

# Лабораторная работа №11

## Отчёт по лабораторной работе №11

Макарова Анастасия Михайловна

### Содержание










#### Цель работы

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

#### Выполнение лабораторной работы

1. Откроем редактор emacs с помощью команды `emacs &` и создадим в нем файл `prog1.sh`.

Используя команды `getopts` `grep`, напомним командный файл, который анализирует командную строку с ключами: `* -i`inputfile — прочитать данные из указанного файла; `* -o`outputfile — вывести данные в указанный файл; `* -r`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 `-r` (Рис.1, 2).

```
File Edit Options Buffers Tools Sh-Script Help
     Save  Undo   

#!/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
        fi
    fi
fi

U:*** prog1.sh Top L28 (Shell-script[sh])
```

Рис.1

```
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
```

U:\*\*\* prog1.sh 38% L28 (Shell-script[sh])

Рис.2

Передадим нашему файлу права на выполнение с помощью команды `chmod` с опцией `+x`, затем создадим 2 файла для проверки работы программы (Рис.3).

```
[ammakarova@10 ~]$ chmod +x prog1.sh
[ammakarova@10 ~]$ touch a.txt
[ammakarova@10 ~]$ touch b.txt
[ammakarova@10 ~]$ mcedit a.txt

[ammakarova@10 ~]$ mcedit b.txt
```

Рис.3

Запускаем программу и видим, что она работает корректно (Рис.4).

```
[ammakarova@10 ~]$ ./prog1.sh -i a.txt -o b.txt -p money -C -n
[ammakarova@10 ~]$ cat a.txt
I wanna money money money[ammakarova@10 ~]$ cat b.txt
1:I wanna money money money
```

Рис.4

2. Создаем в emacs файлы prog2.c и prog2.sh (Рис.5, 6).

```
U:%%-  *GNU Emacs*  All L1
Find file: ~/prog2.sh
```

Рис.5

```
Find file: ~/prog2.c
```

Рис.6

Затем напишем 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 (Рис.7).

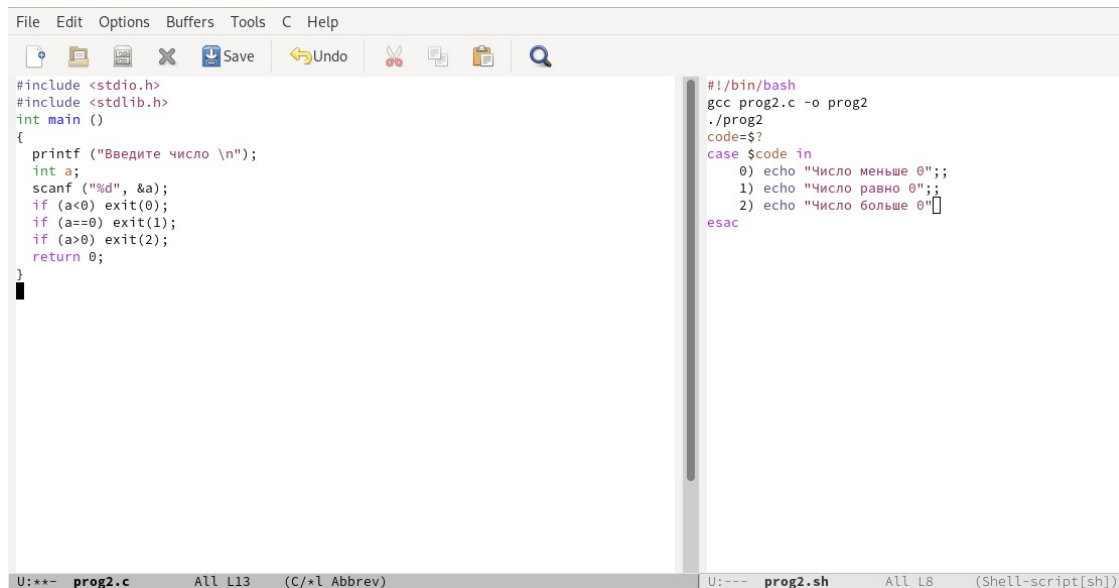


Рис.7

Добавим права на выполнение программы с помощью команды `chmod` с опцией `+x` и запустим ее (Рис.8).

```

[ammakarova@10 ~]$ chmod +x prog2.sh
[ammakarova@10 ~]$ ./prog2.sh
Введите число
7
Число больше 0
[ammakarova@10 ~]$ ./prog2.sh
Введите число
0
Число равно 0
[ammakarova@10 ~]$ ./prog2.sh
Введите число
-3
Число меньше 0
  
```

Рис.8

3. Создадим в редакторе `emacs` файл `prog3.sh` (Рис.9).

**Find file:** ~/prog3.sh

Рис.9

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, передадим в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют) (Рис.10).

```
#!/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:\*\*\*- prog3.sh All L18 (Shell-script[sh])

Рис.10

Добавим права на выполнение программы с помощью команды `chmod` с опцией `+x` и проверим ее работу (Рис.11).

```

[ammakarova@10 ~]$ chmod +x prog3.sh
[ammakarova@10 ~]$ ./prog3.sh -c abc#.txt 4
[ammakarova@10 ~]$ ls
abc1          b.txt          file2.sh       play           text.txt
abc1.txt      catalog        file3.sh       prog1.sh       work
abc2.txt      '#command_file_3#' file3.sh~      prog1.sh~      Видео
abc3.txt      command_file_3.sh file4.sh       prog2          Документы
abc4.txt      command_file_4.sh lab07.sh       prog2.c        Загрузки
a.txt         command_file_4.sh~ lab07.sh~      prog2.cpp      Изображения
australia     command_file.sh  may           prog2.sh       Музыка
backup        conf.txt        monthly        prog3.sh       Общедоступные
backup.sh     feathers        my_os         reports        'Рабочий стол'
backup.sh~    file1.sh        OS_2022       sci.plases     Шаблоны
[ammakarova@10 ~]$ ./prog3.sh -r abc#.txt 4
[ammakarova@10 ~]$ ls
abc1          command_file.sh  my_os          text.txt
a.txt         conf.txt        OS_2022        work
australia     feathers        play           Видео
backup        file1.sh        prog1.sh       Документы
backup.sh     file2.sh        prog1.sh~      Загрузки
backup.sh~    file3.sh        prog2          Изображения
b.txt         file3.sh~      prog2.c        Музыка
catalog       file4.sh        prog2.cpp      Общедоступные
'#command_file_3#' lab07.sh       prog2.sh       'Рабочий стол'
command_file_3.sh lab07.sh~      prog3.sh       Шаблоны
command_file_4.sh may            reports
command_file_4.sh~ monthly        sci.plases

```

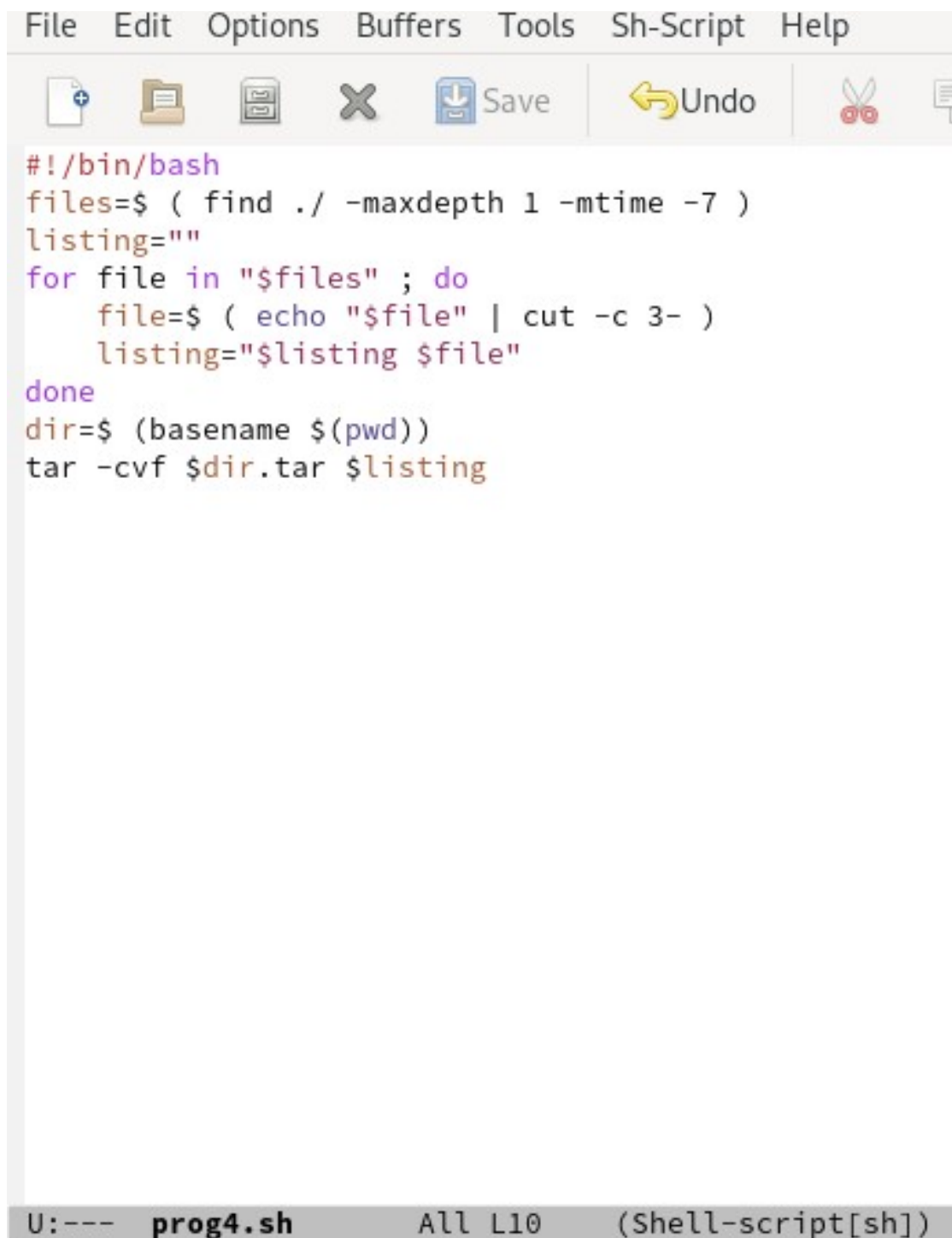
Рис.11

4. Создадим в редакторе emacs файл prog4.sh (Рис.12).

Find file: ~/prog4.sh

Рис.12

Напишем командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицируем его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (используем команду find) (Рис.13).



The image shows a graphical user interface for editing a shell script. At the top is a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Below the menu is a toolbar with icons for opening a file, saving, closing, and undo, along with text labels 'Save' and 'Undo'. The main area contains a bash script that finds files in the current directory and archives them into a tar file. The script is as follows:

```
#!/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
```

At the bottom of the window, a status bar shows the current directory as 'U:---', the filename as 'prog4.sh', the file size as 'All L10', and the editor mode as '(Shell-script[sh])'.

Рис.13

Добавим права на выполнение программы с помощью команды `chmod` с опцией `+x` и проверим ее работу (Рис.14).



```
[ammakarova@10 catalog1]$ ~/prog4.sh
prog1.sh
[ammakarova@10 catalog1]$ ls
catalog1.tar  lab07.sh  prog1.sh
```

Рис.14

## Вывод

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

## Контрольные вопросы

1. Каково предназначение команды `getopts`? 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.
2. Какое отношение метасимволы имеют к генерации имён файлов? Метасимволы - символы, имеющие специальное значение для интерпретатора: `?:*;&()|^<>`. Однако каждый из этих символов может представлять самого себя, если перед ним стоит `.` Все символы, заключенные между кавычка-ми `'` и `'`, представляют самих себя. Между двойными кавычками (`"`) выполняются подстановки команд и параметров, а символы `,` `"` и `$` могут экранироваться предшествующим символом `\`. После всех подстановок в каждом слове команды ищутся символы `*,?`, и `[`. Если находится хотя бы один из них, то это слово рассматривается как шаблон имен файлов и заменяется именами файлов, удовлетворяющих данному шаблону (в алфавитном порядке). Если ни одно имя файла не удовлетворяет шаблону, то он остается неизменным.
3. Какие операторы управления действиями вы знаете? Оператор `if` можно использовать для создания последовательности проверок, покрывающих все возможные варианты. Для этого необходимо начать с проверки первого условия с помощью оператора `if`; а затем посредством операторов `elseif` последовательно проверить все остальные условия. Поместив `else` в конец, вы перекрываете все возможные варианты. Альтернативой структурам `if-elseif-else` является оператор `switch`, работающий с допущением, что

производится сравнение одного выражения и множества возможных значений. Простейшим циклом является цикл while. Выражение проверяется сразу же при первом удобном случае. Если это условие является ложным, программный блок просто пропускается, а если условие дает значение “истина”, программный блок выполняется, после чего управление передается обратно наверх и опять проверяется условие.

4. Какие операторы используются для прерывания цикла? Использование оператора break. Он используется как в операторах цикла, так и в структурах switch. Оператор break прерывает выполнение тела любого цикла for, do или while и передает управление следующему за циклом выполняемому оператору. Еще один способ прерывания цикла использование оператора goto, передающего управление какому-то оператору, расположенному вне тела цикла. Для прерывания циклов, размещенных в функциях, можно воспользоваться оператором return. В отличие от оператора break, оператор return прервет не только выполнение цикла, но и выполнение той функции, в которой расположен цикл. Прервать выполнение цикла, а заодно и блока, в котором расположен цикл, можно также генерацией какого-то исключения. Наиболее часто в этих целях используется процедура Abort, генерирующая «молчаливое» исключение, не связанное с каким-то сообщением об ошибке.
5. Для чего нужны команды false и true? true, - всегда возвращает 0 в качестве кода выхода. false - всегда возвращает 1 в качестве кода выхода.
6. Что означает строка if test -f mans/i.\$s, встреченная в командном файле? Проверяет, существует ли этот файл и является ли он обычным файлом.
7. Объясните различия между конструкциями while и until
  - 1) В конструкции while...do проверка условия выхода выполняется вначале, а не в конце цикла, если условие не удовлетворяется до начала выполнения цикла, то управление передается оператору стоящему сразу за телом цикла.
  - 2) В конструкции while ...do условие выхода удовлетворяется, если выражение, определяющее условие выхода, ложно, а в конструкции repeat ...until - если это выражение истинно.
  - 3) Между зарезервированными словами repeat...until может размещаться не-сколько операторов не применяя операторные скобки begin end, когда, как в конструкции while ...do только один.