

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ”

Факультет физико-математических и естественных наук

ОТЧЕТ

По лабораторной работе №11
“Программирование в командном процессоре ОС UNIX. Ветвления и
циклы”

Выполнил:
Студент группы: НПИбд-02-21
Студенческий билет: №1032217060
ФИО студента: Королев Адам Маратович
Дата выполнения: 28.05.2022

Москва 2022

1 Цель работы:

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

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;
- С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С 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.

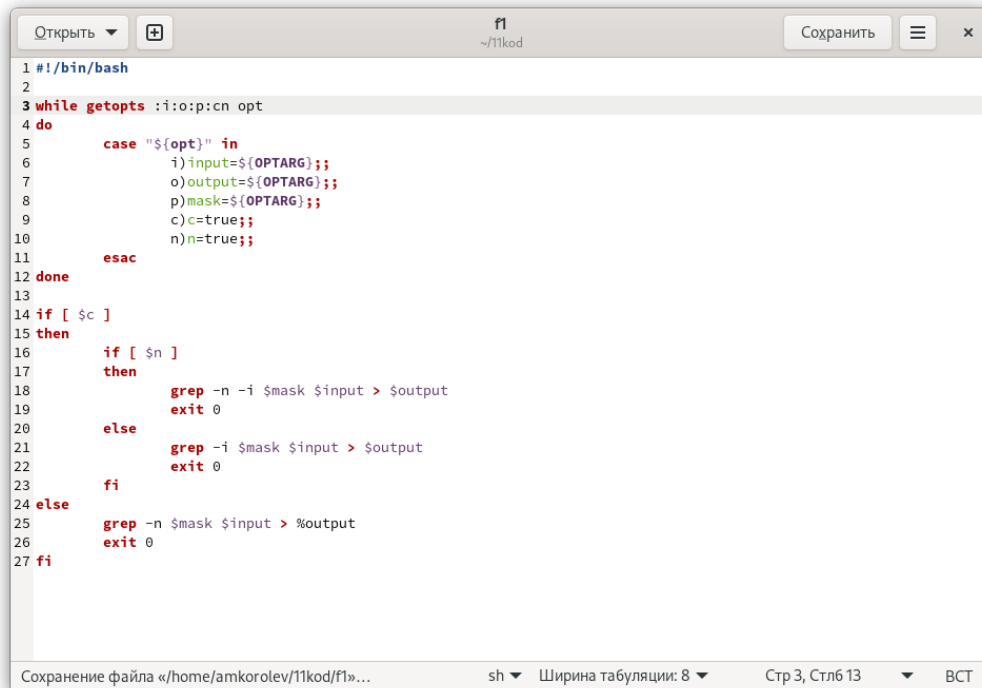
POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

3 Выполнение лабораторной работы:

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; — `-rшаблон` — 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.

Пишем скрипт.

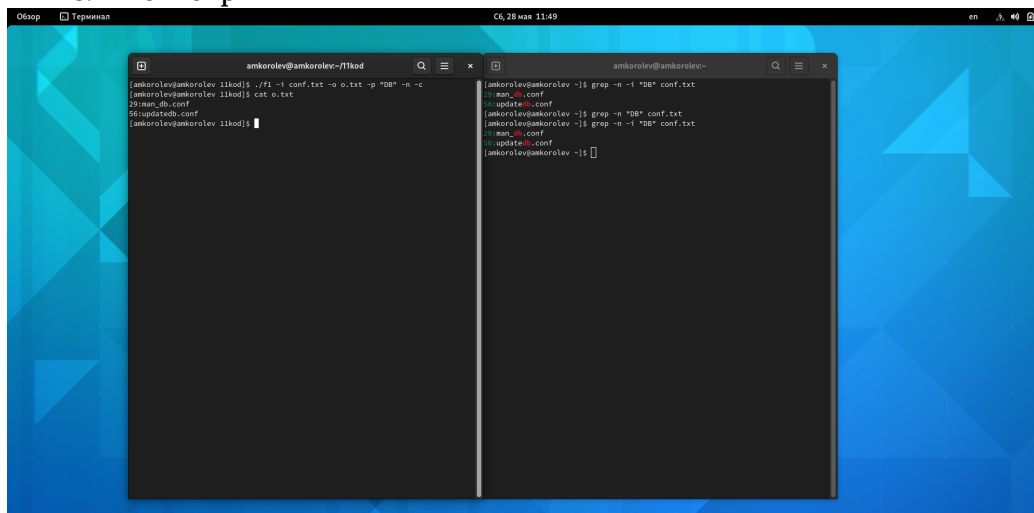


The screenshot shows a text editor window with a light gray title bar containing buttons for 'Открыть', '+', 'f1', '~/.11kod', 'Сохранить', and window controls. The script content is as follows:

```
1 #!/bin/bash
2
3 while getopts :i:o:p:cn opt
4 do
5     case "${opt}" in
6         i) input=${OPTARG};;
7         o) output=${OPTARG};;
8         p) mask=${OPTARG};;
9         c) c=true;;
10        n) n=true;;
11    esac
12 done
13
14 if [ $c ]
15 then
16     if [ $n ]
17     then
18         grep -n -i $mask $input > $output
19         exit 0
20     else
21         grep -i $mask $input > $output
22         exit 0
23     fi
24 else
25     grep -n $mask $input > $output
26     exit 0
27 fi
```

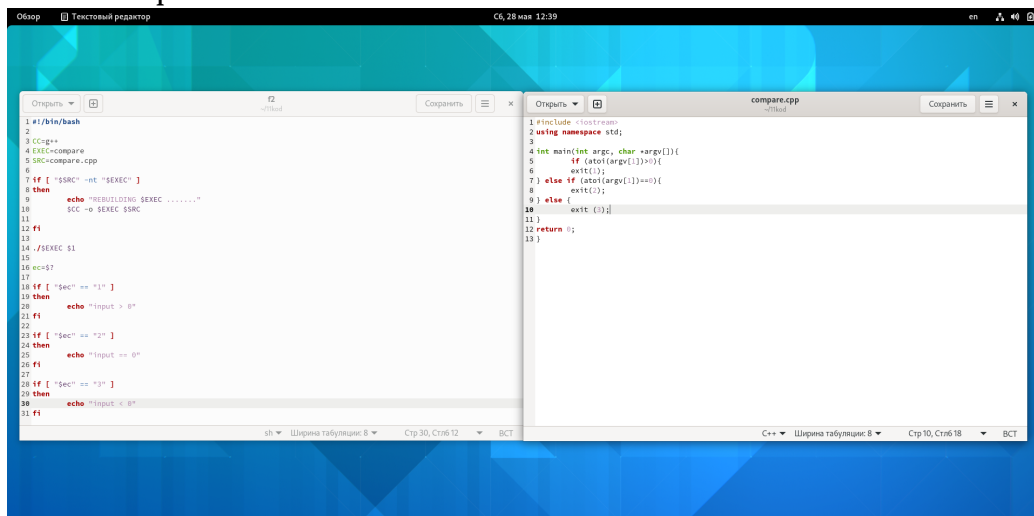
The status bar at the bottom indicates: 'Сохранение файла «/home/amkorolev/11kod/f1»...', 'sh', 'Ширина табуляции: 8', 'Стр 3, Стлб 13', and 'ВСТ'.

Выполняем скрипт



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.

Пишем скрипт.



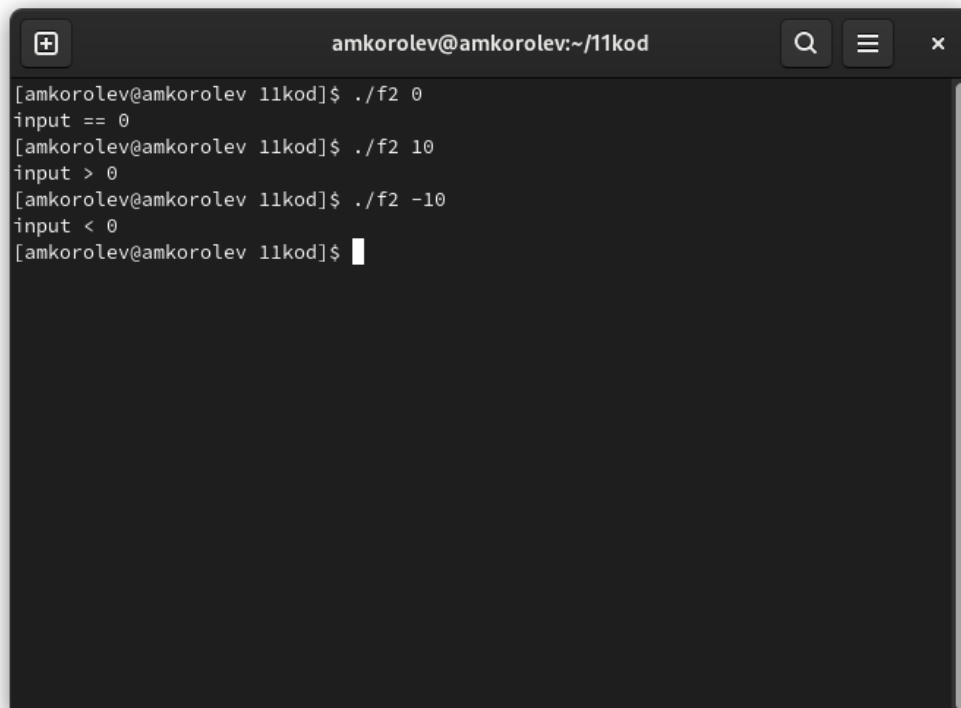
The screenshot shows a code editor with two open files. The left file is a shell script named `if2` with the following content:

```
1 #!/bin/bash
2
3 CC=g++
4 EXEC=compare
5 SRC=compare.cpp
6
7 if [ "$SRC" = "$EXEC" ]
8 then
9     echo "REBUILDING $EXEC ....."
10    gcc -o $EXEC $SRC
11
12 fi
13
14 ./EXEC $1
15
16 ec=$?
17
18 if [ "$ec" = "1" ]
19 then
20     echo "input > 0"
21 fi
22
23 if [ "$ec" = "2" ]
24 then
25     echo "input == 0"
26 fi
27
28 if [ "$ec" = "3" ]
29 then
30     echo "input < 0"
31 fi
```

The right file is a C++ program named `compare.cpp` with the following content:

```
1 #include <iostream>
2 using namespace std;
3
4 int main(int argc, char *argv[]){
5     if (atoi(argv[1])>0){
6         exit(1);
7     } else if (atoi(argv[1])==0){
8         exit(2);
9     } else {
10         exit(3);
11     }
12 return 0;
13 }
```

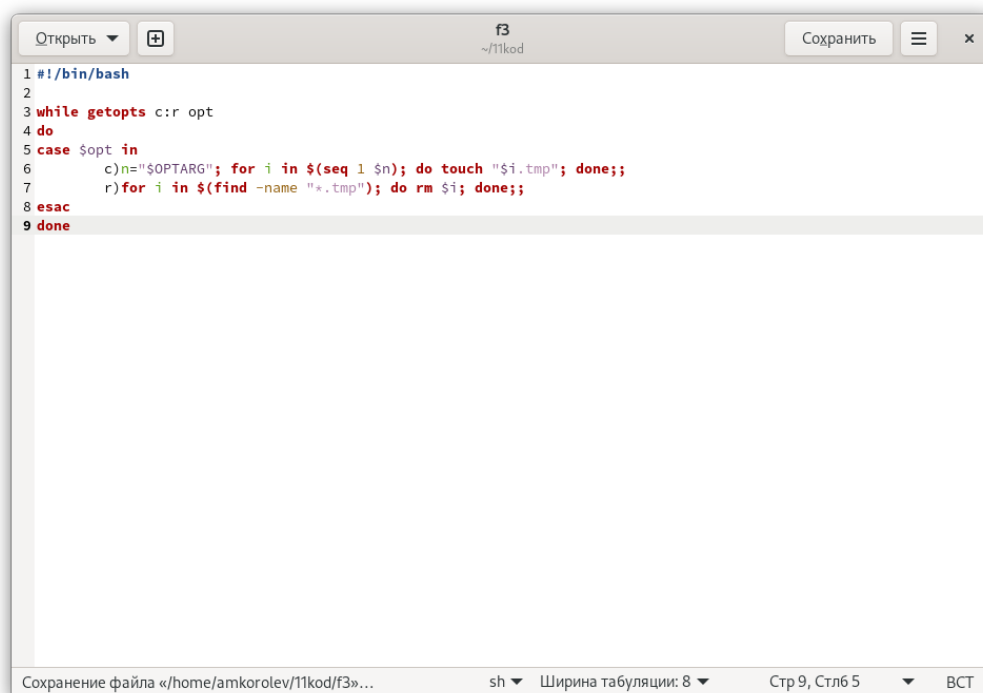
Выполняем скрипт



```
amkorolev@amkorolev:~/11kod
[amkorolev@amkorolev 11kod]$ ./f2 0
input == 0
[amkorolev@amkorolev 11kod]$ ./f2 10
input > 0
[amkorolev@amkorolev 11kod]$ ./f2 -10
input < 0
[amkorolev@amkorolev 11kod]$
```

3. Написать 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).

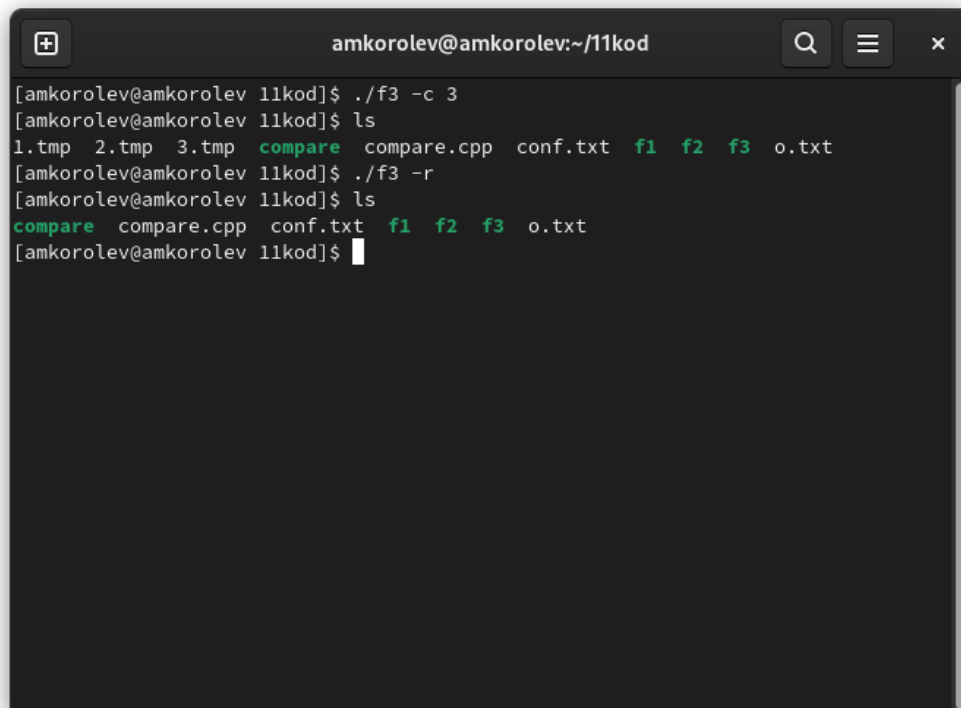
Пишем скрипт.



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

Сохранение файла «/home/amkorolev/11kod/f3»... sh Ширина табуляции: 8 Стр 9, Стлб 5 ВСТ


Выполняем скрипт

A terminal window titled 'amkorolev@amkorolev:~/11kod' with search, menu, and close buttons. The terminal shows a sequence of commands and their outputs. The user runs './f3 -c 3', then 'ls', which lists files including 'compare' in green. Then './f3 -r' is run, followed by another 'ls' command that shows the same file list with 'compare' in green. The prompt ends with a cursor.

```
[amkorolev@amkorolev 11kod]$ ./f3 -c 3
[amkorolev@amkorolev 11kod]$ ls
1.tmp 2.tmp 3.tmp compare compare.cpp conf.txt f1 f2 f3 o.txt
[amkorolev@amkorolev 11kod]$ ./f3 -r
[amkorolev@amkorolev 11kod]$ ls
compare compare.cpp conf.txt f1 f2 f3 o.txt
[amkorolev@amkorolev 11kod]$
```

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`).

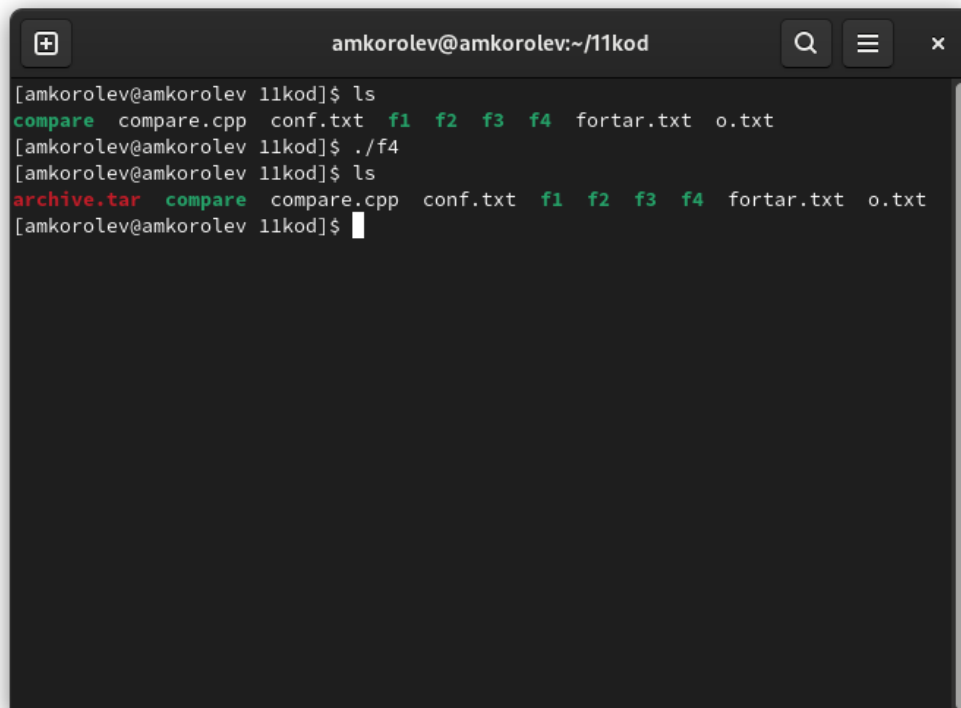
Пишем скрипт.



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

Сохранение файла «/home/amkorolev/11kod/f4»... sh Ширина табуляции: 8 Стр 11, Стлб 35 ВСТ

Выполняем скрипт



```
amkorolev@amkorolev:~/11kod
[amkorolev@amkorolev 11kod]$ ls
compare  compare.cpp  conf.txt  f1  f2  f3  f4  fortar.txt  o.txt
[amkorolev@amkorolev 11kod]$ ./f4
[amkorolev@amkorolev 11kod]$ ls
archive.tar  compare  compare.cpp  conf.txt  f1  f2  f3  f4  fortar.txt  o.txt
[amkorolev@amkorolev 11kod]$
```

4 Выводы:

- В процессе выполнения работы научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Ответы на контрольные вопросы:

1. Весьма необходимой при программировании является команда `getopts`, которая 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, `-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, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

2. При перечислении имен файлов текущего каталога можно использовать следующие символы: `*` — 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. 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. 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 $file] then break fi sleep 10 done`
5. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.
6. Введенная строка означает условие существования файла `mans/i.$s`
7. 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`.