

# **Отчёт по лабораторной работе №11**

*дисциплина: Операционные системы*

Максим Александрович Мишонков

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>5</b>	<b>Выводы</b>	<b>17</b>

## Список иллюстраций

4.1	Создание файла . . . . .	7
4.2	Текст программы . . . . .	7
4.3	Текст программы . . . . .	8
4.4	Проверка работы написанной программы . . . . .	8
4.5	Создание файла . . . . .	8
4.6	Текст программы . . . . .	9
4.7	Текст программы . . . . .	10
4.8	Создание файла . . . . .	10
4.9	Текст программы . . . . .	11
4.10	Проверка работы написанной программы . . . . .	12
4.11	Создание файла . . . . .	12
4.12	Текст программы . . . . .	13

# 1 Цель работы

Целью выполнения данной лабораторной работы является изучение основ программирования в оболочке ОС UNIX, приобретение навыков написания сложных командных файлов с использованием логических управляющих конструкций и циклов.

## 2 Задание

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

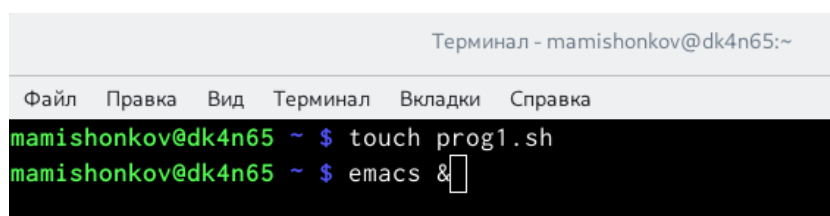
### 3 Теоретическое введение

Команда `getopts` 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`.

## 4 Выполнение лабораторной работы

1. Создал файл для программы 1. (рис. [4.1])

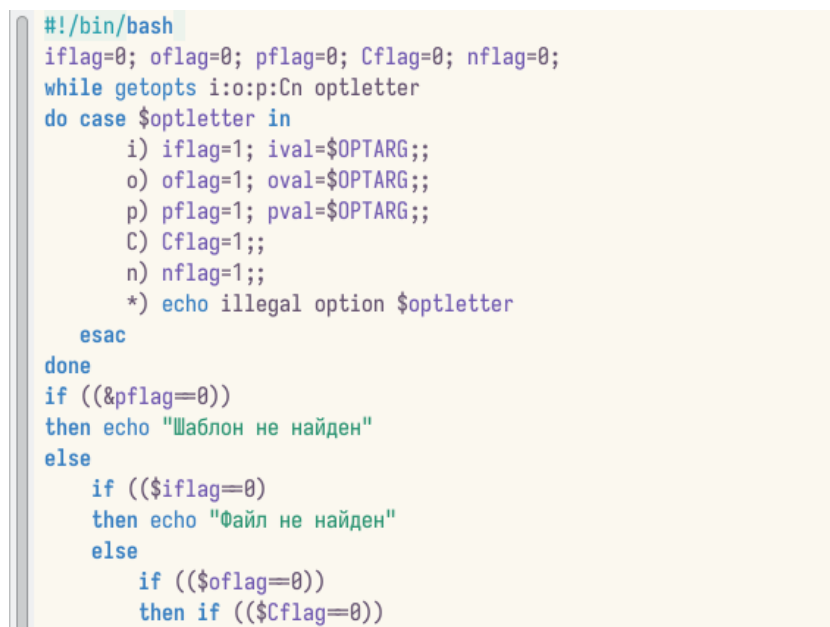


The screenshot shows a terminal window titled "Терминал - mamishonkov@dk4n65:~". The menu bar includes "Файл", "Правка", "Вид", "Терминал", "Вкладки", and "Справка". The command prompt shows the user "mamishonkov@dk4n65" in the directory "~". The user has entered the command "touch prog1.sh" and then "emacs &".

```
Терминал - mamishonkov@dk4n65:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
mamishonkov@dk4n65 ~ $ touch prog1.sh
mamishonkov@dk4n65 ~ $ emacs &
```

Рис. 4.1: Создание файла

2. Написал текст программы 1. (рис. [4.2], [4.3])



The screenshot shows a text editor window with a light yellow background. The code is written in a monospaced font with syntax highlighting. The code is a shell script that uses the getopt command to parse command-line options. It sets flags for -i, -o, -p, -C, and -n, and prints an error message for illegal options. It then checks if the flags are set and prints messages accordingly.

```
#!/bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0;
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 $optletter
    esac
done
if (($pflag==0))
then echo "Шаблон не найден"
else
    if (($iflag==0))
    then echo "Файл не найден"
    else
        if (($oflag==0))
        then if (($Cflag==0))
```

Рис. 4.2: Текст программы

```

        then if (($nflag=0))
        then grep $pval $ival
        else grep -n $pval $ival
        fi
        else if ((nflag=0))
        then grep -i $pval $ival
        else grep -i -n $pval $ival
        fi
        fi
    else if (($Cflag=0))
    then if (($nflag=0))
    then grep $pval $ival > $oval
    else grep -n $pval $ival > $oval
    fi
    else if (($nflag=0))
    then grep -i $pval $ival > $oval
    else grep -i -n $pval $ival > $oval
    fi
    fi
    fi
fi
fi

```

Рис. 4.3: Текст программы

3. Проверил работу написанной программы. (рис. [4.4])

```

mamishonkov@dk4n65 ~ $ chmod u+x prog1.sh
mamishonkov@dk4n65 ~ $ ./prog1.sh -i conf.txt -o output.txt -p h c -n
./prog1.sh: строка 18: синтаксическая ошибка рядом с неожиданным маркером «else»
./prog1.sh: строка 18: `      else      '
mamishonkov@dk4n65 ~ $ ls
backup      file1.doc  format.sh~  prog2.sh   public_html  Изображения
backup.sh   file2.doc  GNUstep     prog2.sh~  tmp          Общедоступные
backup.sh~  file.pdf  lab07.sh~  progl.s.sh  work        'Рабочий стол'
bin         file.txt  prog1.sh   progl.s.sh~  Документы   Шаблоны
conf.txt    format.sh  prog1.sh~  public      Загрузки
mamishonkov@dk4n65 ~ $

```

Рис. 4.4: Проверка работы написанной программы

4. Создал файлы для программы 2. (рис. [4.5])

```

mamishonkov@dk4n65 ~ $ touch chslo.c
mamishonkov@dk4n65 ~ $ touch chslo.sh
mamishonkov@dk4n65 ~ $ emacs &

```

Рис. 4.5: Создание файла



5. Написал текст программы 2. (рис. [4.6], [4.7])

```
#!/bin/bash

gcc chslo.c -o chslo

./chslo

code=$?

case $code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0";;
    )
esac
```

Рис. 4.6: Текст программы

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    printf ("Введите число\n");
    int a;
    scanf ("%d", &a);
    if (a<0) exit(0);
    if (a>0) exit(1);
    if (a==0) exit(2);
    return 0;
}
```

Рис. 4.7: Текст программы

6. Создал файлы для программы 3. (рис. [4.8])

```
mamishonkov@dk4n65 ~ $ touch files.sh
mamishonkov@dk4n65 ~ $ emacs &
```

Рис. 4.8: Создание файла

7. Написал текст программы 3. (рис. [4.9])

```
#!/bin/bash
opt=$1;
format=$2;
number=$3;
function Files()
{
    for (( i=1; i≤$number; i++ )) do
        file=$(echo $format | tr '#' "$i")
        if [ $opt = "-r" ]
        then
            rm -f $file
        elif [ $opt = "-c" ]
        then
            touch $file
        fi
    done
}
Files
```

Рис. 4.9: Текст программы

8. Проверил работу написанной программы. (рис. [4.10])

```

mamishonkov@dk4n65 ~ $ chmod +x files.sh
[3] Завершён emacs
mamishonkov@dk4n65 ~ $ ls
backup      chslo.sh  file.txt  prog2.sh~  Загрузки
backup.sh   chslo.sh~ format.sh  progl.s.sh  Изображения
backup.sh~  conf.txt  format.sh~ progl.s.sh~ Общедоступные
bin         file1.doc GNUstep   public      'Рабочий стол'
'#chslo.c#' file2.doc lab07.sh~ public_html Шаблоны
chslo.c     file.pdf  prog1.sh  tmp
chslo.c~    files.sh  prog1.sh~ work
'#chslo.sh#' files.sh~ prog2.sh  Документы
mamishonkov@dk4n65 ~ $ ./files.sh -c abc#.txt 3
mamishonkov@dk4n65 ~ $ ls
abc1.txt    chslo.c   file.pdf  prog1.sh  tmp
abc2.txt    chslo.c~ files.sh   prog1.sh~ work
abc3.txt    '#chslo.sh#' files.sh~ prog2.sh  Документы
backup      chslo.sh  file.txt  prog2.sh~ Загрузки
backup.sh   chslo.sh~ format.sh  progl.s.sh  Изображения
backup.sh~  conf.txt  format.sh~ progl.s.sh~ Общедоступные
bin         file1.doc GNUstep   public      'Рабочий стол'
'#chslo.c#' file2.doc lab07.sh~ public_html Шаблоны
mamishonkov@dk4n65 ~ $ ./files.sh -r abc#.txt 3
mamishonkov@dk4n65 ~ $ ls
backup      chslo.sh  file.txt  prog2.sh~  Загрузки
backup.sh   chslo.sh~ format.sh  progl.s.sh  Изображения
backup.sh~  conf.txt  format.sh~ progl.s.sh~ Общедоступные
bin         file1.doc GNUstep   public      'Рабочий стол'
'#chslo.c#' file2.doc lab07.sh~ public_html Шаблоны
chslo.c     file.pdf  prog1.sh  tmp
chslo.c~    files.sh  prog1.sh~ work
'#chslo.sh#' files.sh~ prog2.sh  Документы
mamishonkov@dk4n65 ~ $

```

Рис. 4.10: Проверка работы написанной программы

9. Создал файлы для программы 4. (рис. [4.11])

```

mamishonkov@dk4n65 ~ $ touch prog4.sh
mamishonkov@dk4n65 ~ $ emacs &

```

Рис. 4.11: Создание файла

10. Написал текст программы 4. (рис. [4.12])

```
#!/bin/bash
files=$(find ./ -maxdepth 1 -mtime -7)
listing=""
for file in "$files" ; do
    file=$(echo "$file" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

Рис. 4.12: Текст программы

### Контрольные вопросы:

1. Каково предназначение команды getopt?

Команда getopt 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`.

2. Какое отношение метасимволы имеют к генерации имён файлов?

При перечислении имен файлов текущего каталога можно использовать следующие символы:

? – соответствует любому одному символу;

[c1-c1] – соответствует любому символу, лексикографически находящемуся между сим

echo \* – выведет имена всех файлов текущего каталога, что представляет собой прос

ls .c – выведет все файлы с последними двумя символами, равными .c;

echo prog.? – выдаст все файлы, состоящие из пяти или шести символов, первыми пят

[a-z] – соответствует произвольному имени файла в текущем каталоге, начинающемуся

### 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

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 mans/i.$$`, встречаемая в командном файле?

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

7. Объясните различия между конструкциями `while` и `until`.

Разница между циклом `while` (пока) и `until` (пока не) – это условие проверки. Пока ВЫПОЛНЯЕТСЯ условие проверки, цикл `while` будет продолжать работать. Однако цикл `until` будет выполняться только пока условие ЛОЖНО.



## 5 Выводы

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