

# **Отчет по лабораторной работе №11**

**Дисциплина: операционные системы**

Лобанова Полина Иннокентьевна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	12
4	Вывод	15

## Список иллюстраций

2.1	Создание необходимых файлов. . . . .	6
2.2	Текст программы. . . . .	7
2.3	Результат. . . . .	7
2.4	Текст программы на языке Си. . . . .	8
2.5	Текст программы. . . . .	8
2.6	Результат. . . . .	9
2.7	Текст программы. . . . .	9
2.8	Результат. . . . .	10
2.9	Текст программы. . . . .	10
2.10	Результат. . . . .	11

## **Список таблиц**

# 1 Цель работы

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

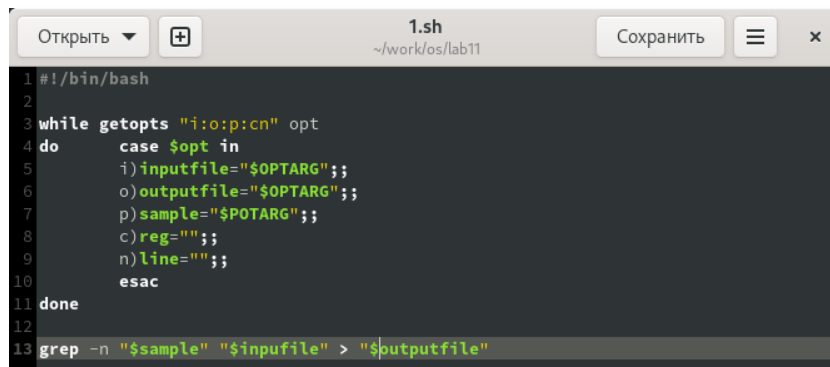
## 2 Выполнение лабораторной работы

1. Создадим все необходимые для дальнейшей работы файлы.

```
[pilobanova@fedora ~]$ mkdir ~/work/os/lab11  
[pilobanova@fedora ~]$ cd ~/work/os/lab11  
[pilobanova@fedora lab11]$ touch 1.sh  
[pilobanova@fedora lab11]$ touch 2.sh  
[pilobanova@fedora lab11]$ touch 3.sh  
[pilobanova@fedora lab11]$ touch 4.sh
```

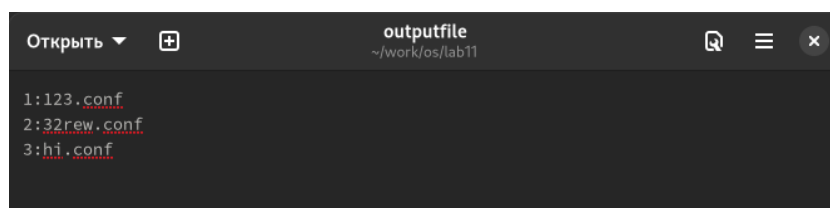
Рис. 2.1: Создание необходимых файлов.

2. Используя команды `getopts` `grep`, напомним командный файл, который анализирует командную строку с ключами: `-iinputfile` — прочитать данные из указанного файла; `-ooutputfile` — вывести данные в указанный файл; `-r` — шаблон — указать шаблон для поиска; `-C` — различать большие и малые буквы; `-n` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-p`.



```
1 #!/bin/bash
2
3 while getopts "i:o:p:cn" opt
4 do     case $opt in
5         i) inputfile="$OPTARG";;
6         o) outputfile="$OPTARG";;
7         p) sample="$OPTARG";;
8         c) reg="";;
9         n) line="";;
10        esac
11 done
12
13 grep -n "$sample" "$inputfile" > "$outputfile"
```

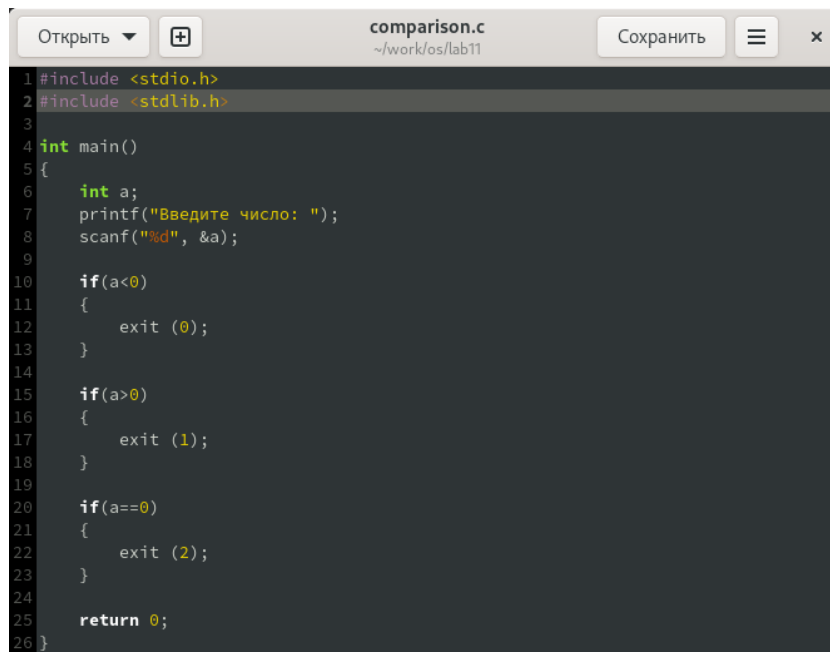
Рис. 2.2: Текст программы.



```
1:123.conf
2:32rew.conf
3:hi.conf
```

Рис. 2.3: Результат.

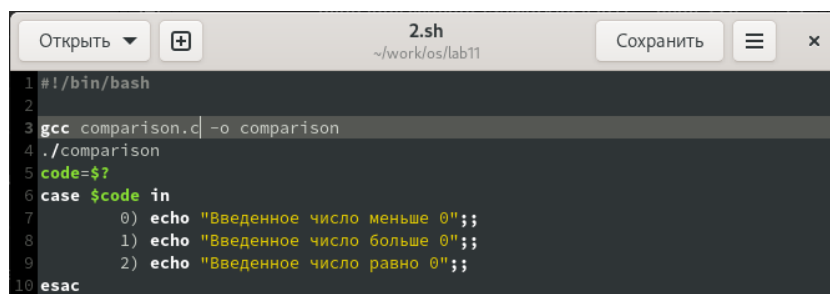
3. Напишем на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.



The screenshot shows a code editor window titled 'comparison.c' with the file path '~/.work/os/lab11'. The editor contains the following C code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int a;
7     printf("Введите число: ");
8     scanf("%d", &a);
9
10    if(a<0)
11    {
12        exit (0);
13    }
14
15    if(a>0)
16    {
17        exit (1);
18    }
19
20    if(a==0)
21    {
22        exit (2);
23    }
24
25    return 0;
26 }
```

Рис. 2.4: Текст программы на языке Си.



The screenshot shows a shell terminal window titled '2.sh' with the file path '~/.work/os/lab11'. The terminal contains the following commands and output:

```
1 #!/bin/bash
2
3 gcc comparison.c -o comparison
4 ./comparison
5 code=$?
6 case $code in
7     0) echo "Введенное число меньше 0";;
8     1) echo "Введенное число больше 0";;
9     2) echo "Введенное число равно 0";;
10 esac
```

Рис. 2.5: Текст программы.



```

[pilobanova@fedora lab11]$ bash 2.sh
Введите число: -6
Введенное число меньше 0
[pilobanova@fedora lab11]$ bash 2.sh
Введите число: 0
Введенное число равно 0
[pilobanova@fedora lab11]$ bash 2.sh
Введите число: 6
Введенное число больше 0

```

Рис. 2.6: Результат.

4. Напишем 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).



```

1#!/bin/bash
2
3while getopts c:r opt
4do
5    case $opt in
6        c)n="$OPTARG";
7        for i in $(seq 1 $n);
8        do touch "$i.tmp";
9        done;;
10       r)for i in $(find -name "*.tmp");
11       do rm $i;
12       done;;
13       esac
14done

```

Рис. 2.7: Текст программы.

```
[pilobanova@fedora lab11]$ bash 3.sh -c 4
[pilobanova@fedora lab11]$ ls
1.sh  2.sh  3.sh  4.sh  arch.txt  comparison.c  outputfile
1.tmp 2.tmp 3.tmp 4.tmp  comparison  conf.txt      res.txt
[pilobanova@fedora lab11]$ bash 3.sh -r 4
[pilobanova@fedora lab11]$ ls
1.sh  3.sh  arch.txt  comparison.c  outputfile
2.sh  4.sh  comparison  conf.txt      res.txt
```

Рис. 2.8: Результат.

5. Напишем командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).



```
1 #!/bin/bash
2
3 while getopts :p opt
4 do
5     case $opt in
6         p) dir="$OPTARG";;
7         esac
8     done
9
10 find $dir -mtime -7 -mtime +0 -type f > arch.txt
11
12 tar -cf res.tar -T arch.txt
```

Рис. 2.9: Текст программы.

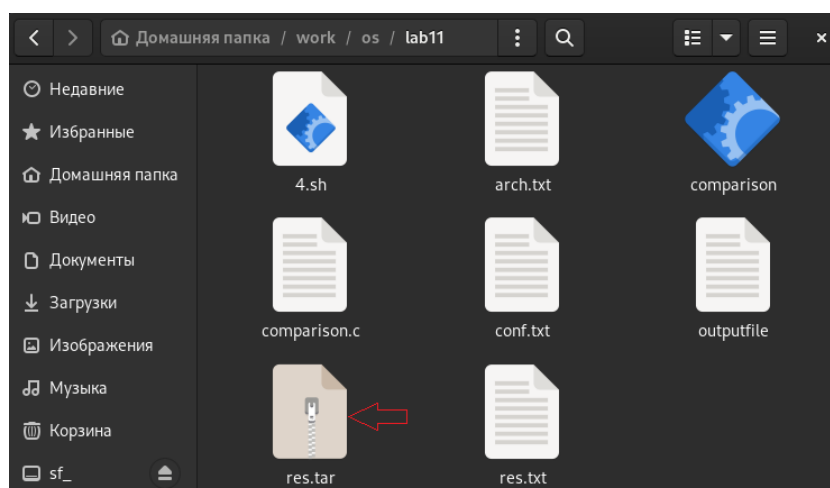


Рис. 2.10: Результат.

### 3 Контрольные вопросы

#### 1. Каково предназначение команды getopt?

Весьма необходимой при программировании является команда `getopt`, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopt option-string variable [arg ...]` Флаги – это опции командной строки, обычно помеченные знаком минус; Например, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopt` может распознать аргумент, она возвращает истину. Принято включать `getopt` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: `testprog -ifile_in.txt -ofile_out.doc -L -t -r` Вот как выглядит использование оператора `getopt` в этом случае: 

```
while getopt o:i:Ltr optletter do
case optletter in
o) oflag=1; oval=OPTARG;;
i) iflag=1; ival=$OPTARG;;
L) Lflag=1;;
t) tflag=1;;
r) rflag=1;;
*) echo Illegal option $optletter
esac
done
```

 Функция `getopt` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равна `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упомянутый

аргумент. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

## 2. Какое отношение метасимволы имеют к генерации имён файлов?

- `-` — соответствует произвольной, в том числе и пустой строке; • `?` — соответствует любому одному символу; • `[c1-c1]` — соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`. • `echo *` — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; • `ls .c` — выведет все файлы с последними двумя символами, равными `.c`. • `echo prog.?` — выдаст 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.

## 3. Какие операторы управления действиями вы знаете?

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 Вам 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 дела являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда.

## 4. Какие операторы используются для прерывания цикла?

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 перестает быть правильным. Пример бесконечного цикла while, с прерыванием в момент, когда файл перестает существовать: `while true do if [! -f $file] then break fi sleep 10 done`

5. Для чего нужны команды false и true?

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.

6. Что означает строка `if test -f mans/i.$$`, встреченная в командном файле?

Введенная строка означает условие существования файла `mans/i.$$`

7. Объясните различия между конструкциями while и until.

Если речь идет о 2-х параллельных действиях, то это while. когда мы показываем, что сначала делается 1-е действие. потом оно заканчивается при наступлении 2-го действия, применяем until.

## 4 Вывод

Я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.