

Отчёт по лабораторной работе №11

Операционные системы

Самсонова Мария Ильинична

Содержание

Цель работы	1
Задание	1
Выполнение лабораторной работы	2
Вывод.....	11
Ответы на контрольные вопросы	11

Цель работы

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

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;
 - `-ршаблон` — 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 ∞ (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).

Выполнение лабораторной работы

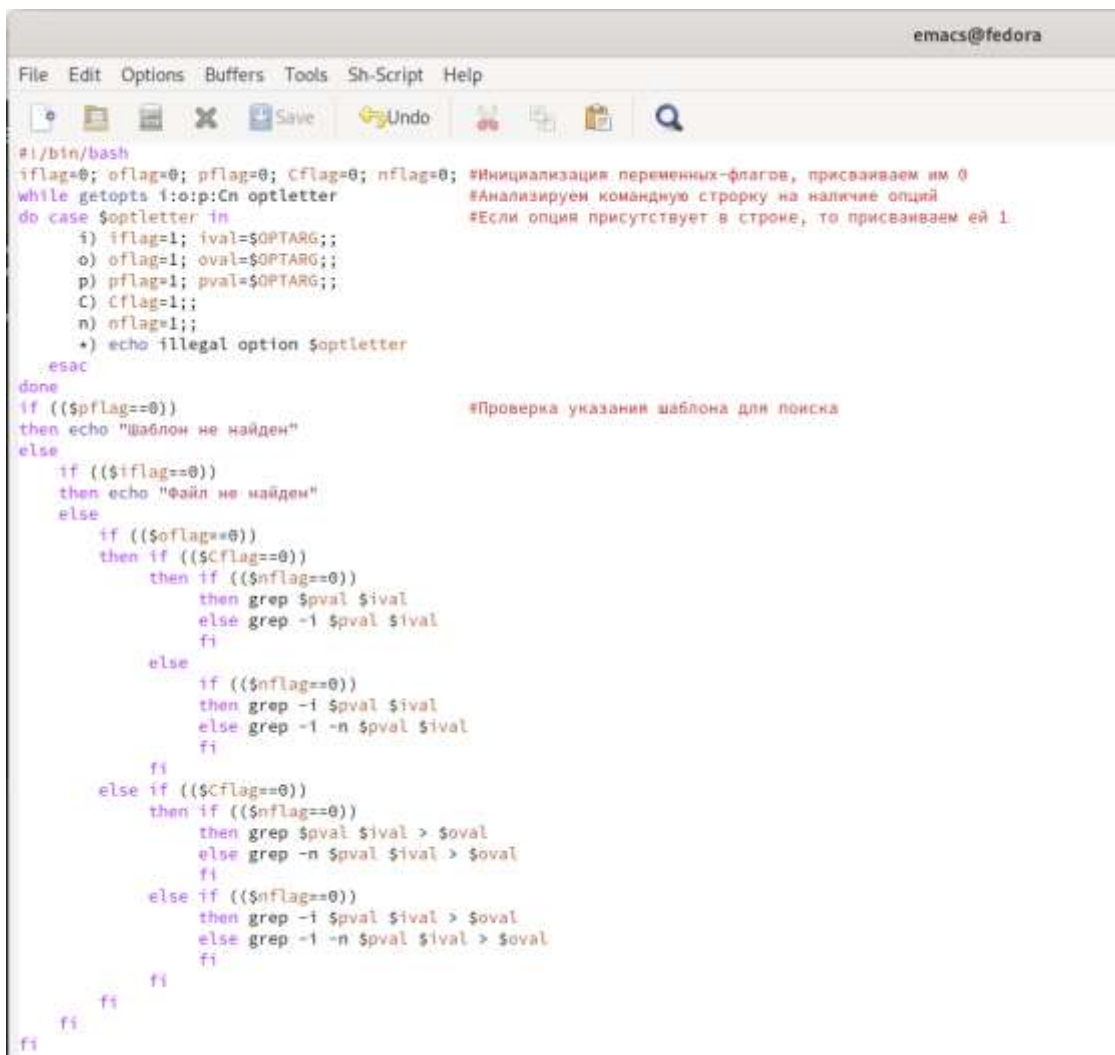
1. Используя команды getopts grep, для начала мы написали командный файл, который анализирует командную строку с ключами:
 - -iinputfile — прочитать данные из указанного файла;
 - -ooutputfile — вывести данные в указанный файл;
 - -p шаблон — указать шаблон для поиска;
 - -C — различать большие и малые буквы;
 - -n — выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом -p.
 - Для данной задачи мы создали файл prog1.sh (рис. -@fig:001) и написали соответствующие скрипты. (рис. -@fig:002).



```
misamsonova@fedora:~  
[misamsonova@fedora ~]$ touch prog1.sh  
[misamsonova@fedora ~]$ emacs &
```

Создание файла

{ #fig:001 width=70% }



```
emacs@fedora
File Edit Options Buffers Tools Sh-Script Help
#i/bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0; #Инициализация переменных-флагов, присваиваем им 0
while getopts i:op:Cn optletter #Анализируем командную строку на наличие опций
do case $optletter in #Если опция присутствует в строке, то присваиваем ей 1
i) iflag=1; ival=$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)) #Проверка указания шаблона для поиска
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 -i $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
```

Скринш №1

{ #fig:002 width=70% }

- Далее проверили работу написанного скрипта, используя различные опции (например, команда «./prog1.sh -i a1.txt -o a2.txt -p -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 (рис. - @fig:003 , -@fig:004). Скрипт работает корректно.



```
misamsonova@fedora:~
[misamsonova@fedora ~]$ touch a1.txt a2.txt
[misamsonova@fedora ~]$ chmod +x prog1.sh
[misamsonova@fedora ~]$
```

Предоставление прав доступа

{ #fig:003 width=70% }

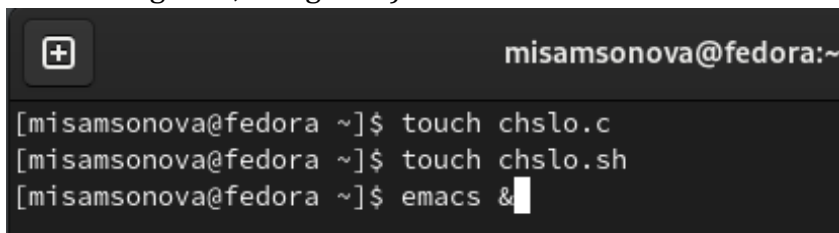


```
[misamsonova@fedora ~]$ cat a1.txt
water abc abcs
asd
progl
water water
[misamsonova@fedora ~]$ ./progl.sh -i a1.txt -o a2.txt -p water -n
[misamsonova@fedora ~]$ cat a2.txt
1:water abc abcs
4:water water
[misamsonova@fedora ~]$ ./progl.sh -i a1.txt -o a2.txt -p water -C -n
[misamsonova@fedora ~]$ cat a2.txt
1:water abc abcs
4:water water
```

Проверка работы программы

{ #fig:004 width=70% }

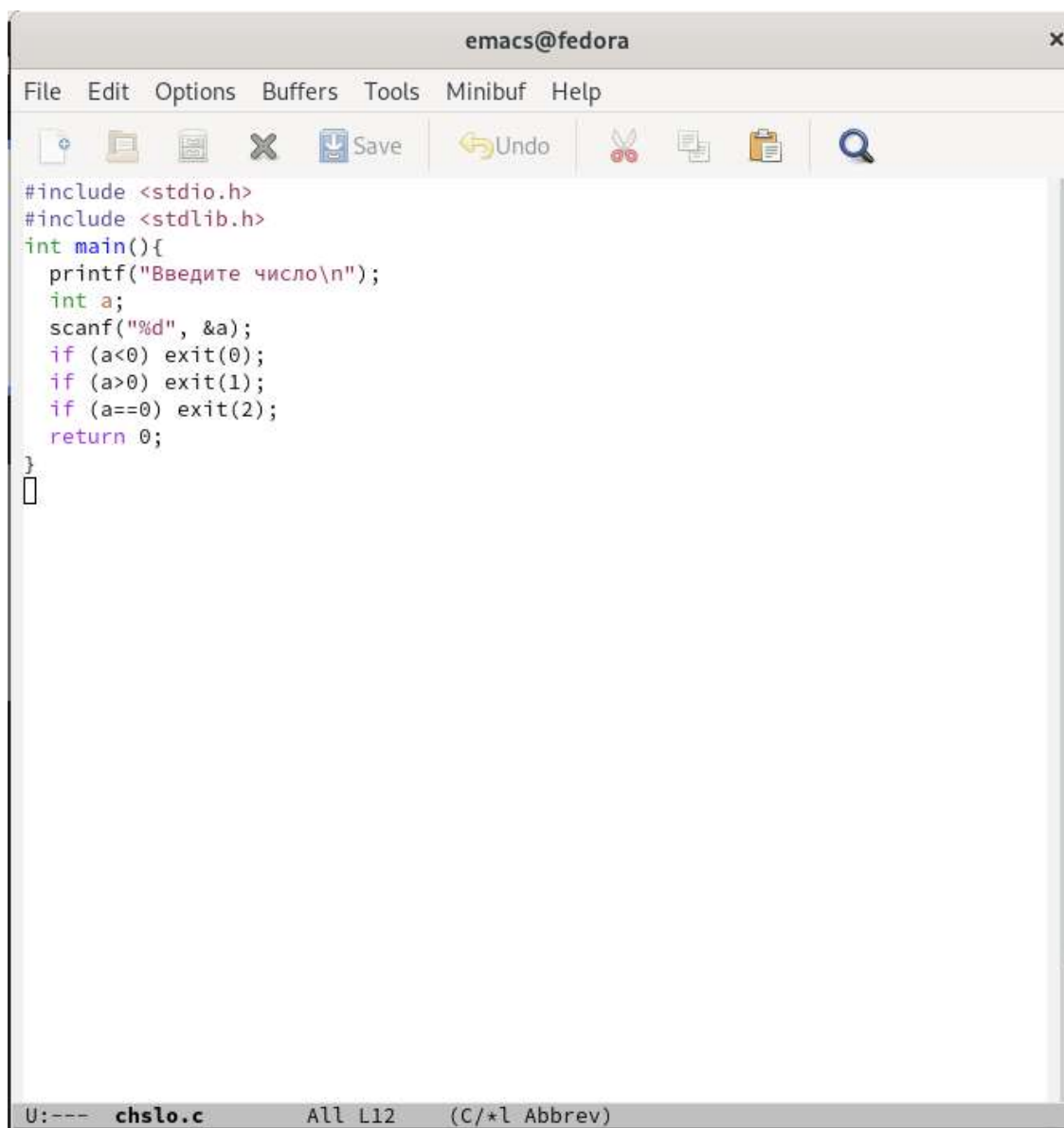
2. Теперь написали на языке C программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю, затем завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено. Для данной задачи мы создали 2 файла: `chslo.c` и `chslo.sh` (рис. -@fig:005) и написали соответствующие скрипты. (команды «`touch prog2.sh`» и «`emacs &`») (рис. -@fig:006 , -@fig:007).



```
[misamsonova@fedora ~]$ touch chslo.c
[misamsonova@fedora ~]$ touch chslo.sh
[misamsonova@fedora ~]$ emacs &
```

Создание файлов

{ #fig:005 width=70% }



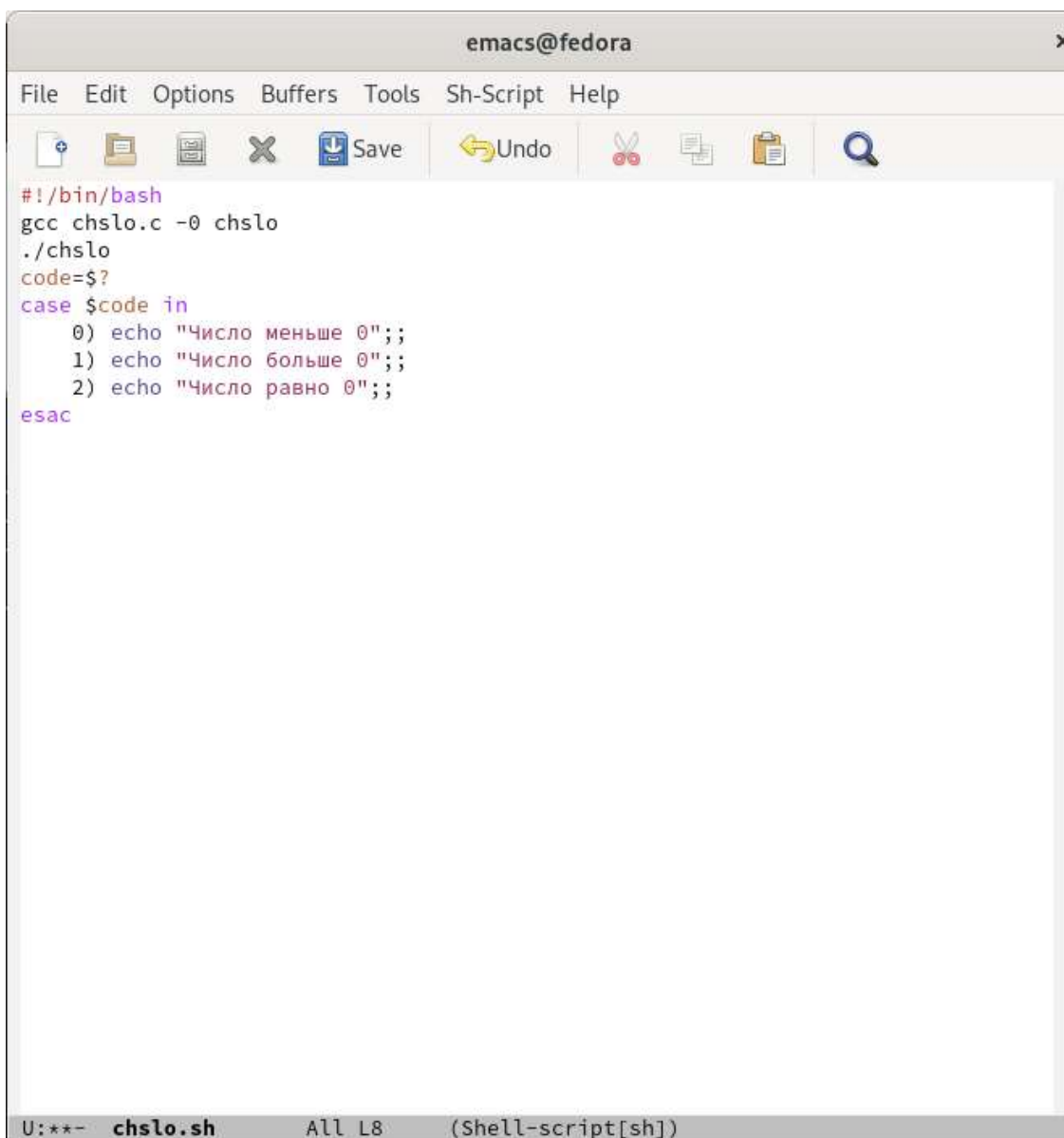
The image shows a screenshot of the Emacs editor window titled "emacs@fedora". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Minibuf", and "Help". The toolbar contains icons for opening a file, saving, undo, redo, cut, copy, paste, and search. The main text area displays the following C code:

```
#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;
}
```

The status bar at the bottom shows "U:--- chslo.c", "All L12", and "(C/*l Abbrev)".

Работа в файле chslo.c

{ #fig:006 width=70% }



```
#!/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
```

Работа в файле *chslo.sh*

{ #fig:007 width=70% }

- Проверили работу написанных скриптов (команда «**./chslo.sh**»), предварительно добавив право на исполнение файла (команда «**chmod +x chslo.sh**») (рис. - @fig:008). Скрипты работают корректно.

```
misamsonova@fedora:~  
[misamsonova@fedora ~]$ chmod +x chslo.sh  
[misamsonova@fedora ~]$ ./chslo.sh  
Введите число  
0  
Число равно 0  
[misamsonova@fedora ~]$ ./chslo.sh  
Введите число  
6  
Число больше 0  
[misamsonova@fedora ~]$ ./chslo.sh  
Введите число  
-1  
Число меньше 0  
[misamsonova@fedora ~]$
```

Проверка скрипта №2

{ #fig:008 width=70% }

3. После чего написали 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 мы создали файл: files.sh (рис. -@fig:009) и написали соответствующий скрипт (рис. -@fig:010).

```
misamsonova@fedora:~  
[misamsonova@fedora ~]$ touch files.sh  
[misamsonova@fedora ~]$ emacs &
```

Создание файлов

{ #fig:009 width=70% }

```
emacs@fedora
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/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
```

U:--- files.sh All L11 (Shell-script[sh])
Wrote /home/misamsonova/files.sh

Скpunm №3

{ #fig:010 width=70% }

- Далее мы проверили работу написанного скрипта (команда «./files.sh»), предварительно добавив право на исполнение файла (команда «chmod +x files.sh»). Сначала мы создали три файла (команда «./files.sh -c abc#.txt 3»), удовлетворяющие условию задачи, а потом удалили их (команда «./files.sh -r abc#.txt 3») (рис. -@fig:011).


```

[~] misamsonova@fedora:~$ chmod +x files.sh
[~] misamsonova@fedora:~$ ls
a1.txt      example1.txt      file.pdf      progr1.sh-
a2.txt      example1.txt-     files.sh      progrls.sh-
autozilla  'example2.txt#'  files.sh-    progrls.sh-
backup     example2.txt      file.txt      reports
backup.sh  example2.txt-    format.sh    skt.planes-
backup.sh- 'example3.txt#'  format.sh-   text.txt
etc        'example3.txt#-' 'lab07.sh#-' work
blog       example3.txt      lab07.sh     work
chslc      example3.txt-    lab08.sh     work
chslc.c    'example4.txt#'  may          work
chslc.c-   example4.txt     monthlt      work
chslc.sh   example4.txt-    monthlt      work
chslc.sh-  feathers         my_ss        work
conf.txt   file2.docx       progr1.sh    work
'example1.txt#-' file.docx      progr1.sh-
[~] misamsonova@fedora:~$ ./files.sh -c abc#.txt 3
[~] misamsonova@fedora:~$ ls
a1.txt      conf.txt      file.docx      progr1.sh-
a2.txt      'example1.txt#-' file.pdf      progrls.sh-
abc1.txt    example1.txt  files.sh      progrls.sh-
abc2.txt    example1.txt- files.sh-    reports
abc3.txt    'example2.txt#' file.txt      skt.planes-
autozilla  example2.txt  format.sh     text.txt
backup     example2.txt- format.sh-   work
backup.sh  'example3.txt#' 'lab07.sh#-' work
backup.sh- 'example3.txt#-' lab07.sh     work
etc        example3.txt  lab08.sh     work
blog       example3.txt- may          work
chslc      'example4.txt#' monthlt      work
chslc.c    example4.txt  monthlt      work
chslc.c-   example4.txt- monthlt      work
chslc.sh   feathers      my_ss        work
chslc.sh-  file2.docx   progr1.sh    work
[~] misamsonova@fedora:~$ ./files.sh -r abc#.txt 3
[~] misamsonova@fedora:~$ ls
a1.txt      example1.txt      file.pdf      progr1.sh-
a2.txt      example1.txt-     files.sh      progrls.sh-
autozilla  'example2.txt#'  files.sh-    progrls.sh-
backup     example2.txt      file.txt      reports
backup.sh  example2.txt-    format.sh    skt.planes-
backup.sh- 'example3.txt#'  format.sh-   text.txt
etc        'example3.txt#-' 'lab07.sh#-' work
blog       example3.txt      lab07.sh     work
chslc      example3.txt-    lab08.sh     work
chslc.c    'example4.txt#'  may          work
chslc.c-   example4.txt     monthlt      work
chslc.sh   example4.txt-    monthlt      work
chslc.sh-  feathers         my_ss        work
chslc.sh-  file2.docx       progr1.sh    work

```

Проверка работы скрипта №3

{ #fig:011 width=70% }

4. Наконец, написали командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировали его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). Для данной задачи мы создали файл: prog4.sh (рис. -@fig:012) и написали соответствующий скрипт (рис. -@fig:013).

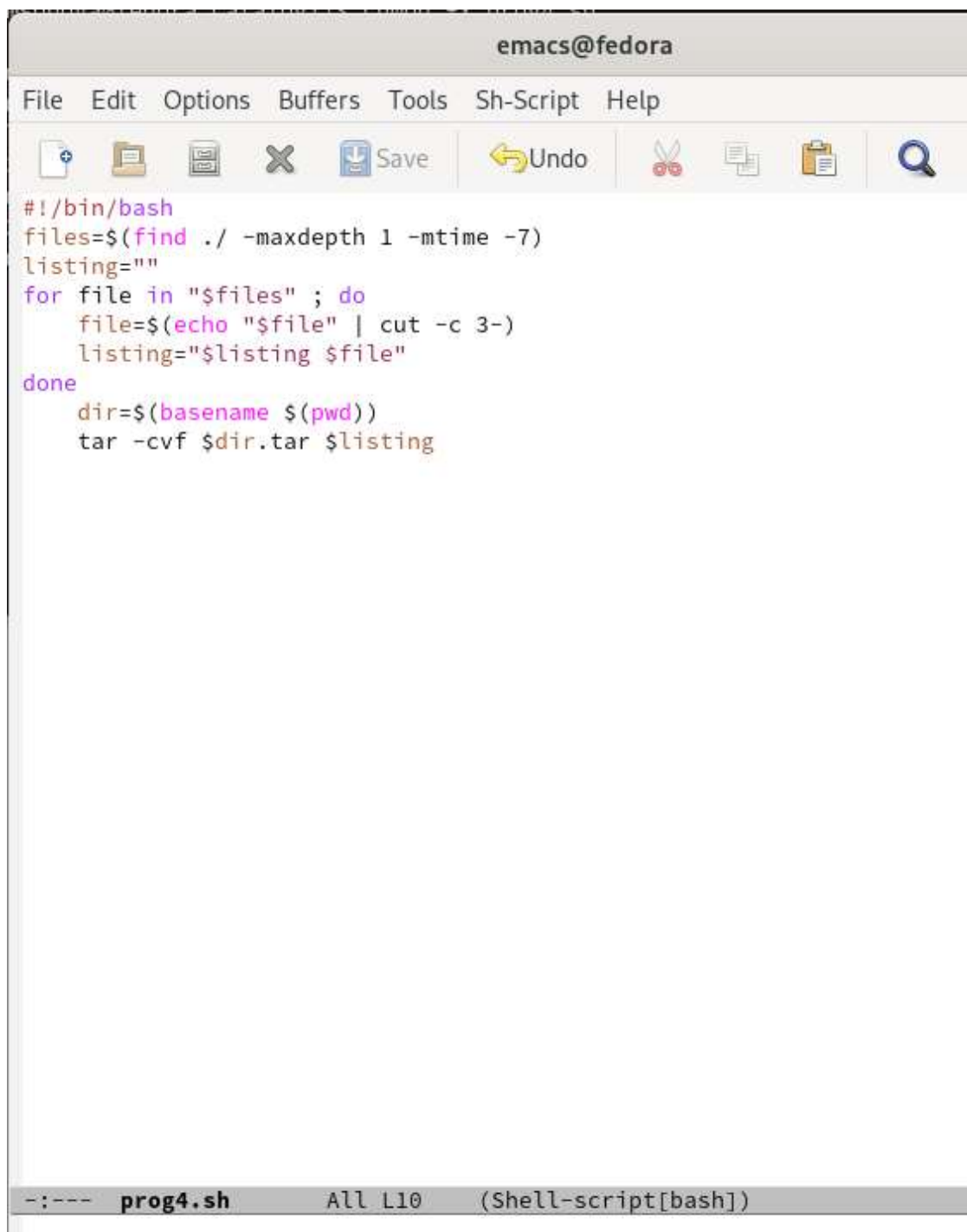
```

[~] misamsonova@fedora:~$ touch prog4.sh
[~] misamsonova@fedora:~$ emacs &

```

Создание файлов

{ #fig:012 width=70% }



```
#!/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
```

Скpunm №4

{ #fig:013 width=70% }

- Далее мы проверили работу написанного скрипта (команды «./**prog4.sh**» и «**tar -tf Catalog1.tar**»), предварительно добавив право на исполнение файла (команда «**chmod +x prog4.sh**») и создав отдельный Catalog1 с несколькими файлами. Как видно из рис. -@fig:014 , файлы, измененные более недели назад, заархивированы не были. Скрипт работает корректно.

```
misamsonova@fedora:~/Catalog1

[misamsonova@fedora Catalog1]$ chmod +x prog4.sh
[misamsonova@fedora Catalog1]$ ~/prog4.sh
a1.txt
a2.txt
chslo
chslo.c
chslo.sh
tar: Catalog1.tar: файл является архивом; не сброшен
prog4.sh~
prog4.sh
[misamsonova@fedora Catalog1]$ ./prog4.sh
a1.txt
a2.txt
chslo
chslo.c
chslo.sh
tar: Catalog1.tar: файл является архивом; не сброшен
prog4.sh~
prog4.sh
[misamsonova@fedora Catalog1]$ tar -tf Catalog1.tar
a1.txt
a2.txt
chslo
chslo.c
chslo.sh
prog4.sh~
prog4.sh
[misamsonova@fedora Catalog1]$
```

Проверка скрипта №4

{ #fig:014 width=70% }

Вывод

В процессе выполнения лабораторной работы мы изучили основы программирования в оболочке ОС UNIX и научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Ответы на 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 минуса; 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 введенные 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 введенных пользователем данных.

2. 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.
 - echprog.? – 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. 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 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` и `until false do echo hello mike done`.
6. Строка `if test -f mans/i.s` проверяет, существует ли файл `mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернёт 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.