

# **Отчёт по лабораторной работе №11**

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

Желдакова Виктория Алексеевна

# Содержание

<b>1</b>	<b>Цель работы</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>
3.1	Первое задание . . . . .	8
3.2	Второе задание . . . . .	9
3.3	Третье задание . . . . .	10
3.4	Четвёртое задание . . . . .	11
<b>4</b>	<b>Выводы</b>	<b>13</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>14</b>

## Список иллюстраций

3.1	Первый скрипт . . . . .	8
3.2	Запуск первого скрипта с разными флагами . . . . .	9
3.3	Программа на языке C++ . . . . .	9
3.4	Второй скрипт . . . . .	10
3.5	Запуск второго скрипта с разными числами . . . . .	10
3.6	Третий скрипт . . . . .	11
3.7	Запуск третьего скрипта с флагом удаления и создания файлов . .	11
3.8	Четвёртый скрипт . . . . .	12
3.9	Запуск четвёртого скрипта и проверка выполнения . . . . .	12

## 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 Теоретическое введение

Команда `getopts` является встроенной командой командной оболочки `bash`, предназначенной для разбора параметров сценариев. Она обрабатывает исключительно однобуквенные параметры как с аргументами, так и без них и этого вполне достаточно для передачи сценариям любых входных данных.

Базовый синтаксис команды выглядит следующим образом:

`$ getopts строка-параметров переменная [набор-параметров]`

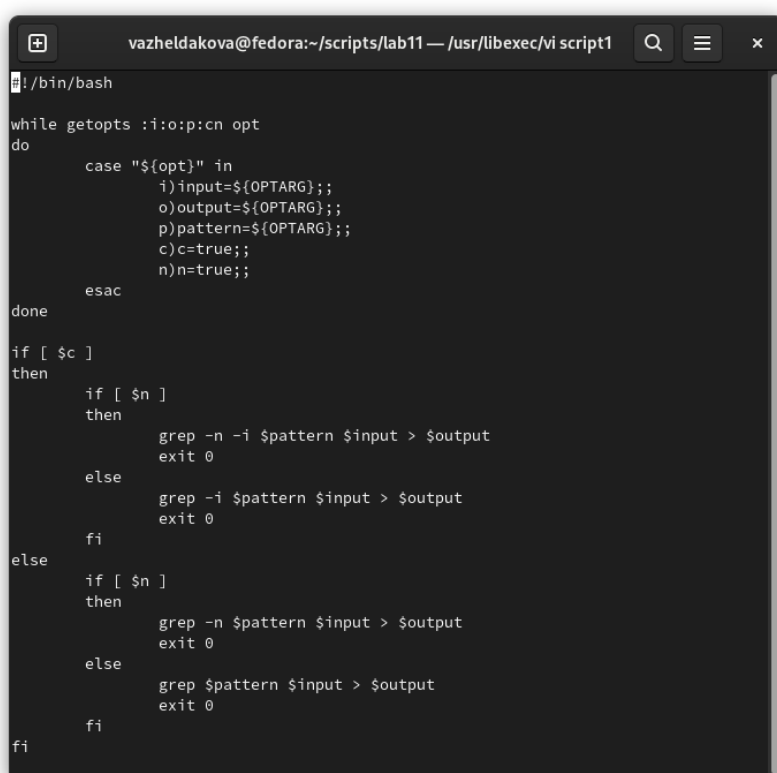
Строка параметров формируется из принимаемых сценарием однобуквенных параметров с символами двоеточия после тех из них, которые должны использоваться совместно с аргументами (символ двоеточия в начале строки активирует режим сокрытия сообщений об ошибках). Переменная используется для хранения текущего параметра. Набор параметров позволяет осуществлять разбор пользовательского списка параметров вместо параметров сценария (используется достаточно редко). В ходе обработки параметров сценария могут использоваться такие переменные, как переменная `$OPTARG`, содержащая аргумент текущего параметра, переменная `$OPTIND`, содержащая номер следующего за текущим параметра и переменная `$OPTERR`, содержащая значение 0 или 1, указывающее на необходимость вывода сообщений об ошибках при обработке каждого из параметров (перед обработкой каждого из параметров устанавливается значение 1, поэтому вам придется самостоятельно устанавливать значение 0 для сокрытия сообщений об ошибках). Чаще всего обработка параметров осуществляется в рамках цикла `while` с вложенной конструкцией `switch-case`, причем в случае необходимости обработки

дополнительных аргументов сценариев (например, в случае передачи списка файлов) после цикла используется конструкция `shift ((OPTIND - 1))`. Команда корректно обрабатывает как отдельно переданные параметры (например, `-a -b -c`), так и объединенные параметры (например, `-abc`).

## 3 Выполнение лабораторной работы

### 3.1 Первое задание

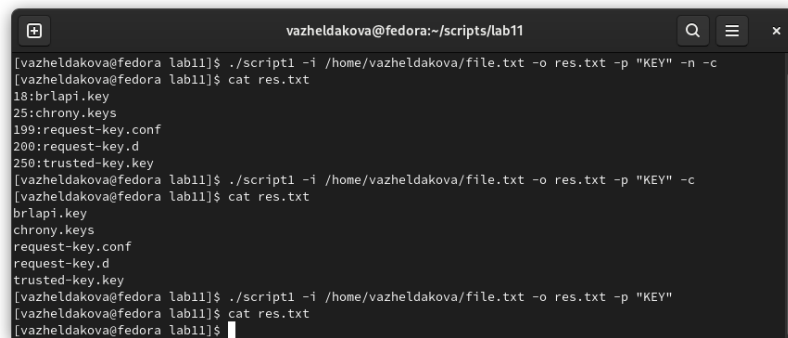
Используя команду `getopts` описали флаги для чтения данных из файла, вывод данных в файл, указания шаблона, включение игнорирования регистра букв и нумерации строк. Затем описали параметры поиска и вывода данных в каждом из возможных случаев, используя `grep` и циклы `if-else` (рис. 3.1 и рис. 3.2).



```
#!/bin/bash
while getopts :i:o:p:cn opt
do
    case "${opt}" in
        i) input=${OPTARG};;
        o) output=${OPTARG};;
        p) pattern=${OPTARG};;
        c) c=true;;
        n) n=true;;
    esac
done
if [ $c ]
then
    if [ $n ]
    then
        grep -n -i $pattern $input > $output
        exit 0
    else
        grep -i $pattern $input > $output
        exit 0
    fi
else
    if [ $n ]
    then
        grep -n $pattern $input > $output
        exit 0
    else
        grep $pattern $input > $output
        exit 0
    fi
fi
```

Рис. 3.1: Первый скрипт



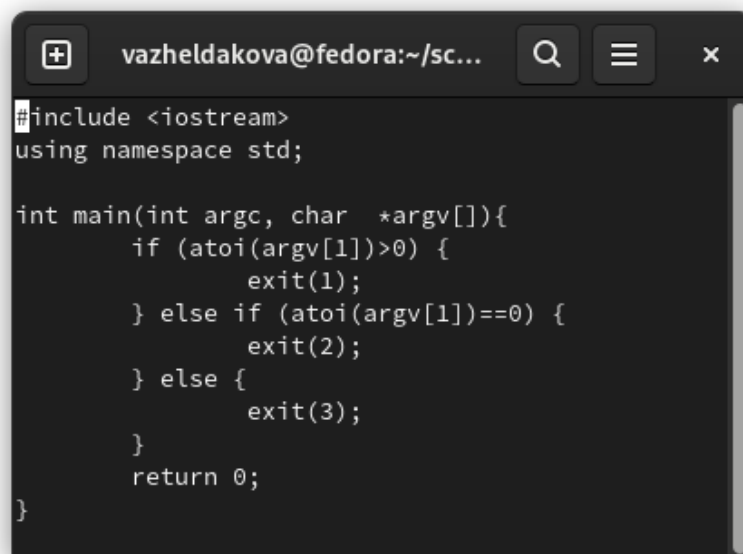


```
vazheldakova@fedora:~/scripts/lab11
[vazheldakova@fedora lab11]$ ./script1 -i /home/vazheldakova/file.txt -o res.txt -p "KEY" -n -c
[vazheldakova@fedora lab11]$ cat res.txt
18:brlapi.key
25:chrony.keys
199:request-key.conf
200:request-key.d
250:trusted-key.key
[vazheldakova@fedora lab11]$ ./script1 -i /home/vazheldakova/file.txt -o res.txt -p "KEY" -c
[vazheldakova@fedora lab11]$ cat res.txt
brlapi.key
chrony.keys
request-key.conf
request-key.d
trusted-key.key
[vazheldakova@fedora lab11]$ ./script1 -i /home/vazheldakova/file.txt -o res.txt -p "KEY"
[vazheldakova@fedora lab11]$ cat res.txt
[vazheldakova@fedora lab11]$
```

Рис. 3.2: Запуск первого скрипта с разными флагами

## 3.2 Второе задание

Создали файл формата сpp, оформили программу принимающую на вход число. С помощью цикла if-else if-else сделали разный вывод для чисел больше нуля, равных нулю и меньше нуля, а именно exit(1), exit(2), exit(3) (рис. 3.3).



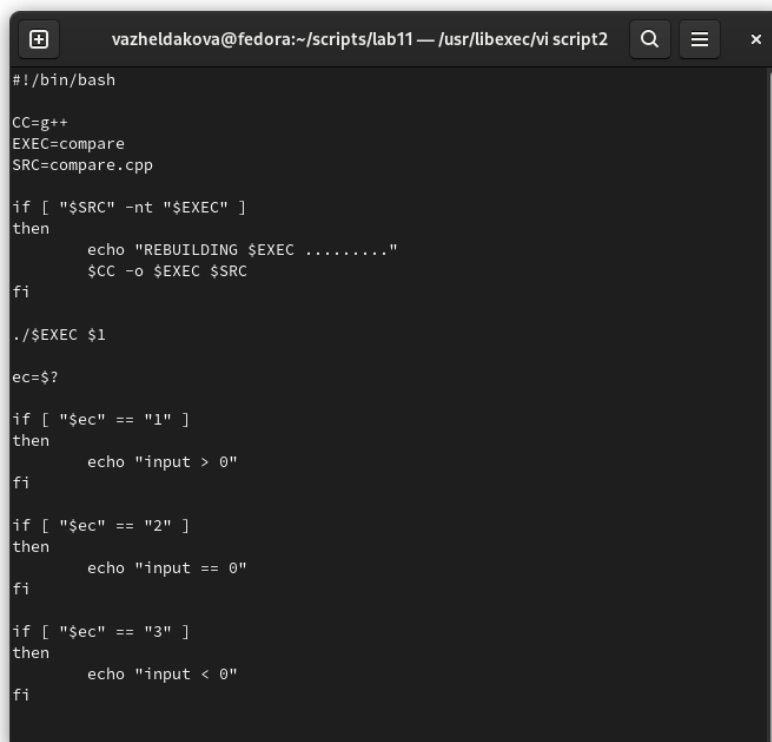
```
vazheldakova@fedora:~/sc...
#include <iostream>
using namespace std;

int main(int argc, char *argv[]){
    if (atoi(argv[1])>0) {
        exit(1);
    } else if (atoi(argv[1])==0) {
        exit(2);
    } else {
        exit(3);
    }
    return 0;
}
```

Рис. 3.3: Программа на языке C++

Перешли к оформлению второго скрипта, задали source файл (ранее созданный) и исполняемый. В зависимости от результата работы программы

оформили разные сообщения вывода (рис. 3.4 и рис. 3.5).



```
#!/bin/bash

CC=g++
EXEC=compare
SRC=compare.cpp

if [ "$SRC" -nt "$EXEC" ]
then
    echo "REBUILDING $EXEC ....."
    $CC -o $EXEC $SRC
fi

./$EXEC $1

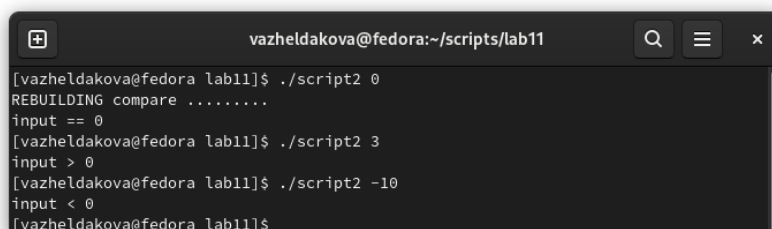
ec=$?

if [ "$ec" == "1" ]
then
    echo "input > 0"
fi

if [ "$ec" == "2" ]
then
    echo "input == 0"
fi

if [ "$ec" == "3" ]
then
    echo "input < 0"
fi
```

Рис. 3.4: Второй скрипт



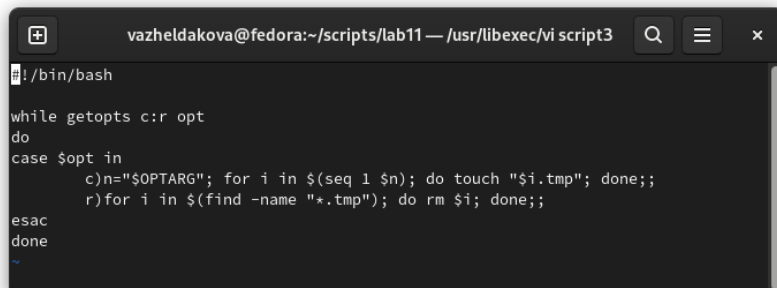
```
[vazheldakova@fedora lab11]$ ./script2 0
REBUILDING compare .....
input == 0
[vazheldakova@fedora lab11]$ ./script2 3
input > 0
[vazheldakova@fedora lab11]$ ./script2 -10
input < 0
[vazheldakova@fedora lab11]$
```

Рис. 3.5: Запуск второго скрипта с разными числами

### 3.3 Третье задание

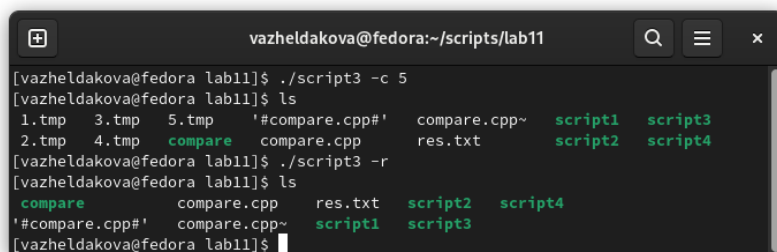
Используя команду `getopts`, описали флаги для создания заданного кол-ва файлов и их удаления. Для создания использовали цикл `for` от 1 до

переданного числа и создали шаблонный файл с каждой итерацией. Для удаления использовали цикл `for` для найденных файлов, название которых соответствовало бы шаблону (рис. 3.6 и рис. 3.7).



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

Рис. 3.6: Третий скрипт



```
[vazheldakova@fedora lab11]$ ./script3 -c 5
[vazheldakova@fedora lab11]$ ls
1.tmp  3.tmp  5.tmp  '#compare.cpp#'  compare.cpp~  script1  script3
2.tmp  4.tmp  compare  compare.cpp  res.txt      script2  script4
[vazheldakova@fedora lab11]$ ./script3 -r
[vazheldakova@fedora lab11]$ ls
compare  compare.cpp  res.txt  script2  script4
'#compare.cpp#'  compare.cpp~  script1  script3
[vazheldakova@fedora lab11]$
```

Рис. 3.7: Запуск третьего скрипта с флагом удаления и создания файлов

### 3.4 Четвёртое задание

Оформили команду `getopts`, которой передаётся название директории. С помощью команды `find` нашли все файлы, которые были изменены не позднее, чем неделю назад и записали их в новый текстовый файл. А затем заархивировали все файлы, перечисленные в нём (рис. 3.8 и рис. 3.8).

A terminal window titled 'vazheldakova@fedora:~/scripts/lab11 — /usr/libexec/vi script4'. The terminal shows the content of a script named 'script4'. The script starts with a shebang line '#!/bin/bash', followed by a while loop that iterates over command-line options. Inside the loop, there is a case statement for the 'd' option that sets a directory variable. After the loop, the script uses 'find' to search for files in the specified directory and then uses 'tar' to create an archive named 'new.tar' from the found files.

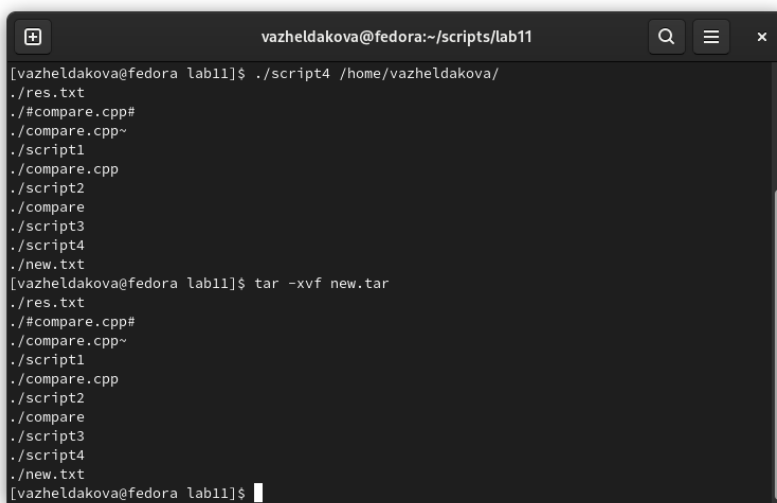
```
#!/bin/bash

while getopts :d: opt; do
case $opt in
    d)dir="$OPTARG";;
esac
done

find $dir -type f -mtime -7 > new.txt

tar -cvf new.tar -T new.txt
```

Рис. 3.8: Четвёртый скрипт

A terminal window titled 'vazheldakova@fedora:~/scripts/lab11'. The terminal shows the execution of 'script4' with the path '/home/vazheldakova/' as an argument. This is followed by a 'tar -xvf new.tar' command to extract the archive. The output of the script is listed, showing files like 'res.txt', 'compare.cpp', and various 'script' files. The terminal ends with a prompt for the user.

```
[vazheldakova@fedora lab11]$ ./script4 /home/vazheldakova/
./res.txt
./#compare.cpp#
./compare.cpp~
./script1
./compare.cpp
./script2
./compare
./script3
./script4
./new.txt
[vazheldakova@fedora lab11]$ tar -xvf new.tar
./res.txt
./#compare.cpp#
./compare.cpp~
./script1
./compare.cpp
./script2
./compare
./script3
./script4
./new.txt
[vazheldakova@fedora lab11]$
```

Рис. 3.9: Запуск четвёртого скрипта и проверка выполнения

## 4 Выводы

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

## 5 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`.

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

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. Какие операторы управления действиями вы знаете?

- Точка с запятой (;) Вы можете разместить две и более команд в одной и той же строке, разделив эти команды с помощью символа точки с запятой ;.
- Амперсанд (&) В том случае, если строка команды оканчивается символом амперсанда `&`, командная оболочка не будет ожидать завершения исполнения этой команды. Сразу же после ввода команды будет выведено новое приглашение командной оболочки, а сама команда будет исполняться в фоновом режиме. В момент завершения исполнения команды в фоновом режиме вы получите соответствующее сообщение.
- Символ доллара со знаком вопроса (\$) Код завершения предыдущей команды сохраняется в переменной командной оболочки с именем `$?`.
- Двойной амперсанд (&&) Командная оболочка будет интерпретировать последовательность символов `&&` как логический оператор “И”. При использовании оператора `&&` вторая команда будет исполняться только в том случае, если исполнение первой команды успешно завершится (будет возвращен нулевой код завершения).
- Двойная вертикальная черта (||) Оператор `||` представляет логическую операцию “ИЛИ”. Вторая команда исполняется только тогда, когда

исполнение первой команды заканчивается неудачей (возвращается ненулевой код завершения).

- Комбинирование операторов `&&` и `||` Вы можете использовать описанные логические операторы “И” и “ИЛИ” для создания структур условных переходов в рамках строк команд.
- Знак фунта (`#`) Все написанное после символа фунта (`#`) игнорируется командной оболочкой. Это обстоятельство оказывается полезным при возникновении необходимости в написании комментариев в сценариях командной оболочки, причем комментарии ни коим образом не будут влиять на процесс исполнения команд или процесс раскрытия команд командной оболочкой.
- Экранирование специальных символов (`\`) Символ обратного слэша позволяет использовать управляющие символы без их интерпретации командной оболочкой; процедура добавления данного символа перед управляющими символами называется экранированием символов.

#### 4. Какие операторы используются для прерывания цикла?

Для управления ходом выполнения цикла служат команды `break` и `continue` [1] и точно соответствуют своим аналогам в других языках программирования. Команда `break` прерывает исполнение цикла, в то время как `continue` передает управление в начало цикла, минуя все последующие команды в теле цикла.

#### 5. Для чего нужны команды `false` и `true`?

Команда `true` всегда возвращает ноль в качестве выходного статуса для индикации успеха.

Команда `false` всегда возвращает не-ноль в качестве выходного статуса для индикации неудачи.

#### 6. Что означает строка `if test -f man$ /i.$s`, встречаемая в командном файле?

Веденная строка означает условие существования файла `man$ /i.$s`



7. Объясните различия между конструкциями while и until.

Разница между циклом while и until – это условие проверки. Пока выполняется условие проверки, цикл while будет продолжать работать. Однако цикл until будет выполняться только в том случае, если условие ложно.