

# **Отчёт по лабораторной работе № 13**

**Программирование в командном процессоре ОС UNIX. Ветвления и  
циклы**

Дагделен Зейнап Реджеповна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
<b>5</b>	<b>Выводы</b>	<b>17</b>
<b>6</b>	<b>Ответы на контрольные вопросы</b>	<b>18</b>
<b>7</b>	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

4.1	Создание файла . . . . .	10
4.2	Код программы . . . . .	11
4.3	Результат работы программы . . . . .	12
4.4	Создание файла . . . . .	12
4.5	Код программы на Си . . . . .	13
4.6	Код программы . . . . .	14
4.7	Результат работы программы . . . . .	14
4.8	Создание файла . . . . .	15
4.9	Код программы . . . . .	15
4.10	Результат работы программы . . . . .	16
4.11	Код программы . . . . .	16
4.12	Результат работы программы . . . . .	16

## Список таблиц

# 1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 2 Задание

1. 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;
  - `-р`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 `-р`.
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).

### 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).

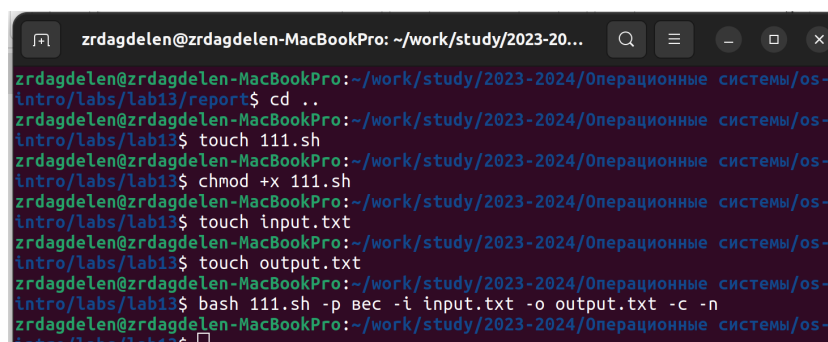
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных 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.

## 4 Выполнение лабораторной работы

Создаю файл с разрешением на исполнение (рис. fig. 4.1).



```
zrdagdelen@zrdagdelen-MacBookPro: ~/work/study/2023-20...
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab13/report$ cd ..
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab13$ touch 111.sh
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab13$ chmod +x 111.sh
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab13$ touch input.txt
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab13$ touch output.txt
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab13$ bash 111.sh -p vec -i input.txt -o output.txt -c -n
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab13$
```

Рис. 4.1: Создание файла

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; `-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` (рис. fig. 4.2).

```
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. 4.2: Код программы

```
#!/bin/bash
```

```
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 ! test $cflag
then
    cf=-i
fi
```

```
if test $nflag
then
    nf=-n
fi
```

```
grep $cf $nf $pval $ival >> $oval
```

Результат работы программы в файле output.txt (рис. fig. 4.3).

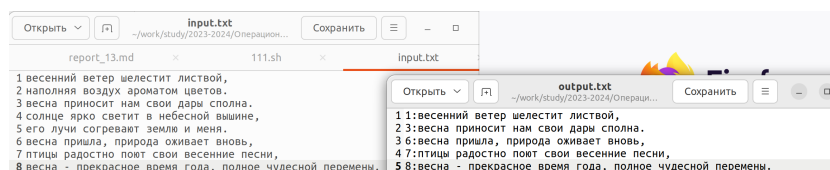


Рис. 4.3: Результат работы программы

Создаю исполняемый файл для второй программы, также создаю файл 12.c для программы на Си (рис. fig. 4.4).

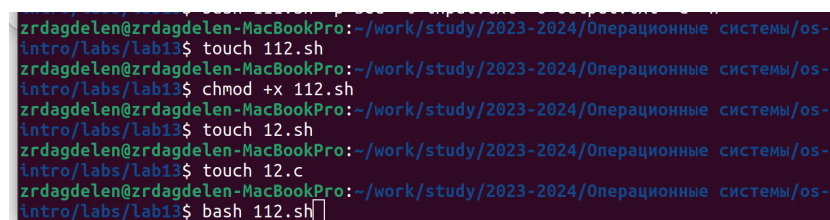
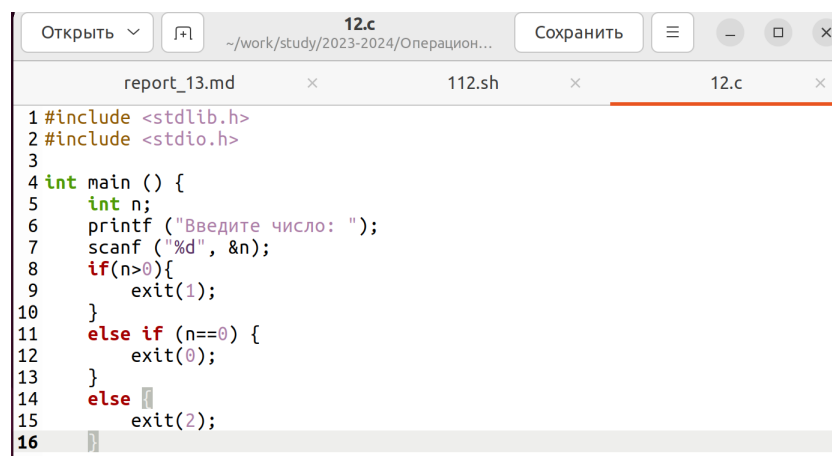


Рис. 4.4: Создание файла

Пишу программу на языке Си, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершает-

ся с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку (рис. fig. 4.5).



```
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    }
```

Рис. 4.5: Код программы на Си

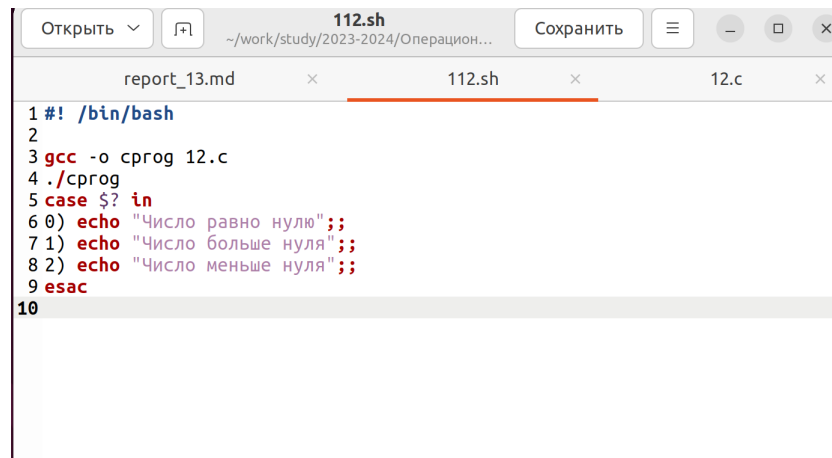
```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
int main () {
    int n;
    printf ("Введите число: ");
    scanf ("%d", &n);
    if(n>0){
        exit(1);
    }
    else if (n==0) {
        exit(0);
    }
    else {
        exit(2);
    }
}
```

Командный файл должен вызывать эту программу и, проанализировав с по-

мощью команды `$?`, выдать сообщение о том, какое число было введено (рис. fig. 4.6).



```
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
10
```

Рис. 4.6: Код программы

```
#!/bin/bash
```

```
gcc -o cprog 12.c
```

```
./cprog
```

```
case $? in
```

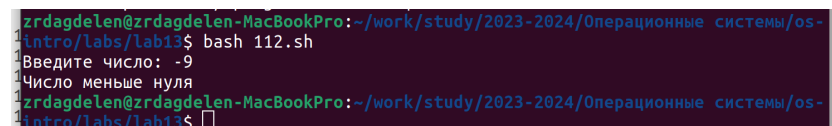
```
0) echo "Число равно нулю";;
```

```
1) echo "Число больше нуля";;
```

```
2) echo "Число меньше нуля";;
```

```
esac
```

Программа работает корректно (рис. fig. 4.7).



```
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ bash 112.sh
Введите число: -9
Число меньше нуля
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$
```

Рис. 4.7: Результат работы программы

Создаю исполняемый файл для третьей программы (рис. fig. 4.8).

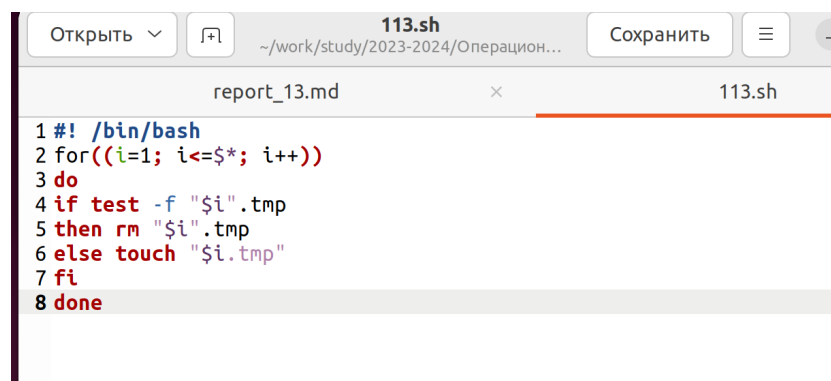
```

zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ touch 113.sh
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ chmod +x 113.sh
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$

```

Рис. 4.8: Создание файла

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) (рис. fig. 4.9).



```

113.sh
~/work/study/2023-2024/Операцион...
report_13.md x 113.sh
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. 4.9: Код программы

```

#!/bin/bash

for((i=1; i<=*$*; i++))
do
if test -f "$i".tmp
then rm "$i".tmp
else touch "$i.tmp"
fi
done

```

Проверяю, что программа создала файлы и удалила их при соответствующих запросах (рис. fig. 4.10).

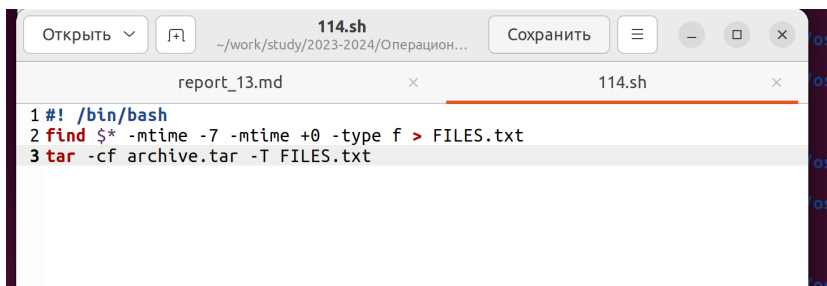
```

zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ bash 113.sh 5
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ ls
111.sh 113.sh 1.tmp 3.tmp 5.tmp input.txt presentation
112.sh 12.c 2.tmp 4.tmp cprog output.txt report
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ bash 113.sh 5
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ ls
111.sh 113.sh cprog output.txt report
112.sh 12.c input.txt presentation
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$

```

Рис. 4.10: Результат работы программы

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. fig. 4.11).



```

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

```

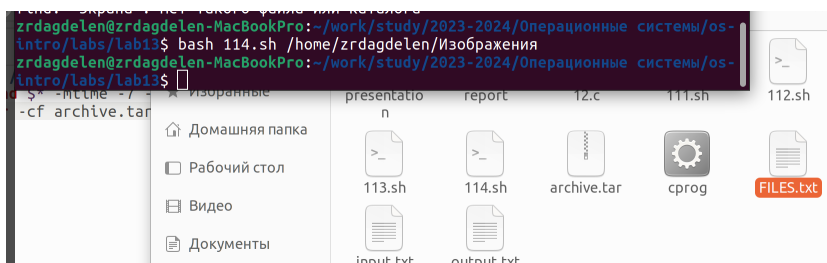
Рис. 4.11: Код программы

```

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

```

Проверяю работу программы (рис. fig. 4.12).



```

zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$ bash 114.sh /home/zrdagdelen/Изображения
zrdagdelen@zrdagdelen-MacBookPro:~/work/study/2023-2024/Операционные системы/os-
intro/labs/lab1$

```

Рис. 4.12: Результат работы программы



## 5 Выводы

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

## 6 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) iflag = 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 упомянутый аргумент. Функция `getopts` также понимает переменные типа массив, следова-

тельно, 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`, встреченная в командном файле?

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. 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.

## **7 Список литературы**

ОС лабораторная 13