

Лабораторная работа №12

Дисциплина: Операционные системы

Егорова Александра

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Выводы	11
4	Контрольные вопросы	12
5	Библиография	15

List of Figures

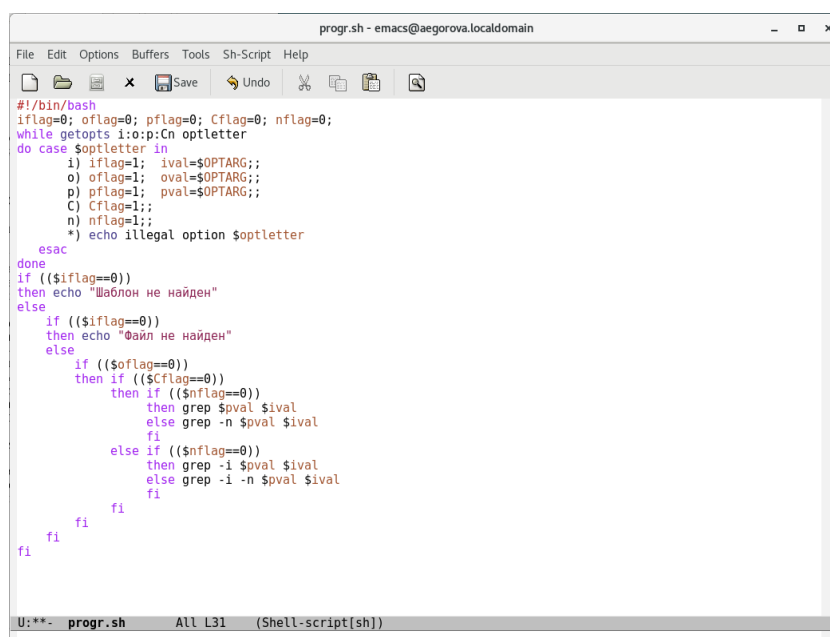
2.1	Первая программа	5
2.2	Проверка первой программы	6
2.3	Вторая программа на с	7
2.4	Вторая программа sh	7
2.5	Проверка второй программы	8
2.6	Третья программа	8
2.7	Проверка третьей программы	9
2.8	Проверка третьей программы	9
2.9	Четвертая программа	10
2.10	Проверка четвертой программы	10

1 Цель работы

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

2 Выполнение лабораторной работы

- 1) Используя команды `getopts` `grep`, написала командный файл, который анализирует командную строку с ключами: 1) `-iinputfile` — прочитать данные из указанного файла; 2) `-ooutputfile` — вывести данные в указанный файл; 3) `-rшаблон` — 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`. Я создала файл `progr1.sh` и написала соответствующие скрипты. (рис. -fig. 2.1)

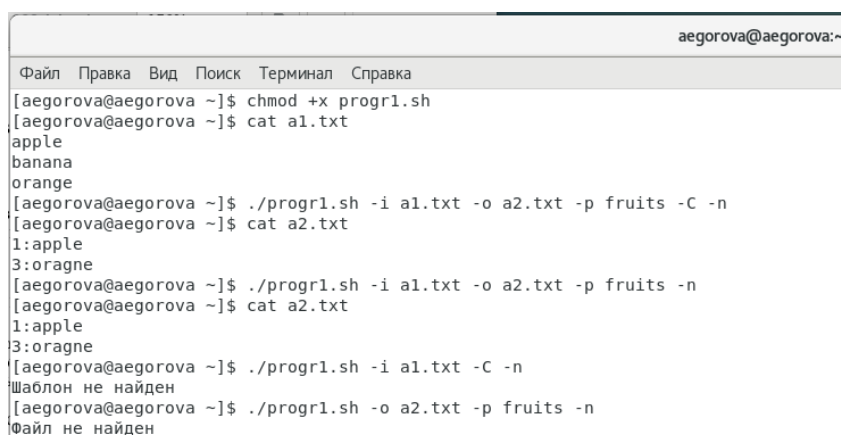


```
progr.sh - emacs@aegorova.localdomain
File Edit Options Buffers Tools Sh-Script Help
# /bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0;
while getopts i:o:p:Cn optletter
do case $optletter in
  i) iflag=1; ival=$OPTARG;;
  o) oflag=1;  oval=$OPTARG;;
  p) pflag=1;  pval=$OPTARG;;
  C) Cflag=1;;
  n) nflag=1;;
  *) echo illegal option $optletter
  esac
done
if ((iflag==0))
then echo "Шаблон не найден"
else
  if ((oflag==0))
  then echo "Файл не найден"
  else
    if ((Cflag==0))
    then if ((nflag==0))
        then grep $pval $ival
        else grep -n $pval $ival
        fi
    else if ((nflag==0))
        then grep -i $pval $ival
        else grep -i -n $pval $ival
        fi
    fi
  fi
fi
fi
U:***- progr.sh All L31 (Shell-script[sh])
```

Figure 2.1: Первая программа

Далее я проверила работу написанного скрипта, используя различные опции («./progr1.sh -I a1.txt -o a2.txt -r capital -C -n»), предварительно добавив право на

исполнение файла («chmod +x progr1.sh») и создав 2 файла, которые необходимы для выполнения программы: a1.txt и a2.txt. Скрипт работает корректно. (рис. -fig. 2.2)



```
aegorova@aegorova:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
[aegorova@aegorova ~]$ chmod +x progr1.sh  
[aegorova@aegorova ~]$ cat a1.txt  
apple  
banana  
orange  
[aegorova@aegorova ~]$ ./progr1.sh -i a1.txt -o a2.txt -p fruits -C -n  
[aegorova@aegorova ~]$ cat a2.txt  
1:apple  
3:orange  
[aegorova@aegorova ~]$ ./progr1.sh -i a1.txt -o a2.txt -p fruits -n  
[aegorova@aegorova ~]$ cat a2.txt  
1:apple  
3:orange  
[aegorova@aegorova ~]$ ./progr1.sh -i a1.txt -C -n  
Шаблон не найден  
[aegorova@aegorova ~]$ ./progr1.sh -o a2.txt -p fruits -n  
Файл не найден
```

Figure 2.2: Проверка первой 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: progr2.c и progr2.sh и написала соответствующие скрипты. (рис. -fig. 2.3) (рис. -fig. 2.4)

The image shows an Emacs editor window with the title "progr2.c - emacs@aegorova.localdomain". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "C", and "Help". The toolbar contains icons for file operations and editing. The code in the buffer is a C program that prompts the user to enter a number and then checks if it is less than, greater than, or equal to zero. The code is as follows:

```
#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;
}
```

Figure 2.3: Вторая программа на c

The image shows an Emacs editor window with the title "progr2.sh - emacs@aegorova.localdomain". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". The toolbar contains icons for file operations and editing. The code in the buffer is a shell script that compiles the C program "progr2.c" into "progr2" and then uses a case statement to print messages based on the exit code of "progr2". The script is as follows:

```
#!/bin/bash
gcc progr2.c -o progr2
./progr2
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0";;
esac
```

Figure 2.4: Вторая программа sh

Далее я проверила работу написанных скриптов («./progr2.sh»), предварительно добавив право на исполнение файла («chmod +x progr2.sh»). Скрипты работают корректно. (рис. -fig. 2.5)

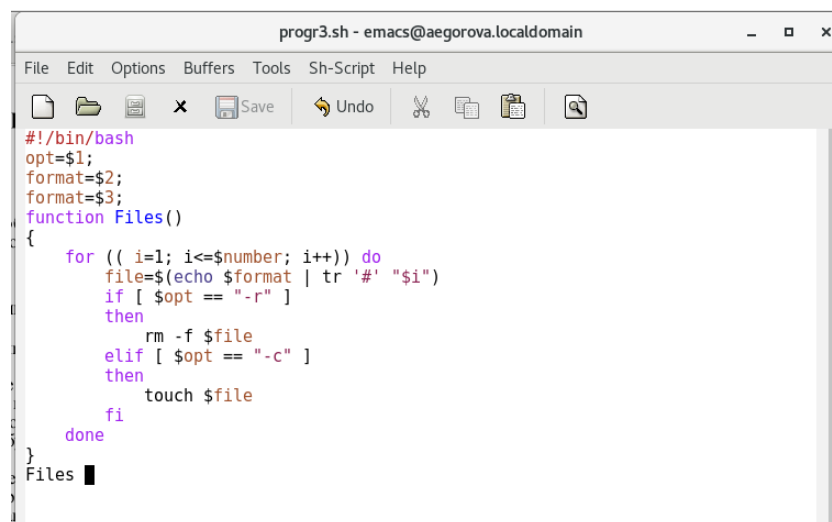
```

[aegorova@aegorova ~]$ touch progr2.sh
[2]+  Done                  emacs
[aegorova@aegorova ~]$ emacs &
[2] 5078
[aegorova@aegorova ~]$ chmod +x progr2.sh
[1]-  Done                  emacs
[2]+  Done                  emacs
[aegorova@aegorova ~]$ ./progr2.sh
Введите число
6
Число больше 0
[aegorova@aegorova ~]$ ./progr2.sh
Введите число
0
Число равно 0
[aegorova@aegorova ~]$ ./progr2.sh
Введите число
-8
Число меньше 0

```

Figure 2.5: Проверка второй программы

- 3) Написала командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и 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: progr3.sh и написала соответствующий скрипт. (рис. -fig. 2.6)



```

#!/bin/bash
opt=$1;
format=$2;
format=$3;
function Files()
{
    for (( i=1; i<=$number; i++)) do
        file=$(echo $format | tr '#' 'i')
        if [ $opt == "-r" ]
        then
            rm -f $file
        elif [ $opt == "-c" ]
        then
            touch $file
        fi
    done
}
Files

```

Figure 2.6: Третья программа

Далее я проверила работу написанного скрипта («./progr3.sh»), предварительно

добавив право на исполнение файла («`chmod +x progr3.sh`»). Сначала я создала три файла («`./progr3.sh -c abc#.txt 3`»), удовлетворяющие условию задачи, а потом удалила их («`./progr3.sh -r abc#.txt 3`»). (рис. -fig. 2.7) (рис. -fig. 2.8)

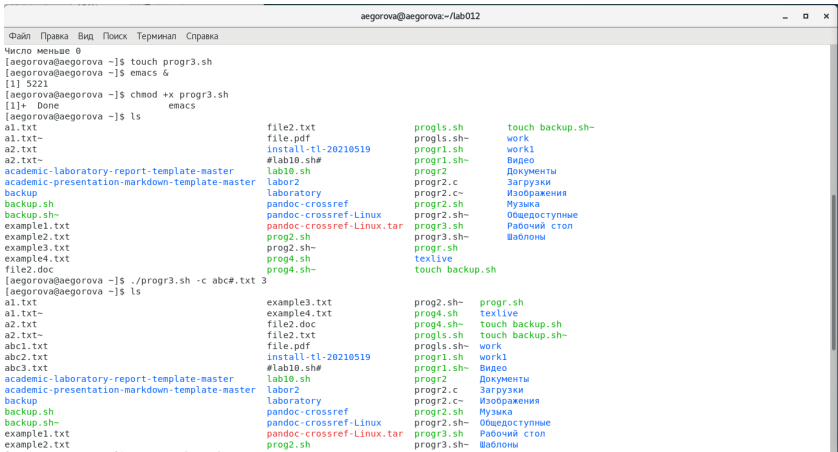
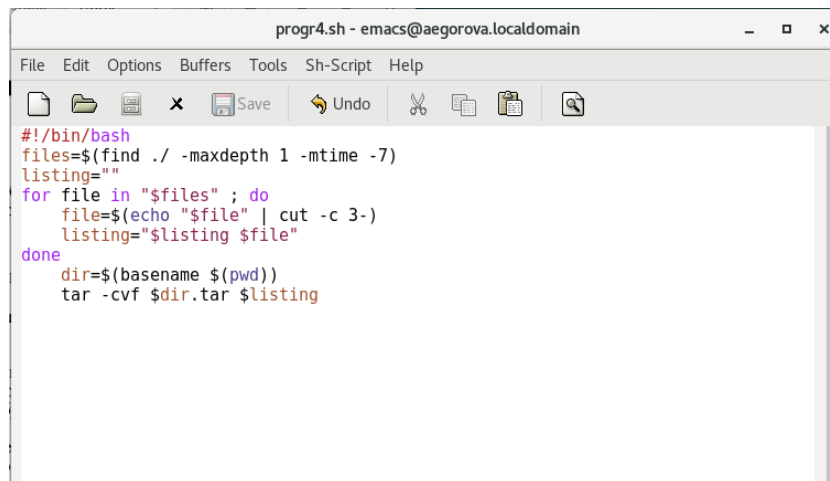


Figure 2.7: Проверка третьей программы



Figure 2.8: Проверка третьей программы

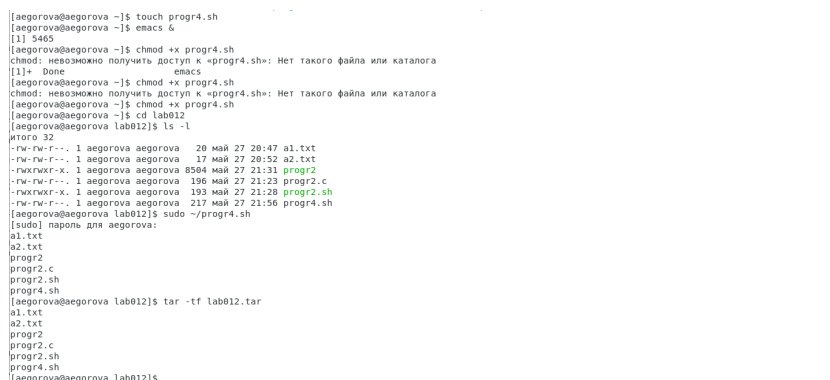
- 4) Написала командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (команда `find`). Для данной задачи я создала файл: `progr4.sh` и написала соответствующий скрипт. (рис. -fig. 2.9)



```
#!/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
```

Figure 2.9: Четвертая программа

Далее я проверила работу написанного скрипта (команды «`sudo ~/prog4.sh`» и «`tar -tf Catalog1.tar`»), предварительно добавив право на исполнение файла (команда «`chmod +x progr4.sh`») и создав отдельный Catalog1 с несколькими файлами. Скрипт работает корректно. (рис. -fig. 2.10)



```
[aegorova@aegorova ~]$ touch progr4.sh
[aegorova@aegorova ~]$ emacs &
[1] 5465
[aegorova@aegorova ~]$ chmod +x progr4.sh
chmod: невозможно получить доступ к «progr4.sh»: Нет такого файла или каталога
[1]~ Done
[aegorova@aegorova ~]$ emacs
[aegorova@aegorova ~]$ chmod +x progr4.sh
chmod: невозможно получить доступ к «progr4.sh»: Нет такого файла или каталога
[aegorova@aegorova ~]$ chmod +x progr4.sh
[aegorova@aegorova ~]$ cd lab012
[aegorova@aegorova lab012]$ ls -l
итого 32
-rw-rw-r--. 1 aegorova aegorova 20 май 27 20:47 a1.txt
-rw-rw-r--. 1 aegorova aegorova 17 май 27 20:52 a2.txt
-rwxrwxr-x. 1 aegorova aegorova 6504 май 27 21:31 progr2
-rw-rw-r--. 1 aegorova aegorova 196 май 27 21:23 progr2.c
-rwxrwxr-x. 1 aegorova aegorova 193 май 27 21:28 progr2.sh
-rw-rw-r--. 1 aegorova aegorova 217 май 27 21:56 progr4.sh
[aegorova@aegorova lab012]$ sudo ~/progr4.sh
[sudo] пароль для aegorova:
a1.txt
a2.txt
progr2
progr2.c
progr2.sh
progr4.sh
[aegorova@aegorova lab012]$ tar -tf lab012.tar
a1.txt
a2.txt
progr2
progr2.c
progr2.sh
progr4.sh
[aegorova@aegorova lab012]$
```

Figure 2.10: Проверка четвертой программы

3 Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

4 Контрольные вопросы

- 1) Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, и 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 введенные данные с помощью оператора `case`. Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, 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, `echo *` – выведет имена всех файлов теку-

- щего каталога, что представляет собой простейший аналог команды `ls`; 2) `ls .c` – выведет все файлы с последними двумя символами, совпадающими с `.c`. 3) `echo prog.?` – выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.`. 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, не равный 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.

5 Библиография

Лабораторная работа No 8. Программирование в командном процессоре ОС UNIX.
Командные файлы