

# **Отчет по лабораторной работе №11**

**Операционные системы**

Шуваев Сергей Александрович

# Содержание

<b>1</b>	<b>Задание</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>4</b>	<b>Выводы</b>	<b>15</b>
<b>5</b>	<b>Ответы на контрольные вопросы</b>	<b>16</b>

## Список иллюстраций

3.1	Создаю файл с разрешением на исполнение . . . . .	8
3.2	Командный файл, с командами <code>getopts</code> и <code>grep</code> , который анализирует командную строку с ключами: <code>iinputfile</code> — прочитать данные из указанного файла; <code>-ooutputfile</code> — вывести данные в указанный файл; <code>-p</code> шаблон — указать шаблон для поиска; <code>-C</code> — различать большие и малые буквы; <code>-n</code> — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом <code>-p</code> . . . . .	9
3.3	Результат работы программы в файле <code>output.txt</code> . . . . .	10
3.4	Создаю исполняемый файл для второй программы, также создаю файл <code>12.c</code> для программы на Си . . . . .	10
3.5	Пишу программу на языке Си, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции <code>exit(n)</code> , передавая информацию в о коде завершения в оболочку . . . . .	11
3.6	Командный файл должен вызывать эту программу и, проанализировав с помощью команды <code>\$?</code> , выдать сообщение о том, какое число было введено . . . . .	12
3.7	Программа работает корректно . . . . .	12
3.8	Создаю исполняемый файл для третьей программы . . . . .	13
3.9	Командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например <code>1.tmp</code> , <code>2.tmp</code> , <code>3.tmp</code> , <code>4.tmp</code> 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 . . . . .	13
3.10	Проверяю, что программа создала файлы и удалила их при соответствующих запросах . . . . .	14
3.11	Создаю исполняемый файл для четвертой программы. Это командный файл, который с помощью команды <code>tar</code> 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 <code>find</code> ) . . . . .	14
3.12	Проверяю работу программы . . . . .	14

## **Список таблиц**

# 1 Задание

Используя команды `getopts` `grep`, написать 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 `-r`. 2. Написать 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. 3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до  $\infty$  (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). 4. Написать 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`).

## 2 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

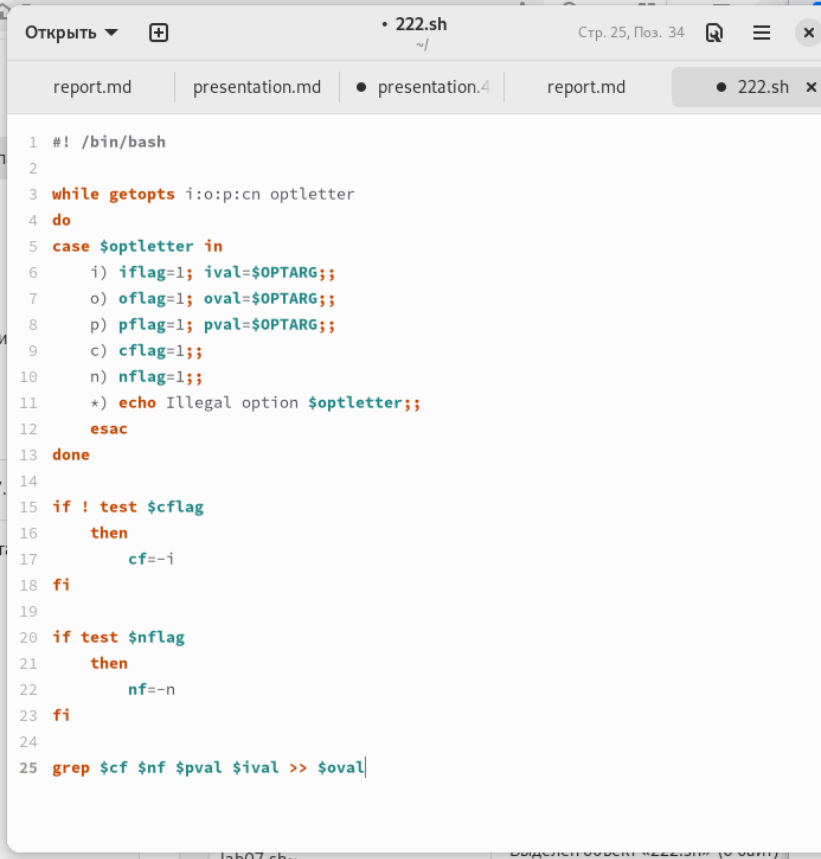
описанными ниже.

### 3 Выполнение лабораторной работы

```
[sergeyshuwaew@sergeyshuwaew ~]$ touch 222.sh  
[sergeyshuwaew@sergeyshuwaew ~]$ chmod +x 222.sh  
[sergeyshuwaew@sergeyshuwaew ~]$ bash 222.sh -p улит -i input.txt -o output.txt  
-c -n  
[sergeyshuwaew@sergeyshuwaew ~]$
```

Рис. 3.1: Создаю файл с разрешением на исполнение





```
1 #!/bin/bash
2
3 while getopts i:o:p:cn optletter
4 do
5 case $optletter in
6   i) iflag=1; ival=$OPTARG;;
7   o) oflag=1; oval=$OPTARG;;
8   p) pflag=1; pval=$OPTARG;;
9   c) cflag=1;;
10  n) nflag=1;;
11  *) echo Illegal option $optletter;;
12  esac
13 done
14
15 if ! test $cflag
16 then
17   cf=-i
18 fi
19
20 if test $nflag
21 then
22   nf=-n
23 fi
24
25 grep $cf $nf $pval $ival >> $oval
```

Рис. 3.2: Командный файл, с командами `getopts` и `grep`, который анализирует командную строку с ключами: `i`inputfile — прочитать данные из указанного файла; `-o`outputfile — вывести данные в указанный файл; `-p` 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`

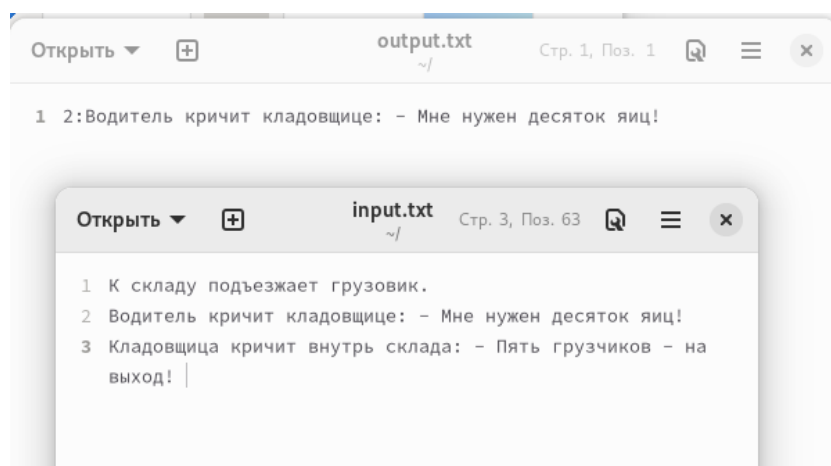
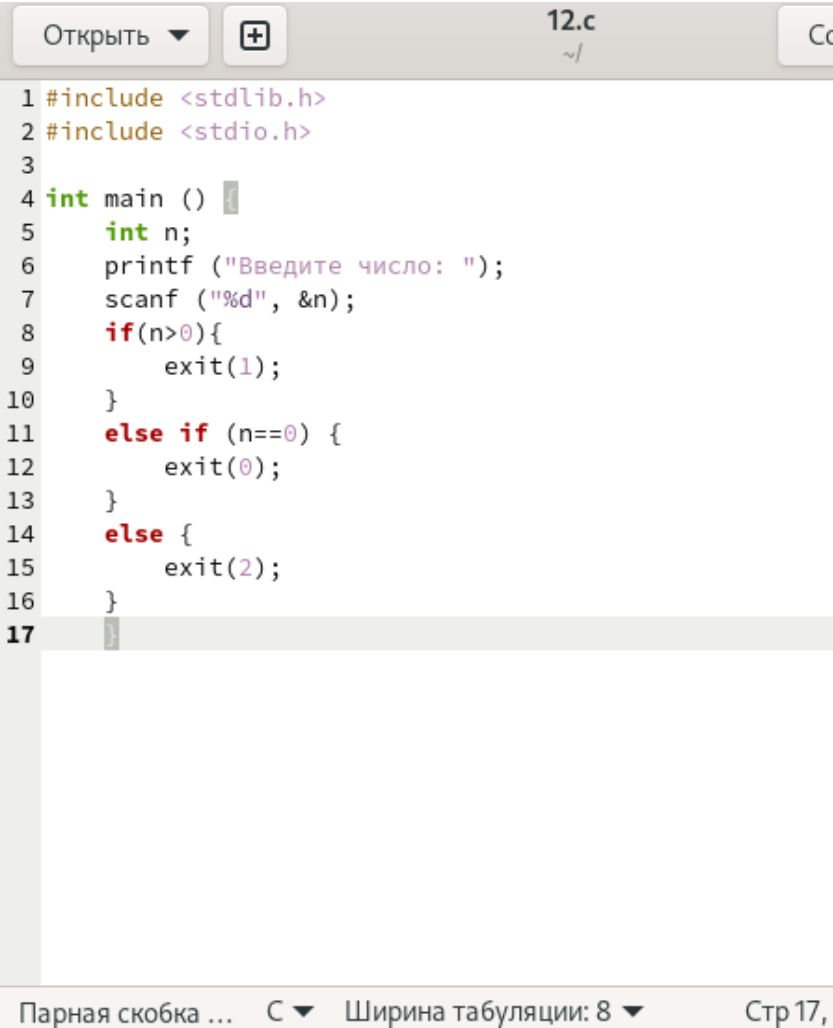


Рис. 3.3: Результат работы программы в файле output.txt

```
[sergeyshuwaew@sergeyshuwaew ~]$ touch 222.sh
[sergeyshuwaew@sergeyshuwaew ~]$ chmod +x 222.sh
[sergeyshuwaew@sergeyshuwaew ~]$ touch 12.cpp
```

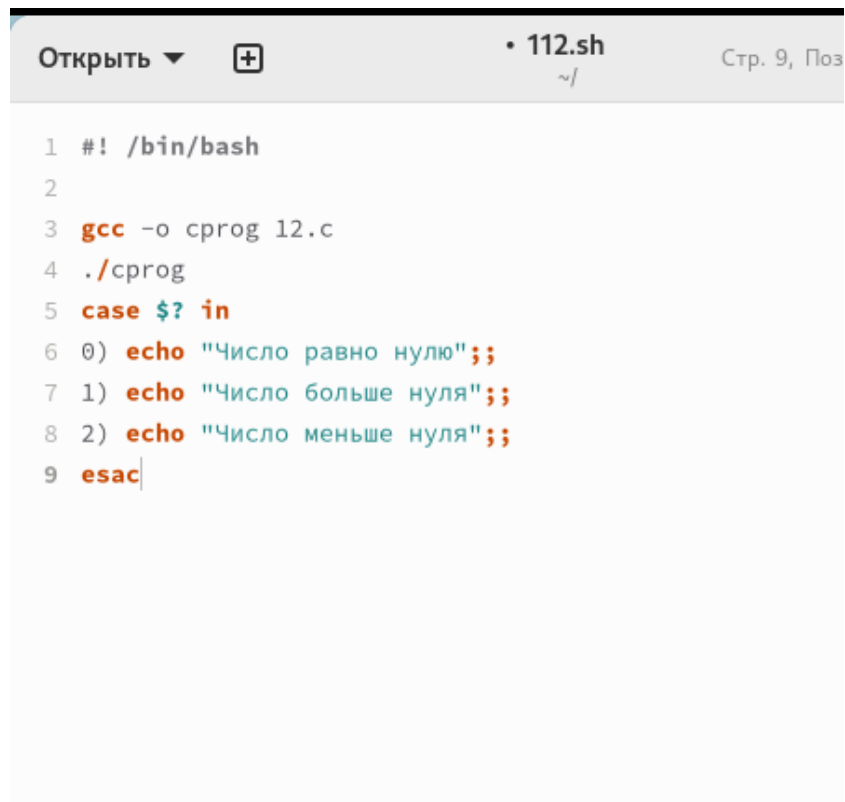
Рис. 3.4: Создаю исполняемый файл для второй программы, также создаю файл 12.c для программы на Си



```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main () {
5     int n;
6     printf ("Введите число: ");
7     scanf ("%d", &n);
8     if(n>0){
9         exit(1);
10    }
11    else if (n==0) {
12        exit(0);
13    }
14    else {
15        exit(2);
16    }
17 }
```

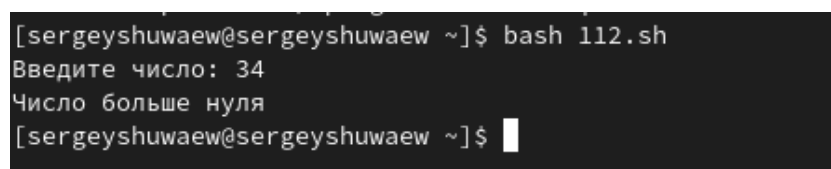
Парная скобка ... С ▾ Ширина табуляции: 8 ▾ Стр 17,

Рис. 3.5: Пишу программу на языке Си, 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



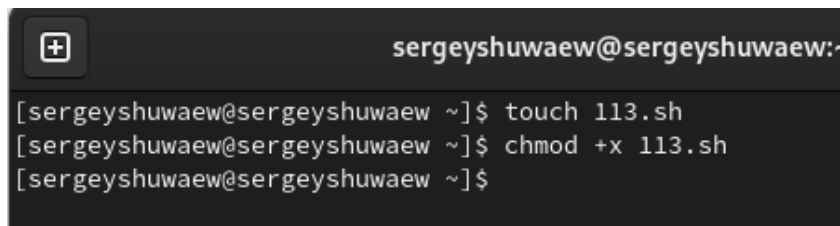
```
Открыть ▾ + • 112.sh ~/  
Стр. 9, Поз  
1 #!/bin/bash  
2  
3 gcc -o cprog 12.c  
4 ./cprog  
5 case $? in  
6 0) echo "Число равно нулю";;  
7 1) echo "Число больше нуля";;  
8 2) echo "Число меньше нуля";;  
9 esac
```

Рис. 3.6: Командный файл должен вызывать эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено



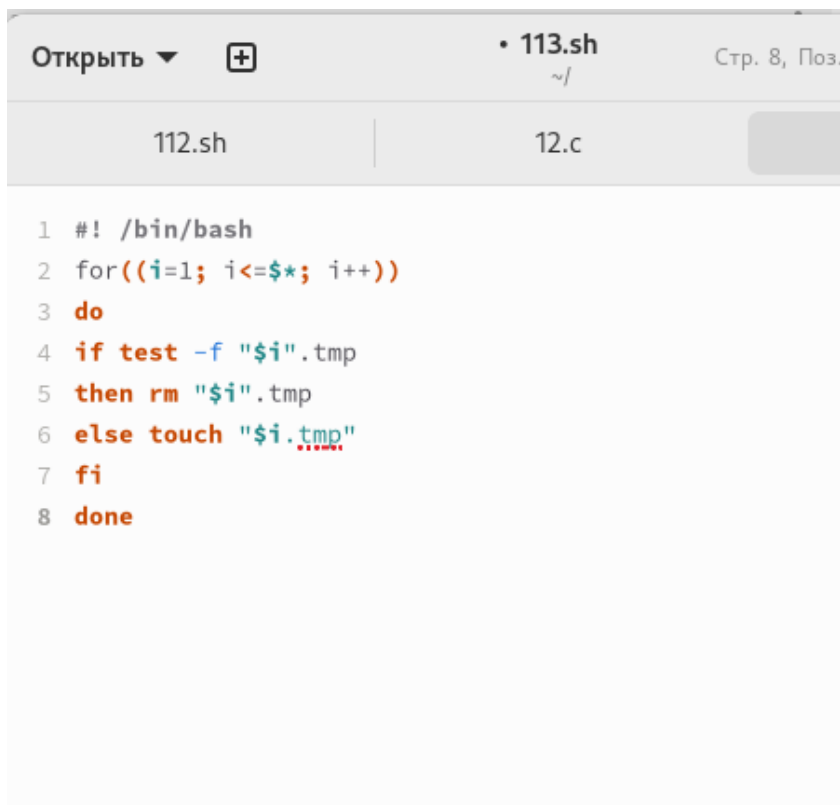
```
[sergeyshuwaew@sergeyshuwaew ~]$ bash 112.sh  
Введите число: 34  
Число больше нуля  
[sergeyshuwaew@sergeyshuwaew ~]$
```

Рис. 3.7: Программа работает корректно



```
sergeyshuwaew@sergeyshuwaew:~$ touch 113.sh
sergeyshuwaew@sergeyshuwaew:~$ chmod +x 113.sh
sergeyshuwaew@sergeyshuwaew:~$
```

Рис. 3.8: Создаю исполняемый файл для третьей программы



```
Открыть ▾ + • 113.sh ~/ Стр. 8, Поз.
112.sh | 12.c
1  #!/bin/bash
2  for((i=1; i<=$*; i++))
3  do
4  if test -f "$i".tmp
5  then rm "$i".tmp
6  else touch "$i.tmp"
7  fi
8  done
```

Рис. 3.9: 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)

```
[sergeyshuwaew@sergeyshuwaew ~]$ bash 113.sh 4
[sergeyshuwaew@sergeyshuwaew ~]$ ls
111.sh  family      lab07.sh  play      Видео
112.sh  feathers    lab07.sh~ prog1.sh   Документы
113.sh  file.txt    letters   prog2.sh   Загрузки
12.c    fun         memos     prog3.sh   Изображения
abc1    input.txt   misk      prog4.sh   Музыка
australia '##lab07##' monthly   reports    Общедоступные
backup  '#lab07#'  montly.00 ski.places 'Рабочий стол'
bin     lab07      my_os     tutorial   Шаблоны
cprog   '#lab07.sh#' output.txt work
```

Рис. 3.10: Проверяю, что программа создала файлы и удалила их при соответствующих запросах



```
114.sh
Стр. 3, Поз. 33

report.md  114.sh x

1  #!/bin/bash
2  find $* -mtime -7 -mtime +0 -type f > FILES.txt
3  tar -cf archive.tar -T FILES.txt
```

Рис. 3.11: Создаю исполняемый файл для четвертой программы. Это 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)

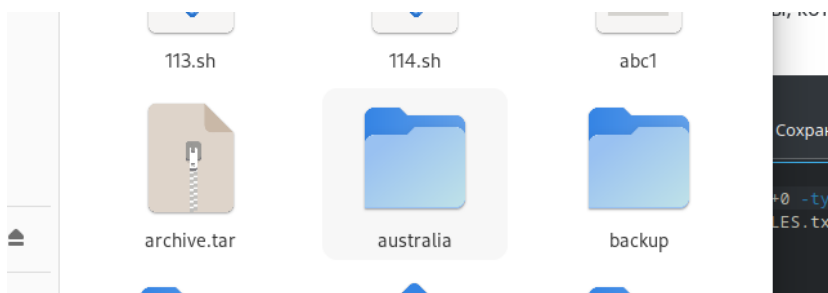


Рис. 3.12: Проверяю работу программы

## 4 Выводы

При выполнении данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 5 Ответы на контрольные вопросы

1. Каково предназначение команды `getopts`? Осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable`. 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 `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` 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 `getopts` в этом случае: 

```
while getopts 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 `getopts` 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 упо-



мянутый 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? 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.
3. Какие операторы управления действиями вы знаете? Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов 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. 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 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. Для чего нужны команды `false` и `true`? 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).
6. Что означает строка `if test -f mans/i.s, ?if test -f mans/i.s` проверяет, существует ли файл `mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).
7. Объясните различия между конструкциями `while` и `until`. 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.