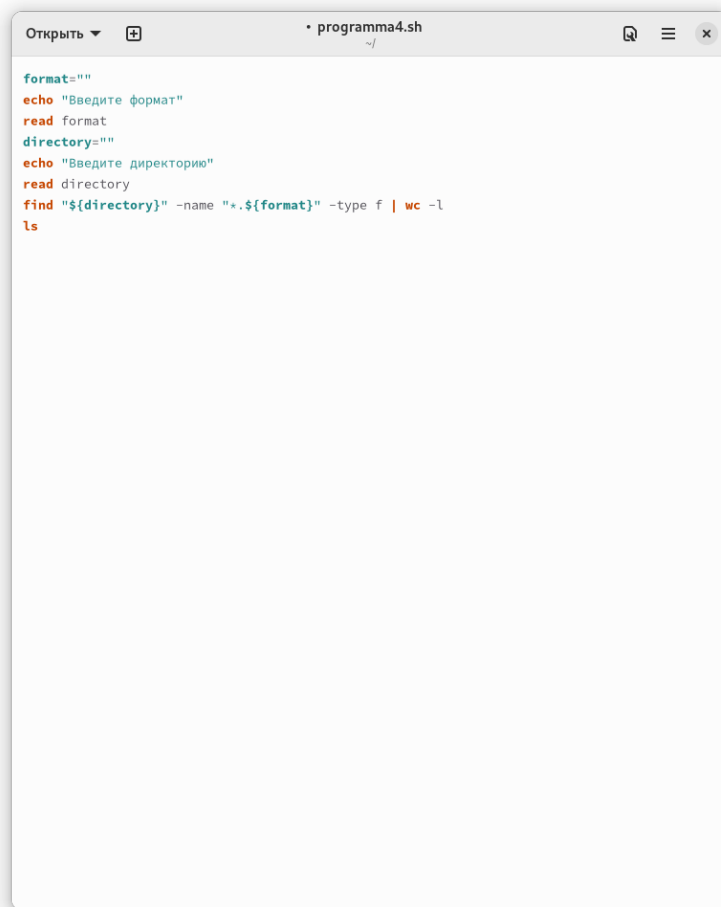
Рис. 4.10: Выполнение программы

Четвертое задание выполнялось так же. Создал файл и добавил права доступа (рис. 4.11), написал код (рис. 4.12), и выполнил его (рис. 4.13).



```
iavolgin@fedora:~  
[iavolgin@fedora ~]$ touch programma4.sh  
[iavolgin@fedora ~]$ chmod +x programma4.sh  
[iavolgin@fedora ~]$
```

Рис. 4.11: Создание файла



The image shows a terminal window with a title bar containing the text "programma4.sh" and standard window controls. The terminal displays a shell script with the following lines of code:

```
format=""
echo "Введите формат"
read format
directory=""
echo "Введите директорию"
read directory
find "${directory}" -name ".*${format}" -type f | wc -l
ls
```

Рис. 4.12: Код

```
iavolgin@fedora:~  
Введите формат  
.sh  
Введите директорию  
/home/iavolgin  
0  
backup      lab07.sh      work          Общедоступные  
bin          pandoc        Видео         'Рабочий стол'  
blog        programma1.sh Документы     Шаблоны  
Ivan-Volgin.github.io programma2.sh Загрузки  
'#lab07.sh#' programma3.sh Изображения  
'#lab07.sh#~' programma4.sh Музыка  
[iavolgin@fedora ~]$ bash programma4.sh  
Введите формат  
sh  
Введите директорию  
/home/iavolgin  
8  
backup      lab07.sh      work          Общедоступные  
bin          pandoc        Видео         'Рабочий стол'  
blog        programma1.sh Документы     Шаблоны  
Ivan-Volgin.github.io programma2.sh Загрузки  
'#lab07.sh#' programma3.sh Изображения  
'#lab07.sh#~' programma4.sh Музыка  
[iavolgin@fedora ~]$
```

Рис. 4.13: Выполнение программы

5 Выводы

В ходе выполнения лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.