

# Лабораторная работа №11

Программирование в командном процессоре ОС UNIX. Ветвления и  
циклы

Кузнецова София Вадимовна

# Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	8
Контрольные вопросы	19
Выводы	22

## Список иллюстраций

0.1	Создание нового файла для скрипта . . . . .	8
0.2	Написание первого скрипта . . . . .	9
0.3	Право на выполнение, запуск файла и проверка . . . . .	10
0.4	Создание двух файлов и открытие emacs . . . . .	10
0.5	Написание программы на языке Си . . . . .	11
0.6	Написание командного файла для второго задания . . . . .	12
0.7	Право на выполнение, запуск файла . . . . .	13
0.8	Создание третьего файла . . . . .	13
0.9	Написание третьего скрипта . . . . .	15
0.10	Право на выполнение, запуск файла . . . . .	16
0.11	Создание четвёртого скрипта . . . . .	16
0.12	Написание четвёртого скрипта . . . . .	17
0.13	Право на выполнение, запуск файла для каталога Catalog1 . . . . .	18
0.14	Проверка . . . . .	18

## Список таблиц

## 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;
  - `-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 `-r`.
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 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).
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

Открываем терминал и создаём в домашнем каталоге файл prog1.sh. После чего перейдём в emacs.

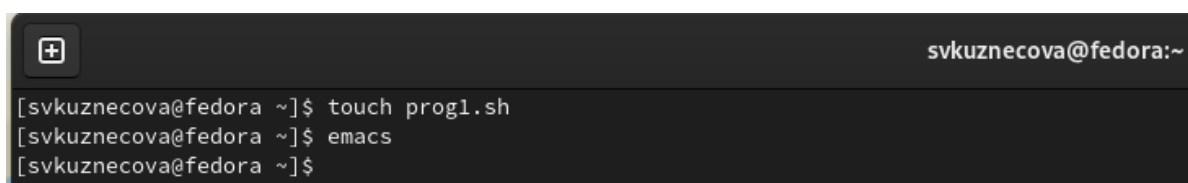
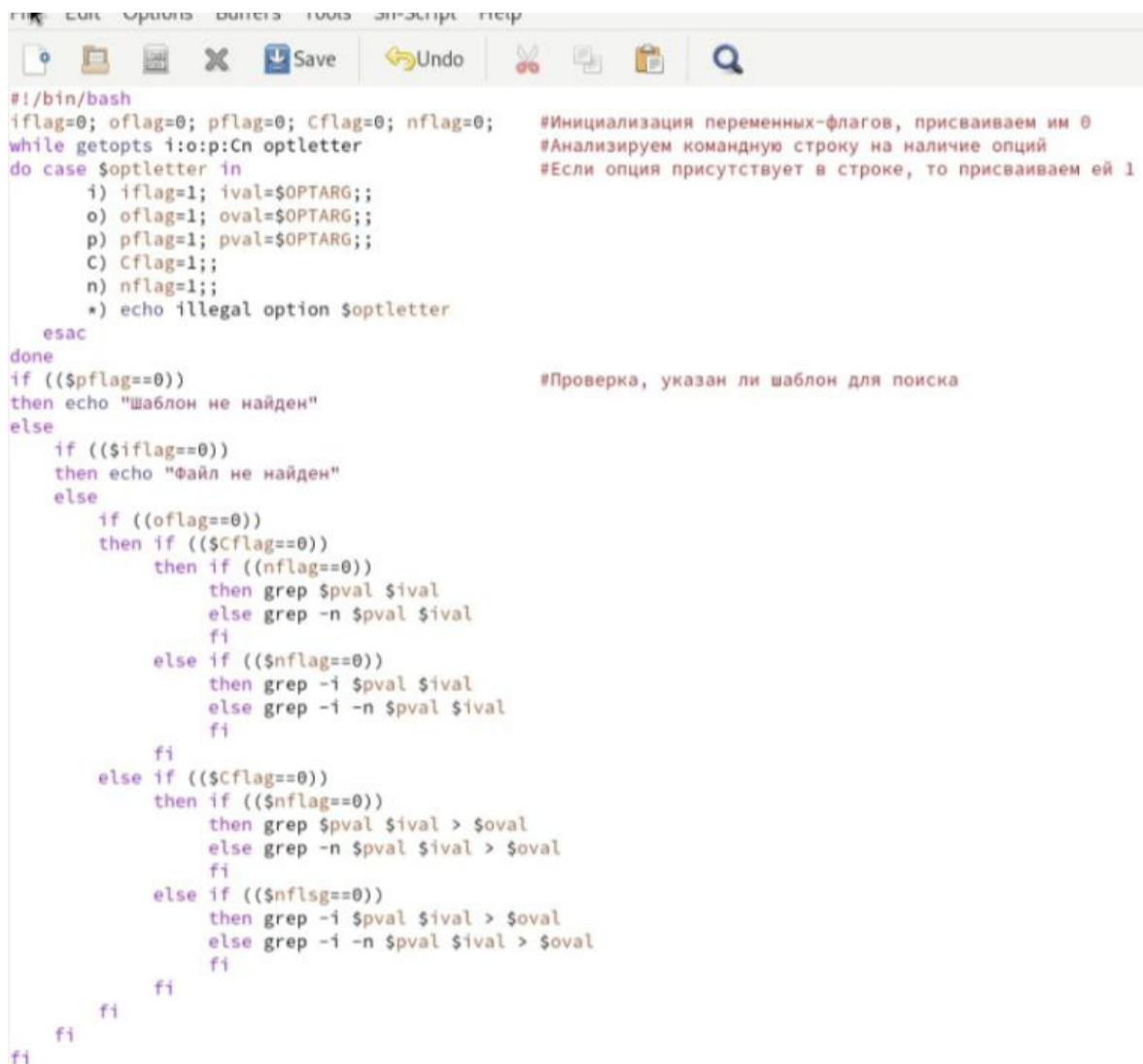
A screenshot of a terminal window with a dark background. The window title bar shows a plus icon on the left and the text 'svkuznecova@fedora:~' on the right. The terminal contains three lines of text: the first line is '[svkuznecova@fedora ~]\$ touch prog1.sh', the second line is '[svkuznecova@fedora ~]\$ emacs', and the third line is '[svkuznecova@fedora ~]\$'.

Рис. 0.1: Создание нового файла для скрипта

В emacs откроем созданный файл prog1.sh и приступим к написанию скрипта, который анализирует командную строку с определёнными ключами, а затем ищет в указанном файле нужные строки, определяемые ключом p.





```
#!/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 $optletter in
    i) iflag=1; ival=$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, указан ли шаблон 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 -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. 0.2: Написание первого скрипта

После того как скрипт написан мы сохраняем файл и закрываем emacs. В терминале мы даём этому файлу право на выполнение. Также создаём через терминал два текстовых файла (a1.txt и a2.txt). Далее прописываем нужные нам команды и проверяем корректность работы скрипта.

```

[ismahorin@fedora ~]$ touch prog1.sh
[ismahorin@fedora ~]$ emacs
[ismahorin@fedora ~]$ touch a1.txt a2.txt
[ismahorin@fedora ~]$ ls
a1.txt  abc1  '#lab07.sh#'  lab10_1.sh~  lab10_3.sh  lab10_4.sh~  work  Загрузки  Общедоступные
a2.txt  backup  lab07.sh  lab10_2.sh  lab10_3.sh~  prog1.sh  index  Изображения  'Рабочий стол'
'#abc1#'  bin  lab10_1.sh  lab10_2.sh~  lab10_4.sh  prog1.sh~  документы  Музыка  Шаблоны
[ismahorin@fedora ~]$ chmod +x prog1.sh
[ismahorin@fedora ~]$ cat a1.txt
water abc abcs
asd
prog1
water water
[ismahorin@fedora ~]$ cat a2.txt
water water water
abs
abs abs
water
water 1232
[ismahorin@fedora ~]$ ./prog1.sh -i a1.txt -o a2.txt -p water -n
grep: a1.txt: Нет такого файла или каталога
[ismahorin@fedora ~]$ ./prog1.sh -i a1.txt -o a2.txt -p water -n
[ismahorin@fedora ~]$ cat a2.txt
1:water abc abcs
4:water water
[ismahorin@fedora ~]$ ./prog1.sh -i a1.txt -o a2.txt -p water -C -n
./prog1.sh: строка 35: ((: ==0: синтаксическая ошибка: ожидается операнд (неверный маркер «==0»)
[ismahorin@fedora ~]$ emacs
[ismahorin@fedora ~]$ ./prog1.sh -i a1.txt -o a2.txt -p water -C -n
[ismahorin@fedora ~]$ cat a2.txt
1:water abc abcs
4:water water
[ismahorin@fedora ~]$ ./prog1.sh -i a1.txt -C -n
шаблон не найден
[ismahorin@fedora ~]$ ./prog1.sh -o a2.txt -p water -C -n
файл не найден
[ismahorin@fedora ~]$ clear

```

Рис. 0.3: Право на выполнение, запуск файла и проверка

Теперь открываем в терминале два файла(chslo.c и chslo.sh), для второго задания. Открываем emacs.

```

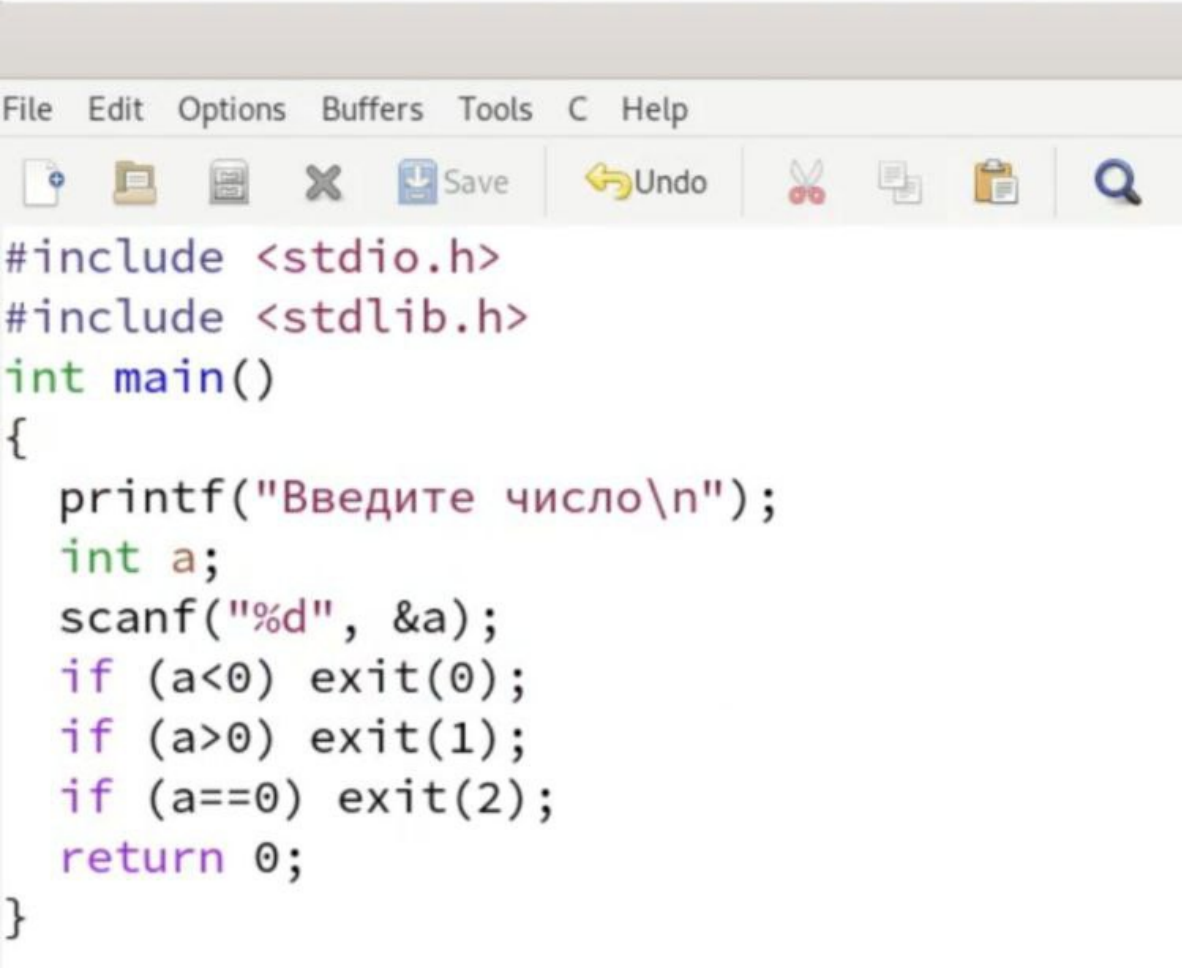
[svkuznecova@fedora ~]$ touch chslo.c
[svkuznecova@fedora ~]$ touch chslo.sh
[svkuznecova@fedora ~]$ emacs
[svkuznecova@fedora ~]$

```

Рис. 0.4: Создание двух файлов и открытие emacs

Открываем файл chslo.c и начинаем писать на языке Си программу, которая вводит

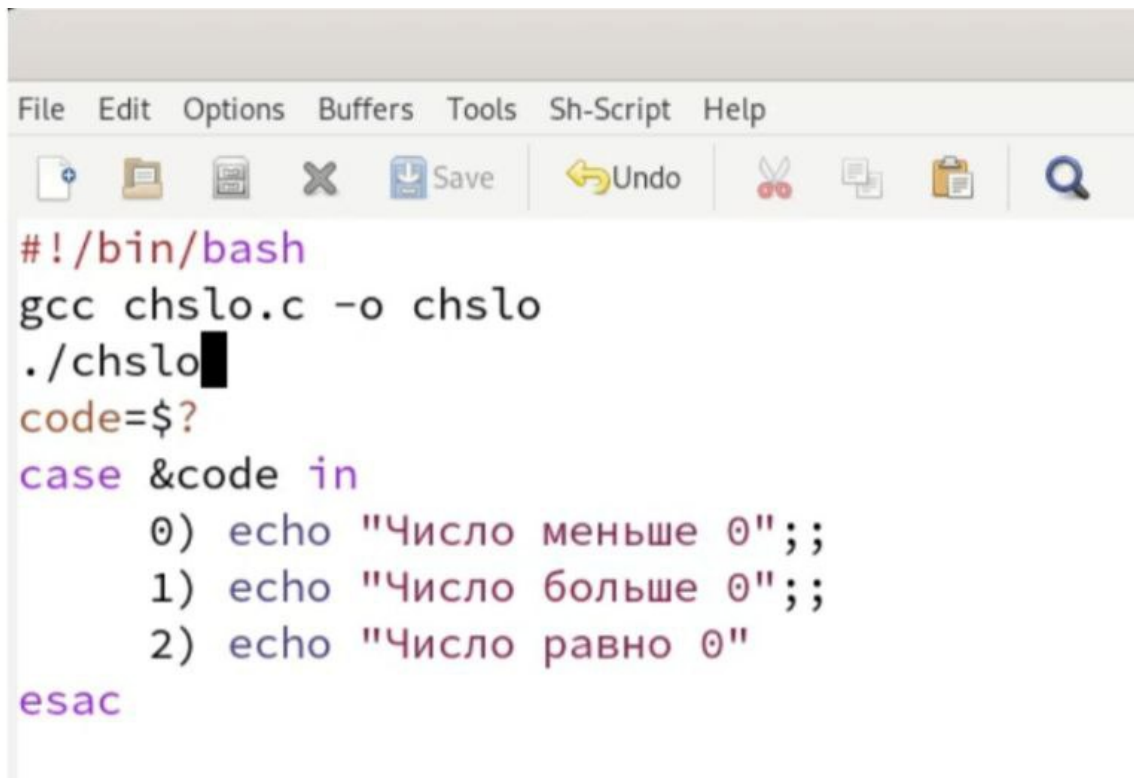
число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку.



```
#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. 0.5: Написание программы на языке Си

После программы на Си, в файле `chslo.sh` пишем командный файл, который должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было выведено.



The image shows a screenshot of a text editor window. The menu bar at the top includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Below the menu bar is a toolbar with icons for opening a file, saving, undo, redo, and search. The main text area contains a shell script with the following content:

```
#!/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. 0.6: Написание командного файла для второго задания

Сохраняем файл и также даём в терминале право на выполнение для файла chslo.sh. Запускаем файл chslo.sh.

```

[ismahorin@fedora ~]$ touch chslo.c
[ismahorin@fedora ~]$ touch chslo.sh
[ismahorin@fedora ~]$ emacs
[ismahorin@fedora ~]$ emacs
[ismahorin@fedora ~]$ chmod +x chslo.sh
[ismahorin@fedora ~]$ ./chslo.sh
Введите число
0
./chslo.sh: строка 5: синтаксическая ошибка рядом с неожиданным маркером «&»
./chslo.sh: строка 5: `case &code in'
[ismahorin@fedora ~]$ emacs
[ismahorin@fedora ~]$ ./chslo.sh
Введите число
0
Число равно 0
[ismahorin@fedora ~]$ ./chslo.sh
Введите число
0
Число равно 0
[ismahorin@fedora ~]$ ./chslo.sh
Введите число
-1
Число меньше 0
[ismahorin@fedora ~]$ ./chslo.sh
Введите число
00
Число больше 0
[ismahorin@fedora ~]$ clear

```

Рис. 0.7: Право на выполнение, запуск файла

Снова в домашнем каталоге создаём файл, но уже для третьего задания. Запускаем emacs.

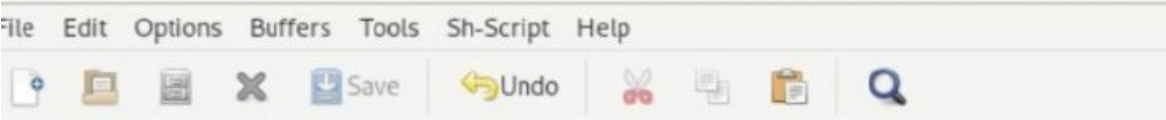
```

[svkuznecova@fedora ~]$ touch files.sh
[svkuznecova@fedora ~]$ emacs
[svkuznecova@fedora ~]$

```

Рис. 0.8: Создание третьего файла

После открытия файла `files.sh` напишем 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).



```
#!/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. 0.9: Написание третьего скрипта

Сохраняем наш скрипт и даём право на выполнение. Запускаем файл и создаём три текстовых файла. Удаляем эти три файла.

```

ismahorin@fedora ~]$ touch files.sh
ismahorin@fedora ~]$ emacs
ismahorin@fedora ~]$ chmod +x files.sh
ismahorin@fedora ~]$ ls
a1.txt  backup  chslo.c~  files.sh~  lab19_1.sh~  lab19_3.sh~  prog1.sh~  Загрузки  'Рабочий стол'
a2.txt  bin     chslo.sh  '#lab07.sh#'  lab19_2.sh  lab19_4.sh  work      Изображения  Шаблоны
#abc1#  chslo  chslo.sh~  lab07.sh  lab19_2.sh~  lab19_4.sh~  Видео     Музыка
abc1    chslo.c  files.sh  lab19_1.sh  lab19_3.sh  prog1.sh  Документы  Общедоступные
ismahorin@fedora ~]$ ./files.sh -c abc#.txt 3
ismahorin@fedora ~]$ ls
a1.txt  abc2.txt  chslo.c  files.sh~  lab19_2.sh  lab19_4.sh~  Документы  'Рабочий стол'
a2.txt  abc3.txt  chslo.c~  '#lab07.sh#'  lab19_2.sh~  prog1.sh  Загрузки  Шаблоны
#abc1#  backup  chslo.sh  lab07.sh  lab19_3.sh  prog1.sh~  Изображения
abc1    bin     chslo.sh~  lab19_1.sh  lab19_3.sh~  work      Музыка
abc1.txt  chslo  files.sh  lab19_1.sh~  lab19_4.sh  Видео     Общедоступные
ismahorin@fedora ~]$ ./files.sh -r abc#.txt 3
ismahorin@fedora ~]$ ls
a1.txt  backup  chslo.c~  files.sh~  lab19_1.sh~  lab19_3.sh~  prog1.sh~  Загрузки  'Рабочий стол'
a2.txt  bin     chslo.sh  '#lab07.sh#'  lab19_2.sh  lab19_4.sh  work      Изображения  Шаблоны
#abc1#  chslo  chslo.sh~  lab07.sh  lab19_2.sh~  lab19_4.sh~  Видео     Музыка
abc1    chslo.c  files.sh  lab19_1.sh  lab19_3.sh  prog1.sh  Документы  Общедоступные
ismahorin@fedora ~]$ clear

```

Рис. 0.10: Право на выполнение, запуск файла

Создаём последний файл для четвёртого скрипта. Запускаем emacs.

```

[svkuznecova@fedora ~]$ touch prog4.sh
[svkuznecova@fedora ~]$ emacs
[svkuznecova@fedora ~]$

```

Рис. 0.11: Создание четвёртого скрипта

В четвёртом файле напишем 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`).



A screenshot of a text editor window with a menu bar (File, Edit, Options, Buffers, Tools, Sh-Script, Help) and a toolbar with icons for file operations and editing. The editor contains a shell script with the following code:

```
#!/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. 0.12: Написание четвёртого скрипта

Сохраним файл и выйдем из emacs. Как делали ранее, дадим файлу право на выполнение и запустим его для каталога Catalog1 (который мы создали ранее в терминале и в который перенесли некоторые файлы из домашнего каталога).

```
[ismahorin@fedora ~]$ touch prog4.sh
[ismahorin@fedora ~]$ emacs
[ismahorin@fedora ~]$ chmod +x prog4.sh
[ismahorin@fedora ~]$ mkdir Catalog1
[ismahorin@fedora ~]$ ls
a1.txt  bin      chslo.sh  lab07.sh  lab10_3.sh  prog1.sh~  Документы  'Рабочий стол'
a2.txt  Catalog1 chslo.sh~ lab10_1.sh lab10_3.sh~ prog4.sh  Загрузки   Шаблоны
'#abc1#' chslo    files.sh  lab10_1.sh~ lab10_4.sh~ prog4.sh~  Изображения
abc1    chslo.c  files.sh~ lab10_2.sh  lab10_4.sh~ work      Музыка
backup  chslo.c~ '#lab07.sh#' lab10_2.sh~ prog1.sh  Видео     Общедоступные

[ismahorin@fedora ~]$ cd ~/Catalog1
[ismahorin@fedora Catalog1]$ ls
[ismahorin@fedora Catalog1]$ ls
a1.txt a2.txt abc1 chslo.c chslo.sh files.sh prog1.sh
[ismahorin@fedora Catalog1]$ ls -l
-rw-r--r-- 1 ismahorin 102216 a1.txt 102217 a2.txt 102218 abc1 102219 chslo.c 102220 chslo.sh 102221 files.sh 102222 prog1.sh
[ismahorin@fedora Catalog1]$ sudo ~/prog4.sh
[sudo] пароль для ismahorin:
a1.txt
a2.txt
chslo.c
chslo.sh
files.sh
prog1.sh
[ismahorin@fedora Catalog1]$ tar -tf Catalog1.tar
a1.txt
a2.txt
chslo.c
chslo.sh
files.sh
prog1.sh
[ismahorin@fedora Catalog1]$
[ismahorin@fedora Catalog1]$
```

Рис. 0.13: Право на выполнение, запуск файла для каталога Catalog1

Перейдём в файл и выполним проверку архивации.



Рис. 0.14: Проверка

# Контрольные вопросы

## 1. Каково предназначение команды `getopts`?

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 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. Какое отношение метасимволы имеют к генерации имён файлов?

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 `c`. 1.3. `echo prog.?` выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.` 1.4. `[a-z]` соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

### 3. Какие операторы управления действиями вы знаете?

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 `OS/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. Какие операторы используются для прерывания цикла?

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

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. Что означает строка `if test -f mans/i.$$`, встречающаяся в командном файле?

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 вернет нулевое значение (ложь).

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

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.

## Выводы

В ходе 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.