

Отчет по лабораторной работе №12

Дисциплина: Операционные системы

Пермякова Елизавета Евгеньевна

Содержание

1	Цель работы	4
2	Задачи	5
3	Выполнение лабораторной работы	6
4	Контрольные вопросы	15
5	Выводы	19
6	Библиография	20

List of Figures

3.1	Первый скрипт	7
3.2	Проверка работы скрипта	8
3.3	Файл prg2.c	9
3.4	Файл prg2.sh	9
3.5	Проверка работы скриптов	10
3.6	Третий скрипт	11
3.7	Проверка работы скрипта	12
3.8	Четвертый скрипт	13
3.9	Проверка работы скрипта	14

1 Цель работы

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

2 Задачи

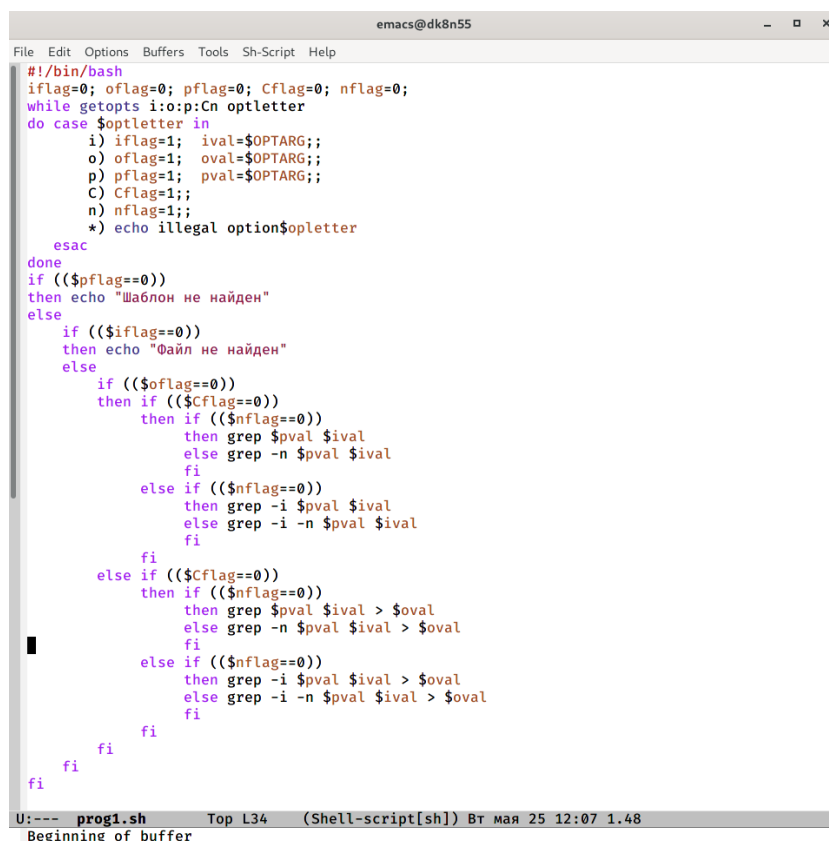
1. Познакомиться с логическими управляющими конструкций и циклов.
2. В ходе работы написать 4 командных файла.
3. Выполнить отчет.

3 Выполнение лабораторной работы

1) Используя команды `getopts` и `grep`, написала командный файл, который анализирует командную строку с ключами:

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-р` — 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`.

Для данной задачи я создала файл `prog1.sh` и написала соответствующие скрипты. (рис. 3.1)



```
emacs@dk8n55
File Edit Options Buffers Tools Sh-Script Help
#!/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 (($Cflag==0))
        then if (($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
fi
fi
fi
U:--- prog1.sh Top L34 (Shell-script[sh]) Вт мая 25 12:07 1.48
Beginning of buffer
```

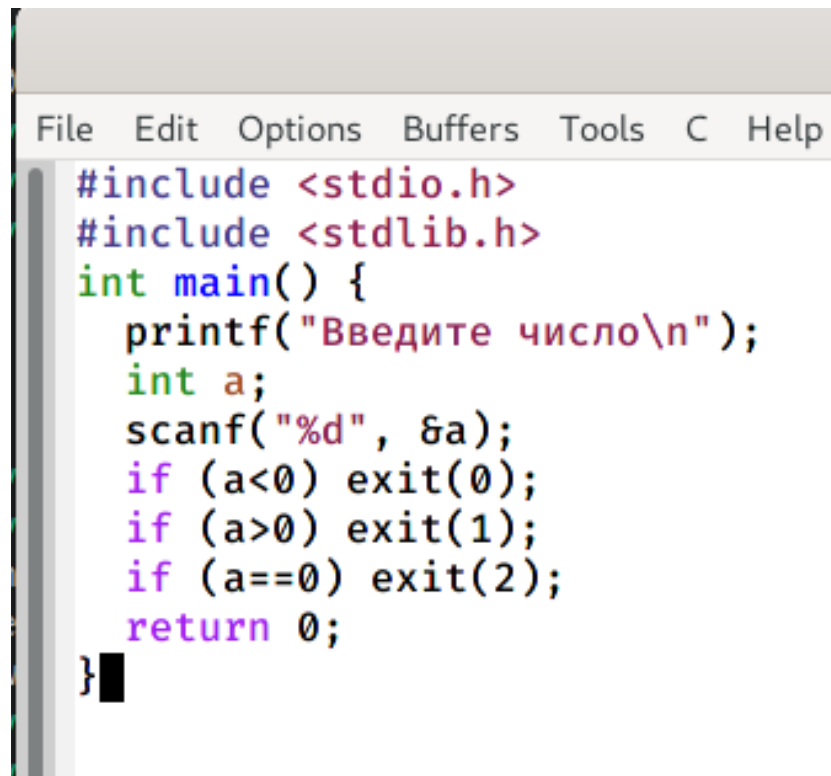
Figure 3.1: Первый скрипт

Далее я проверила работу написанного скрипта, используя различные опции (например, команда «./prog.sh -i a1.txt -o a2.txt -p capital -C -n»), предварительно добавив право на исполнение файла (команда «chmod +x prog1.sh») и создав 2 файла, которые необходимы для выполнения программы: a1.txt и a2.txt. Скрипт работает корректно. (рис. 3.2):

```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
eeperryakova@dk8n55 ~ $ touch prog1.sh
eeperryakova@dk8n55 ~ $ emacs &
[1] 18088
eeperryakova@dk8n55 ~ $ touch a1.txt a2.txt
eeperryakova@dk8n55 ~ $ chmod +x prog1.sh
eeperryakova@dk8n55 ~ $ cat a1.txt
Vienna is the capital of Austria
Sydney is not the Capital of Australia
Ottawa is the CAPITAL of Canada
etc
eeperryakova@dk8n55 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p capital -C -n
eeperryakova@dk8n55 ~ $ cat a2.txt
1:Vienna is the capital of Austria
2:Sydney is not the Capital of Australia
3:Ottawa is the CAPITAL of Canada
eeperryakova@dk8n55 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p capital -n
eeperryakova@dk8n55 ~ $ cat a2.txt
1:Vienna is the capital of Austria
eeperryakova@dk8n55 ~ $ ./prog1.sh -i a1.txt -C -n
Шаблон не найден
eeperryakova@dk8n55 ~ $ ./prog1.sh -o a2.txt -p capital -C -n
Файл не найден
eeperryakova@dk8n55 ~ $
```

Figure 3.2: Проверка работы скрипта

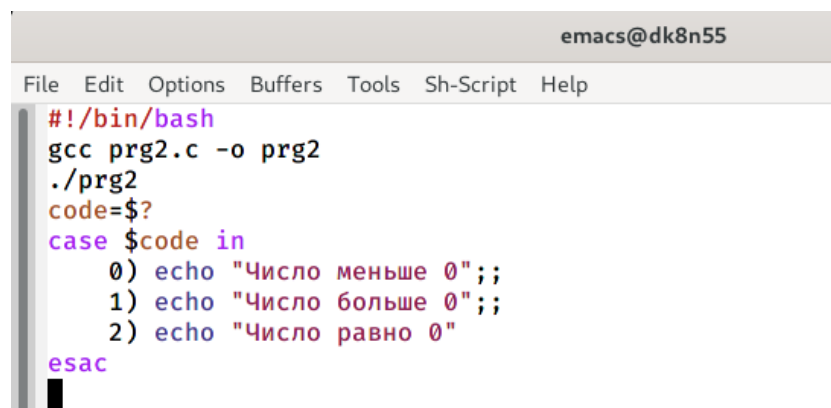
- 2) Написана на языке Си программа, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено. Для данной задачи я создала 2 файла: `prg2.c` и `prg2.sh` и написала соответствующие скрипты. (рис. 3.3) (рис. 3.4)



```
File Edit Options Buffers Tools C Help

#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;
}
```

Figure 3.3: Файл prg2.c



```
emacs@dk8n55
File Edit Options Buffers Tools Sh-Script Help

#!/bin/bash
gcc prg2.c -o prg2
./prg2
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0"
esac
```

Figure 3.4: Файл prg2.sh

Далее я проверила работу написанных скриптов (команда «./prg2.sh»), предварительно добавив право на исполнение файла (команда «chmod +x prg2.sh»). Скрипты работают корректно. (рис. 3.5)

```

eeperryakova@dk8n55 ~ $ touch prg2.c
eeperryakova@dk8n55 ~ $ emacs &
[2] 24876
eeperryakova@dk8n55 ~ $ touch prg2.sh
eeperryakova@dk8n55 ~ $ emacs &
[3] 25846
eeperryakova@dk8n55 ~ $ chmod +x prg2.sh
eeperryakova@dk8n55 ~ $ ./prg2.sh
Введите число
8
Число больше 0
eeperryakova@dk8n55 ~ $ ./prg2.sh
Введите число
0
Число равно 0
eeperryakova@dk8n55 ~ $ ./prg2.sh
Введите число
-3
Число меньше 0
eeperryakova@dk8n55 ~ $ █

```

Figure 3.5: Проверка работы скриптов

- 3) Написала 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). Для данной задачи я создала файл prog3.sh и написала соответствующий скрипт. (рис. 3.6)

The image shows a screenshot of an Emacs editor window. The title bar at the top reads 'emacs@dk8n55'. Below the title bar is a menu bar with the following items: 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The main editing area contains a shell script. The script starts with a shebang line '#!/bin/bash' in red. It then sets three variables: 'opt=\$1;', 'format=\$2;', and 'number=\$3;'. A function named 'Files()' is defined in purple. Inside this function, there is a 'for' loop in purple that iterates from 'i=1' to 'i=\$number'. Inside the loop, a file path is constructed using 'file=\$(echo \$format | tr '#' '\$i')'. Then, an 'if' statement checks if the option 'opt' is equal to '-r'. If true, it runs 'rm -f \$file'. If 'opt' is equal to '-c', it runs 'touch \$file'. The function ends with 'done' and '}'. The cursor is at the end of the line 'Files'.

Figure 3.6: Третий скрипт

Далее я проверила работу написанного скрипта (команда «./prog3.sh»), предварительно добавив право на исполнение файла (команда «chmod +x prog3.sh»). Сначала я создала три файла (команда «./prog3.sh -c b#.txt 3»), удовлетворяющие условию задачи, а потом удалила их (команда «./prog3.sh -r b#.txt 3»). (рис. 3.7)

```

eepermyakova@dk8n55 ~ $ chmod +x prog3.sh
eepermyakova@dk8n55 ~ $ ls
'2021-05-25 09-05-57.mkv' backup file.txt prg2 prog3.sh~ Видео
'2021-05-25 11-09-21.mkv' backup.sh format.sh prg2.c program Документы
'2021-05-25 11-34-45.mkv' backup.sh~ format.sh~ prg2.c~ program.asm Загрузки
a1.txt conf.txt GNUstep prg2.sh program.lst Изображения
a2.txt example1.txt lab07.sh prg2.sh~ public Музыка
abcd example2.txt lab07.sh~ prls.sh public_html Общедоступные
abcl example3.txt laboratory prls.sh~ reports 'Рабочий стол'
asfdg example4.txt lessfun prog1.sh ski.plases Шаблоны
asfdg.as~ feathers may prog1.sh~ special
asfdg file1.doc monthly prog2.sh text.txt
asfdg.as~ file2.doc my_os prog2.sh~ tmp
australia file.pdf play prog2.sh~ work

eepermyakova@dk8n55 ~ $ ./prog3.sh -c b#.txt 3
eepermyakova@dk8n55 ~ $ ls
'2021-05-25 09-05-57.mkv' b1.txt file1.doc monthly prog2.sh text.txt
'2021-05-25 11-09-21.mkv' b2.txt file2.doc my_os prog2.sh~ tmp
'2021-05-25 11-34-45.mkv' b3.txt file.pdf play prog3.sh work
a1.txt backup file.txt prg2 prog3.sh~ Видео
a2.txt backup.sh format.sh prg2.c program Документы
abcd backup.sh~ format.sh~ prg2.c~ program.asm Загрузки
abcl conf.txt GNUstep prg2.sh program.lst Изображения
asfdg example1.txt lab07.sh prg2.sh~ public Музыка
asfdg.as~ example2.txt lab07.sh~ prls.sh public_html Общедоступные
asfdg example3.txt laboratory prls.sh~ reports 'Рабочий стол'
asfdg.as~ example4.txt lessfun prog1.sh ski.plases Шаблоны
australia feathers may prog1.sh~ special

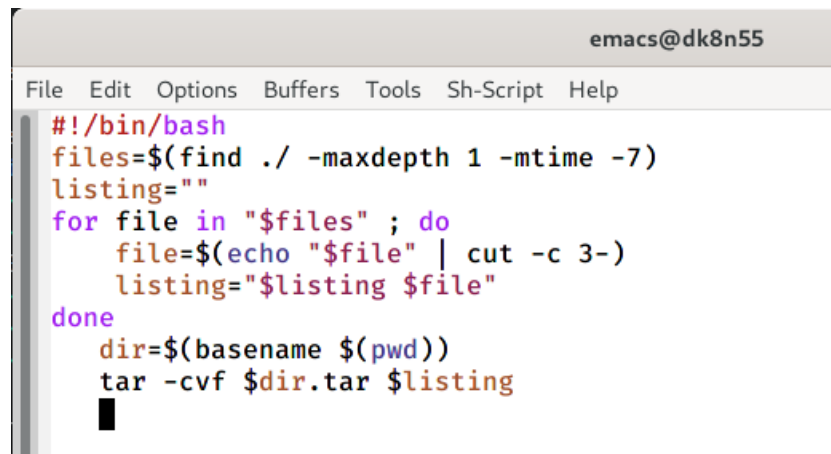
eepermyakova@dk8n55 ~ $ ./prog3.sh -r b#.txt 3
eepermyakova@dk8n55 ~ $ ls
'2021-05-25 09-05-57.mkv' backup file.txt prg2 prog3.sh~ Видео
'2021-05-25 11-09-21.mkv' backup.sh format.sh prg2.c program Документы
'2021-05-25 11-34-45.mkv' backup.sh~ format.sh~ prg2.c~ program.asm Загрузки
a1.txt conf.txt GNUstep prg2.sh program.lst Изображения
a2.txt example1.txt lab07.sh prg2.sh~ public Музыка
abcd example2.txt lab07.sh~ prls.sh public_html Общедоступные
abcl example3.txt laboratory prls.sh~ reports 'Рабочий стол'
asfdg example4.txt lessfun prog1.sh ski.plases Шаблоны
asfdg.as~ feathers may prog1.sh~ special
asfdg file1.doc monthly prog2.sh text.txt
asfdg.as~ file2.doc my_os prog2.sh~ tmp
australia file.pdf play prog3.sh work

eepermyakova@dk8n55 ~ $

```

Figure 3.7: Проверка работы скрипта

- 4) Написала командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). Для данной задачи я создала файл prog4.sh и написала соответствующий скрипт. (рис. 3.8)



```
emacs@dk8n55
File Edit Options Buffers Tools Sh-Script Help
#!/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
```

Figure 3.8: Четвертый скрипт

Далее я проверила работу написанного скрипта (команды «./prog4.sh» и «tar -tf catalog.tar»), предварительно добавив право на исполнение файла (команда «chmod +x prog4.sh») и создав отдельный catalog с несколькими файлами. Скрипт работает корректно. (рис. 3.9)

```

eepermyakova@dk8n55 ~ $ touch prog4.sh
eepermyakova@dk8n55 ~ $ emacs &
[5] 29693
eepermyakova@dk8n55 ~ $ chmod +x prog4.sh
eepermyakova@dk8n55 ~ $ cd ~/catalog
eepermyakova@dk8n55 ~/catalog $ ls -l
итого 9
-rw-r--r-- 1 eepermyakova studsci 108 мая 25 11:59 a1.txt
-rw-r--r-- 1 eepermyakova studsci 35 мая 25 12:04 a2.txt
drwxr-xr-x 2 eepermyakova studsci 2048 мая 25 09:19 backup
-rw-r--r-- 1 eepermyakova studsci 196 мая 25 12:13 prg2.c
-rwxr-xr-x 1 eepermyakova studsci 187 мая 25 12:16 prg2.sh
-rwxr-xr-x 1 eepermyakova studsci 958 мая 25 11:50 prog1.sh
-rwxr-xr-x 1 eepermyakova studsci 246 мая 25 12:26 prog3.sh
-rwxr-xr-x 1 eepermyakova studsci 223 мая 25 12:37 prog4.sh
eepermyakova@dk8n55 ~/catalog $ ./prog4.sh
a1.txt
a2.txt
backup/
backup/backup.sh.bz2
prg2.c
prg2.sh
prog1.sh
prog3.sh
prog4.sh
eepermyakova@dk8n55 ~/catalog $ tar -tf catalog.tar
a1.txt
a2.txt
backup/
backup/backup.sh.bz2
prg2.c
prg2.sh
prog1.sh
prog3.sh
prog4.sh
eepermyakova@dk8n55 ~/catalog $ 

```

Figure 3.9: Проверка работы скрипта

4 Контрольные вопросы

- 1) Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, используется для объявления переменных. Синтаксис команды следующий:

```
getopts option-string variable [arg...]
```

Флаги – это опции командной строки, обычно помеченные знаком минус; Например, для команды `ls` флагом может являться `-F`.

Строка опций `option-string` – это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введённые данные с помощью оператора `case`.

Функция `getopts` включает две специальные переменные среды `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента.

Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных.

- 2) При перечислении имён файлов текущего каталога можно использовать следующие символы:

1. *–соответствует произвольной, в том числе и пустой строке;
2. ?–соответствует любому одинарному символу;
3. [c1-c2] – соответствует любому символу, лексикографически находящемуся между символами c1 и c2.

Например,

1.1. `echo*` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`;

1.2. `ls*.c`–выведет все файлы с последними двумя символами, совпадающими с.с.

1.3. `echoproг.?`–выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `прог..`

1.4. `[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 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) 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) 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, неравный нулю (т.е. ложь). Примеры бесконечных циклов:

```
while true
do echo hello andy

done
until false

do echo hello mike

done
```

6) 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) 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.

5 Выводы

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

6 Библиография

1. https://esystem.rudn.ru/pluginfile.php/1142093/mod_resource/content/3/009-lab_shell_prog_2.pdf
2. Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: <http://lib.rudn.ru/MegaPro2/Download/MObject/6118>.
3. Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)
4. Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ЕТ 50)