

Лабораторная работа №12

Дисциплина: Операционные системы

Коновалова Татьяна Борисовна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Библиография	18
5	Выводы	19

List of Tables

List of Figures

3.1	Создание файла	7
3.2	Скрипт №1	8
3.3	Предоставление прав доступа	8
3.4	Проверка работы программы	8
3.5	Создание файлов	9
3.6	Работа в файле chslo.c	10
3.7	Работа в файле chslo.sh	11
3.8	Проверка скрипта №2	11
3.9	Создание файлов	12
3.10	Скрипт №3	12
3.11	Проверка работы скрипта №3	13
3.12	Создание файлов	13
3.13	Скрипт №4	14
3.14	Проверка скрипта №4	15

1 Цель работы

Цель данной лабораторной работы — Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Сделать отчёт по лабораторной работе №12 в формате Markdown.
2. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

3 Выполнение лабораторной работы

1). Используя команды `getopts` `grep`, написала командный файл, который анализирует командную строку с ключами: 1. `-i`inputfile—прочитать данные из указанного файла; 2. `-o`outputfile—вывести данные в указанный файл; 3. `-r` шаблон —указать шаблон для поиска; 4. `-C`—различать большие и малые буквы; 5. `-n`—выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом `-r`. Для данной задачи я создала файл `prog1.sh` (Рисунки 3.1) и написала соответствующие скрипты. (алгоритм действий представлен на рис. 3.2).

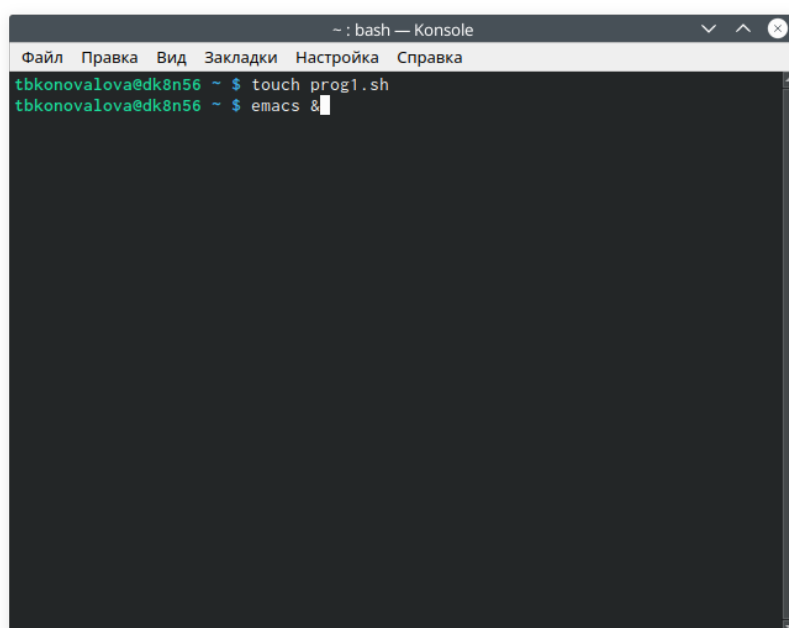


Figure 3.1: Создание файла

```

#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0; #Инициализация переменных-флагов, присвоение им 0
while getopts :la:pC:n optletter; do #Анализируем каждую строку на наличие опции
do case $optletter in
l) iflag=1; val=$OPTARG;;
a) oflag=1; oval=$OPTARG;;
p) oflag=1; oval=$OPTARG;;
C) cflag=1;;
n) nflag=1;;
*) echo illegal option $optletter
&&
done
if (($iflag==0))
then echo "Шаблон не найден" #Проверка, указан ли шаблон для поиска
else
if (($iflag==0))
then echo "Файл не найден"
else
if (($cflag==0))
then if (($iflag==0))
then grep $oval $ival
else grep -n $oval $ival
fi
else if (($nflag==0))
then grep -i $oval $ival
else grep -i -n $oval $ival
fi
else if (($cflag==0))
then if (($iflag==0))
then grep $oval $ival > $oval
else grep -n $oval $ival > $oval
fi
else if (($nflag==0))
then grep -i $oval $ival > $oval
else grep -i -n $oval $ival > $oval
fi
fi
fi
fi
fi

```

Figure 3.2: Скрипт №1

Проверила работу написанного скрипта, используя различные опции (например, команда «./prog.sh -la1.txt -oa2.txt -pcapital -C -n»), предварительно добавив право на исполнение файла (команда «chmod+x prog1.sh») и создав 2 файла, которые необходимы для выполнения программы: a1.txt и a2.txt (алгоритм действий представлен на рис. 3.3 , 3.4). Скрипт работает корректно.

```

tbkonovalova@dk8n56 ~ $ touch a1.txt a2.txt
tbkonovalova@dk8n56 ~ $ chmod +x prog1.sh

```

Figure 3.3: Предоставление прав доступа

```

tbkonovalova@dk8n56 ~ $ cat a1.txt
water abc abcs
asd
prog1
water water
tbkonovalova@dk8n56 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -n
tbkonovalova@dk8n56 ~ $ cat a2.txt
1:water abc abcs
4:water water
tbkonovalova@dk8n56 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -n
tbkonovalova@dk8n56 ~ $ cat a2.txt
1:water abc abcs
4:water water
tbkonovalova@dk8n56 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -C -n
tbkonovalova@dk8n56 ~ $ cat a2.txt
1:water abc abcs
4:water water
tbkonovalova@dk8n56 ~ $ ./prog1.sh -i a1.txt -C -n
Шаблон не найден
tbkonovalova@dk8n56 ~ $ ./prog1.sh -o a2.txt -p water -C -n
Файл не найден
tbkonovalova@dk8n56 ~ $

```

Figure 3.4: Проверка работы программы

2). Написала на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено. Для данной задачи я создала 2 файла: `chslo.c` и `chislo.sh` (Рисунок 3.5) и написала соответствующие скрипты. (команды «`touch prog2.sh`» и «`emacs &`») (Скриншоты 3.6 , 3.7).

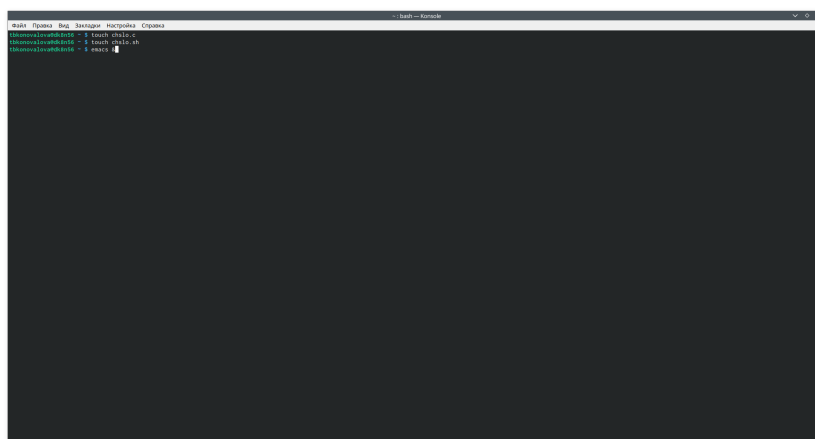
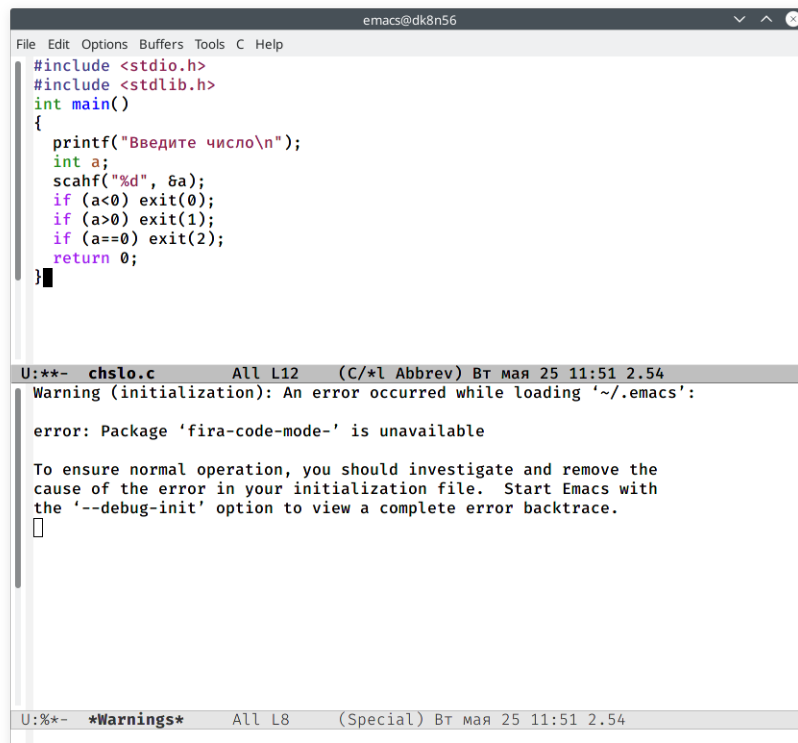


Figure 3.5: Создание файлов



The image shows a screenshot of the Emacs editor window. The title bar at the top reads "emacs@dk8n56". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "C", and "Help". The main editing area contains the following C code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Введите число\n");
    int a;
    scanf("%d", &a);
    if (a<0) exit(0);
    if (a>0) exit(1);
    if (a==0) exit(2);
    return 0;
}
```

Below the code, a status bar shows "U:*** chslo.c All L12 (C/*l Abbrev) Вт мая 25 11:51 2.54". The bottom panel displays a warning message:

```
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
```

At the very bottom, another status bar shows "U:%*- *Warnings* All L8 (Special) Вт мая 25 11:51 2.54".

Figure 3.6: Работа в файле chslo.c

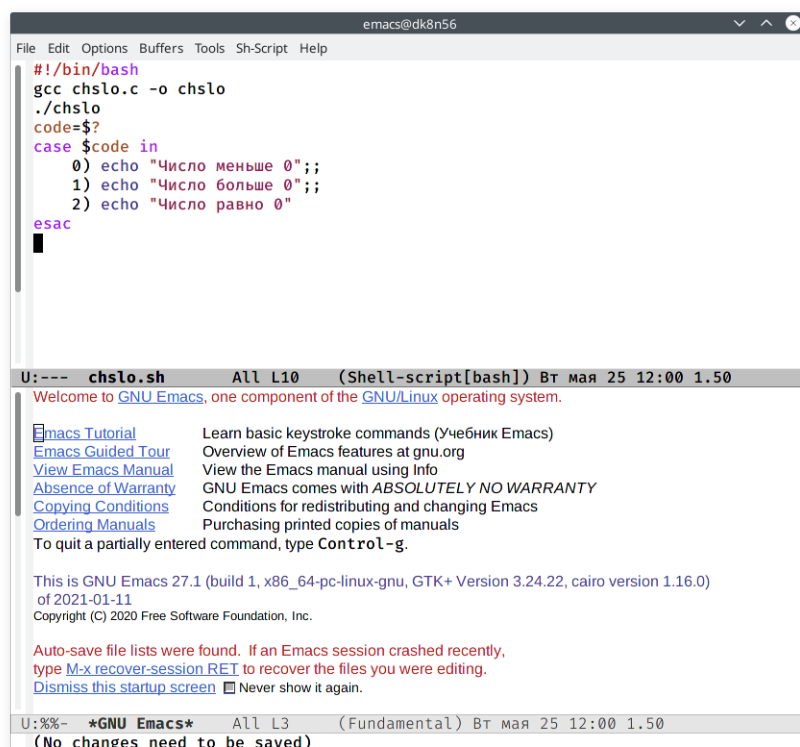


Figure 3.7: Работа в файле chslo.sh

Проверила работу написанных скриптов (команда «./chislo.sh»), предварительно добавив право на исполнение файла (команда «chmod+x chislo.sh») (Рисунок 3.8). Скрипты работают корректно.

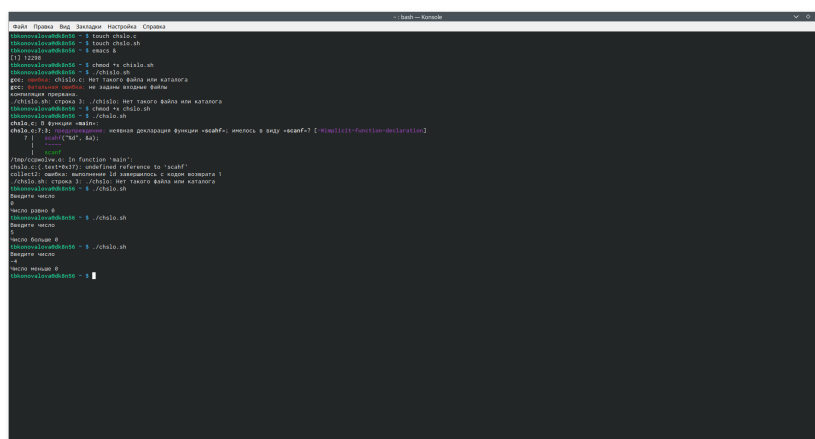


Figure 3.8: Проверка скрипта №2

3). Написала командный файл, создающий указанное число файлов, пронуме-

рованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp,4.tmpи т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). Для данной задачи я создала файл: files.sh (Рисунок 3.9) и написала соответствующий скрипт (алгоритм действий представлен на рис. 3.10).

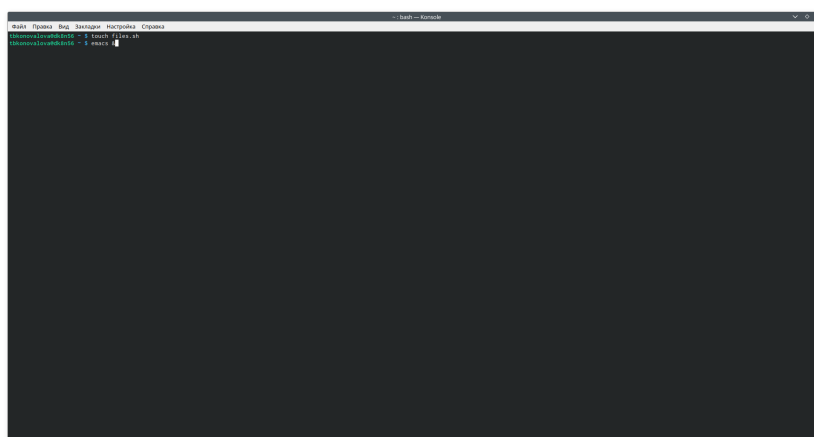


Figure 3.9: Создание файлов

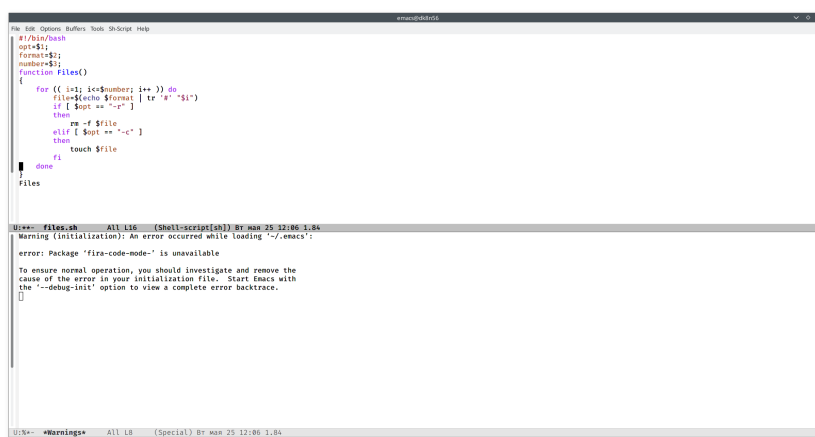


Figure 3.10: Скрипт №3

Далее я проверила работу написанного скрипта (команда «./files.sh»), предварительно добавив право на исполнение файла (команда «chmod+x files.sh»). Сначала я создала три файла (команда «./files.sh-cabc#.txt3»), удовлетворяющие

условию задачи, а потом удалила их (команда «./files.sh-rabc#.txt3») (Скриншот 3.11).

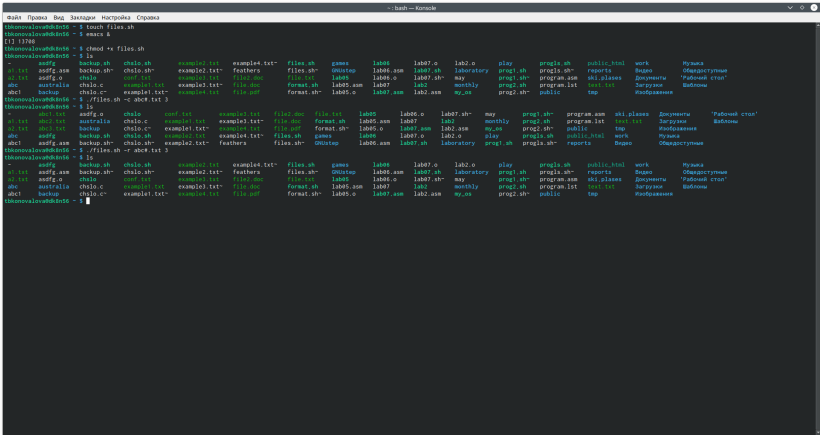


Figure 3.11: Проверка работы скрипта №3

4). Написала командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). Для данной задачи я создала файл: prog4.sh (Скриншот 3.12) и написала соответствующий скрипт (См. рис. 3.13).

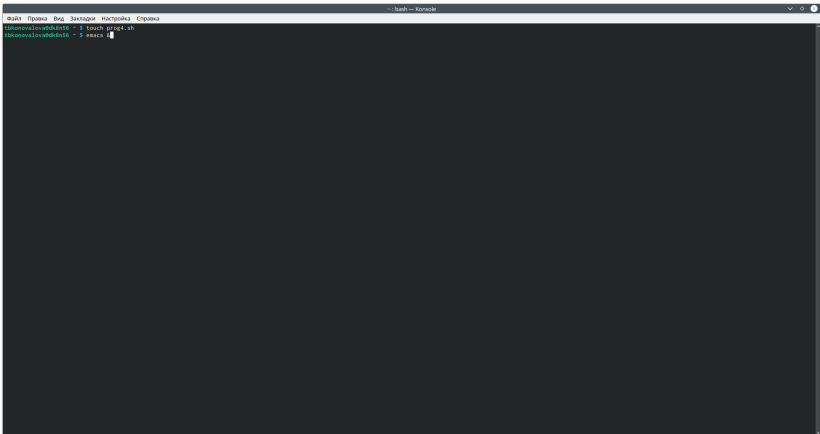
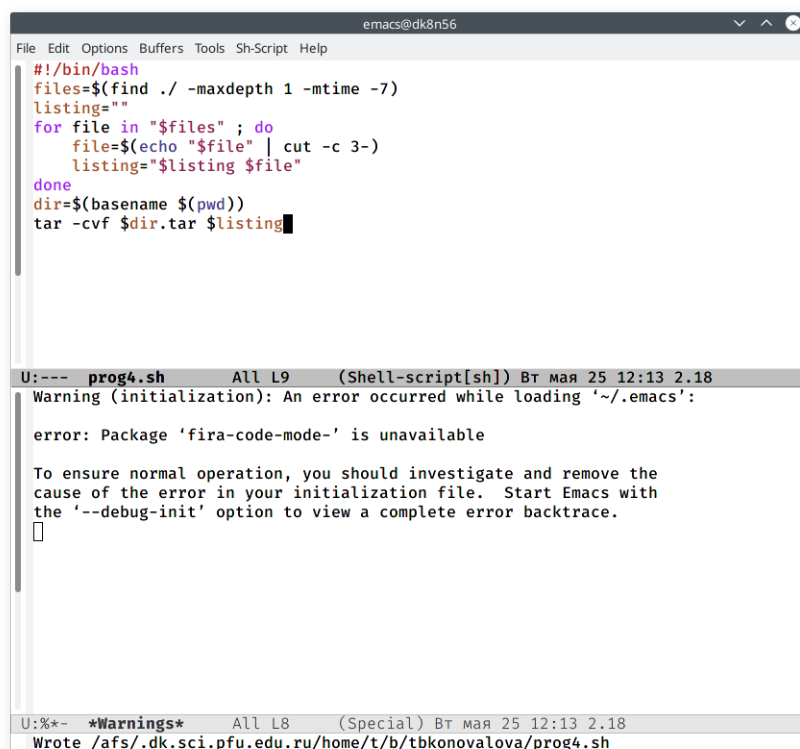


Figure 3.12: Создание файлов



The screenshot shows an Emacs editor window titled 'emacs@dk8n56'. The main buffer contains a shell script named 'prog4.sh' with the following content:

```
#!/bin/bash
files=$(find ./ -maxdepth 1 -mtime -7)
listing=""
for file in "$files" ; do
    file=$(echo "$file" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

Below the script, the output of running the script is shown in a buffer titled 'U:--- prog4.sh All L9 (Shell-script[sh]) Вт мая 25 12:13 2.18'. The output includes a warning and an error message:

```
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
```

At the bottom, a buffer titled 'U:%*- *Warnings* All L8 (Special) Вт мая 25 12:13 2.18' shows the command used to run the script:

```
Wrote /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/prog4.sh
```

Figure 3.13: Скрипт №4

Далее я проверила работу написанного скрипта (команды «./prog4.sh» и «tar-tf Catalog1.tar»), предварительно добавив право на исполнение файла (команда «chmod +x prog4.sh») и создав отдельный Catalog1 с несколькими файлами. Как видно из Рисунков 3.14, файлы, измененные более недели назад, заархивированы не были. Скрипт работает корректно.

```

tbkonovalova@dk8n56 ~/Catalog1 $ ~/prog4.sh
a1.txt
a2.txt
chslo
chslo.c
chslo.sh
tbkonovalova@dk8n56 ~/Catalog1 $ ~/prog4.sh
a1.txt
a2.txt
chslo
chslo.c
chslo.sh
tar: Catalog1.tar: файл является архивом; не сброшен
tbkonovalova@dk8n56 ~/Catalog1 $ ./prog4.sh
bash: ./prog4.sh: Нет такого файла или каталога
tbkonovalova@dk8n56 ~/Catalog1 $ ./prog4.sh
a1.txt
a2.txt
chslo
chslo.c
chslo.sh
tar: Catalog1.tar: файл является архивом; не сброшен
prog4.sh
tbkonovalova@dk8n56 ~/Catalog1 $ tar -tf Catalog1.tar
a1.txt
a2.txt
chslo
chslo.c
chslo.sh
prog4.sh
tbkonovalova@dk8n56 ~/Catalog1 $

```

Figure 3.14: Проверка скрипта №4

Ответы на контрольные вопросы:

1). Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg...]` Флаги – это опции командной строки, обычно помеченные знаком минус; Например, для команды `ls` флагом может являться `-F`. Строка опций `option-string` – это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введённые данные с помощью оператора `case`. Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение

ние, то OPTARG устанавливается в значение этого аргумента. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

2). При перечислении имён файлов текущего каталога можно использовать следующие символы: 1. *–соответствует произвольной, в том числе и пустой строке;* 2. *?–соответствует любому одинарному символу;* 3. *[c1-c2] – соответствует любому символу, лексикографически находящемуся между символами c1 и c2. Например, 1.1 echo – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды ls; 1.2. ls.c–выведет все файлы с последними двумя символами, совпадающими с c. 1.3. echoprog.?-выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются prog.. 1.4.[a-z]–соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.*

3). Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования `bash` предоставляет возможность использовать такие управляющие конструкции, как `for`, `case`, `if` и `while`. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.

4). Два несложных способа позволяют вам прерывать циклы в оболочке `bash`.

Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов. Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестаёт быть правильным. Команда `continue` используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях.

5). Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования `bash`: это команда `true`, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда `false`, которая всегда возвращает код завершения, неравный нулю (т.е. ложь). Примеры бесконечных циклов: `while true do echo hello andy done` `until false do echo hello mike done`.

6). Строка `if test -f mans/i.s, mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).

7). Выполнение оператора цикла `while` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, возвратит ненулевой код завершения (ложь). При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла `while` и оператор цикла `until` идентичны.

4 Библиография

1. Программное обеспечение GNU/Linux. Лекция 3. FHS и процессы (Г. Курячий, МГУ);
2. Программное обеспечение GNU/Linux. Лекция 4. Права доступа (Е. Алёхова, МГУ);
3. Программное обеспечение GNU/Linux. Лекция 6. ПО не из хранилища дистрибутива (Г. Курячий, МГУ)
4. Электронный ресурс: <https://www.skleroznik.in.ua/2013/07/31/cikly-i-vetvleniya/>
5. Электронный ресурс: https://www.opennet.ru/docs/RUS/bash_scripting_guide/c4875.html

5 Выводы

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