

Отчет по лабораторной работе №11

Операционные системы

Морозова Ульяна Константиновна

Список иллюстраций

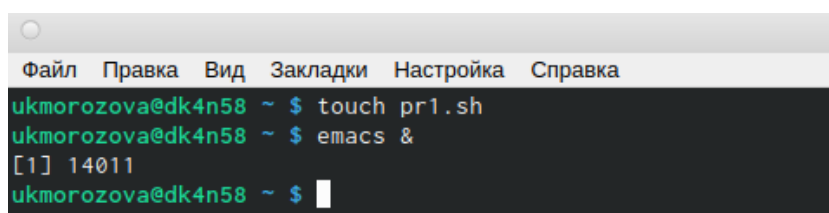
0.1. Создание файла	4
0.2. Командный файл	5
0.3. Права доступа	5
0.4. Выполнение	6
0.5. Программы	7
0.6. Выполнение	7
0.7. Командный файл	8
0.8. Выполнение	8
0.9. Командный файл	9
0.10. Выполнение	9

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. Создадим командный файл pr1.sh (рис.1) и откроем его в emacs.



```
ukmorofova@dk4n58 ~ $ touch pr1.sh
ukmorofova@dk4n58 ~ $ emacs &
[1] 14011
ukmorofova@dk4n58 ~ $
```

Рис. 0.1.: Создание файла

Используя команды `getopts` `grep`, запишем в файл программу (рис.2), которая анализирует командную строку с ключами: `-iinputfile` — прочитать данные из указанного файла; `-ooutputfile` — вывести данные в указанный файл; `-r` — 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`.

```
#!/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
```

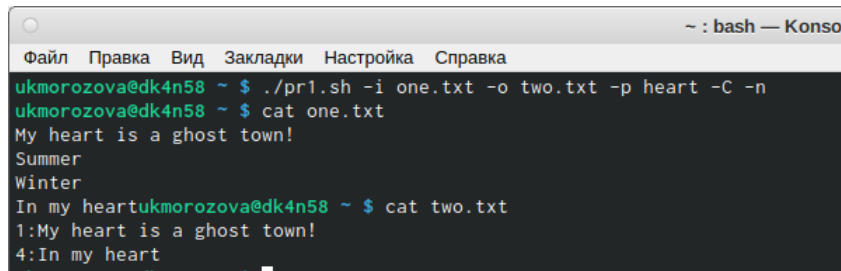
U:*** pr1.sh All L34 (Shell-script[sh]) Чт мая 12 13:59 0.62

Рис. 0.2.: Командный файл

Проверим работу файла, предварительно дав ему права на выполнение (chmod +x *.sh), и создадим два файла для проверки работы (touch one.txt two.txt) (рис.3). Запускаем файл (рис.4).

```
~ : bash —
Файл Правка Вид Закладки Настройка Справка
ukmorofova@dk4n58 ~ $ touch pr1.sh
ukmorofova@dk4n58 ~ $ emacs &
[1] 14011
ukmorofova@dk4n58 ~ $ chmod +x *.sh
ukmorofova@dk4n58 ~ $ touch one.txt
ukmorofova@dk4n58 ~ $ touch two.txt
ukmorofova@dk4n58 ~ $ mcedit one.txt
ukmorofova@dk4n58 ~ $ mcedit two.txt
```

Рис. 0.3.: Права доступа

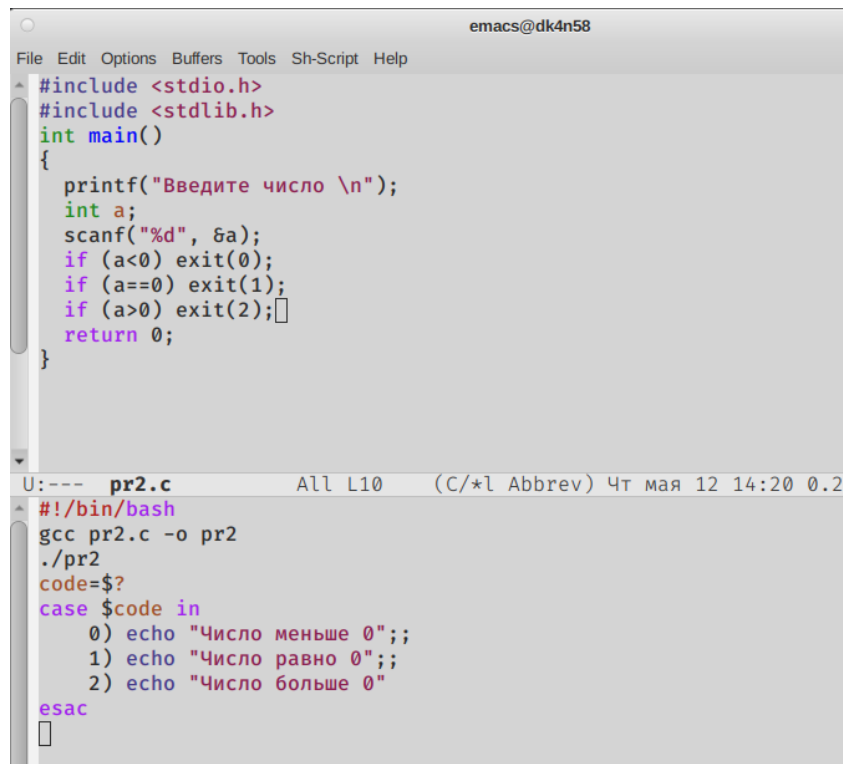


```
~ : bash — Konso
Файл  Правка  Вид  Закладки  Настройка  Справка
ukmorofova@dk4n58 ~ $ ./pr1.sh -i one.txt -o two.txt -p heart -C -n
ukmorofova@dk4n58 ~ $ cat one.txt
My heart is a ghost town!
Summer
Winter
In my heart
ukmorofova@dk4n58 ~ $ cat two.txt
1:My heart is a ghost town!
4:In my heart
```

Рис. 0.4.: Выполнение

2. Создадим два файла для третьего задания (команда `touch pr2.c pr2.sh`) и откроем в `emacs`.

Затем напифем 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.5)



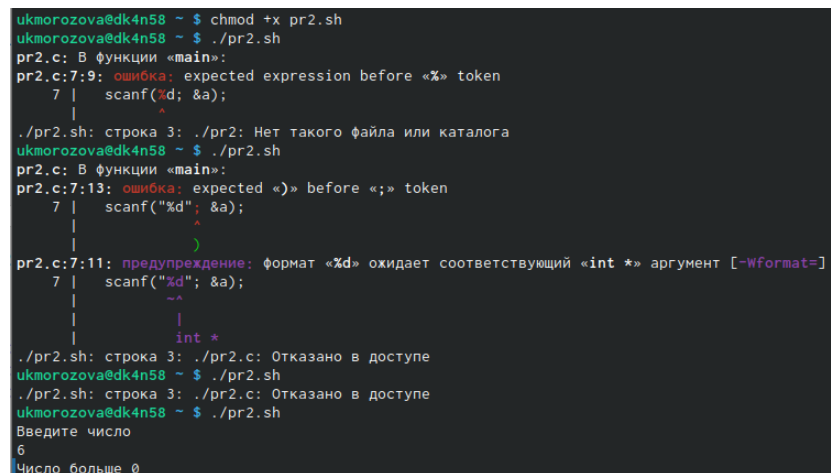
The screenshot shows the Emacs editor interface. The top window displays the source code for `pr2.c`, which includes `<stdio.h>` and `<stdlib.h>`, and contains a `main` function that prompts the user for a number and checks if it is less than, equal to, or greater than zero. The bottom window shows the shell script `pr2.sh`, which compiles `pr2.c` using `gcc` and then uses a `case` statement to echo messages based on the program's exit code.

```
emacs@dk4n58
File Edit Options Buffers Tools Sh-Script 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;
}

U:--- pr2.c All L10 (C/*l Abbrev) Чт мая 12 14:20 0.2
#!/bin/bash
gcc pr2.c -o pr2
./pr2
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число равно 0";;
    2) echo "Число больше 0";;
esac
```

Рис. 0.5.: Программы

Проверим работу командного файла, передав ему права на выполнения и запустив его (рис.6).



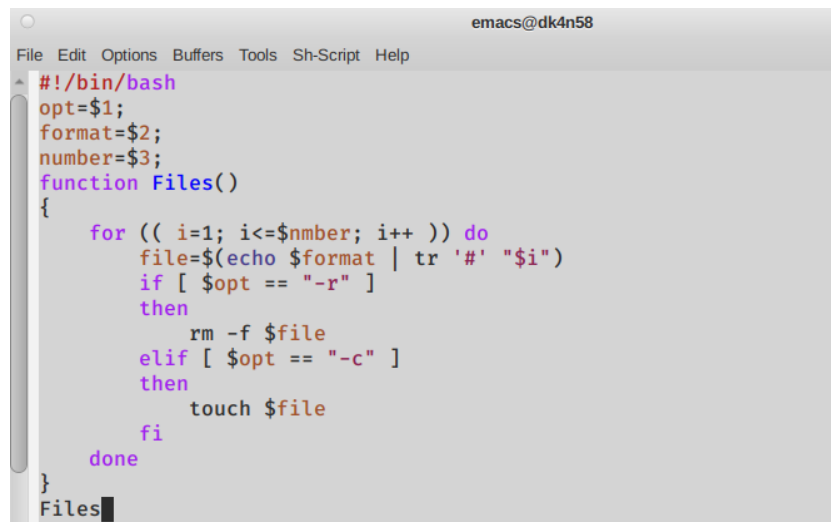
The screenshot shows a terminal session where the user attempts to run `pr2.sh`. The script fails with several errors: a permission error, a file not found error, and two syntax errors in the `scanf` calls. After correcting the syntax, the script runs successfully, prompting the user for a number and displaying the result.

```
ukmorozova@dk4n58 ~$ chmod +x pr2.sh
ukmorozova@dk4n58 ~$ ./pr2.sh
pr2.c: В функции «main»:
pr2.c:7:9: ошибка: expected expression before «%» token
7 | scanf(%d; &a);
  |         ^
./pr2.sh: строка 3: ./pr2: Нет такого файла или каталога
ukmorozova@dk4n58 ~$ ./pr2.sh
pr2.c: В функции «main»:
pr2.c:7:13: ошибка: expected «)» before «;» token
7 | scanf("%d"; &a);
  |             ^
pr2.c:7:11: предупреждение: формат «%d» ожидает соответствующий «int *» аргумент [-Wformat=]
7 | scanf("%d"; &a);
  |             ~^
  |             |
  |             int *
./pr2.sh: строка 3: ./pr2.c: Отказано в доступе
ukmorozova@dk4n58 ~$ ./pr2.sh
./pr2.sh: строка 3: ./pr2.c: Отказано в доступе
ukmorozova@dk4n58 ~$ ./pr2.sh
Введите число
6
Число больше 0
```

Рис. 0.6.: Выполнение

3. Создадим командный файл files.sh и откроем его в emacs.

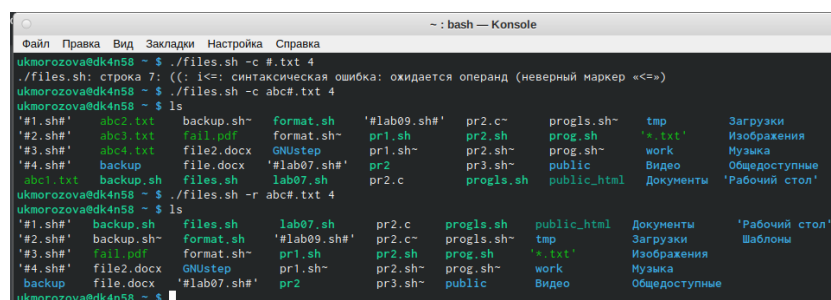
Напишем командный файл (рис.7), создающий указанное число файлов, пронумерованных последовательно от 1 до ∞ (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).



```
emacs@dk4n58
File Edit Options Buffers Tools Sh-Script Help
#!/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. 0.7.: Командный файл

Проверим его работу, передав ему права на выполнения и запустив его (команда ./files.sh) (рис.8)

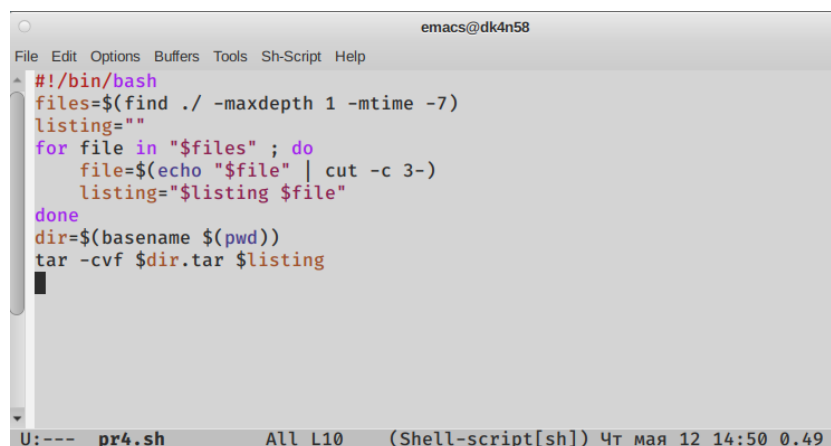


```
~ : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
ukmorofova@dk4n58 ~ $ ./files.sh -c #.txt 4
./files.sh: строка 7: ((: i<=: синтаксическая ошибка: ожидается операнд (неверный маркер «<=»)
ukmorofova@dk4n58 ~ $ ./files.sh -c abc#.txt 4
ukmorofova@dk4n58 ~ $ ls
'1.sh#'  abc2.txt  backup.sh~  format.sh  'lab09.sh#'  pr2.c~  progl.s.sh~  tmp  Загрузки
'2.sh#'  abc3.txt  fail.pdf   format.sh~  pr1.sh      pr2.sh    prog.sh     '*.*txt'  Изображения
'3.sh#'  abc4.txt  file2.docx GNUstep    pr1.sh~     pr2.sh~   prog.sh~    work  Музыка
'4.sh#'  backup   file.docx  'lab07.sh#' pr2         pr3.sh~   public      Видео  Общедоступные
abc1.txt backup.sh files.sh   lab07.sh  pr2.c      progl.s.sh public_html Документы 'Рабочий стол'
ukmorofova@dk4n58 ~ $ ./files.sh -r abc#.txt 4
ukmorofova@dk4n58 ~ $ ls
'1.sh#'  backup.sh  files.sh  lab07.sh  pr2.c  progl.s.sh  public_html  Документы  'Рабочий стол'
'2.sh#'  backup.sh~  format.sh  'lab09.sh#'  pr2.c~  progl.s.sh~  tmp  Загрузки  Шаблоны
'3.sh#'  fail.pdf   format.sh~  pr1.sh      pr2.sh    prog.sh     '*.*txt'  Изображения
'4.sh#'  file2.docx GNUstep    pr1.sh~     pr2.sh~   prog.sh~    work  Музыка
backup   file.docx  'lab07.sh#'  pr2         pr3.sh~   public      Видео  Общедоступные
ukmorofova@dk4n58 ~ $
```

Рис. 0.8.: Выполнение

4. Создадим командный файл pr4.sh и откроем его в emacs.

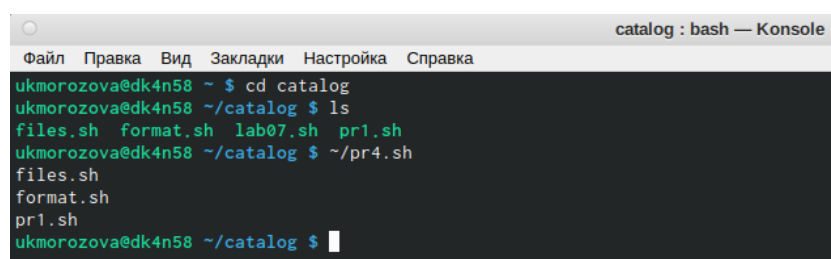
Напишем командный файл (рис.9), который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицируем его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).



```
#!/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. 0.9.: Командный файл

Создадим в домашнем каталоге каталог catalog и перенесем туда некоторые файлы, измененные в разное время. Дадим командному файлу право на выполнение (chmod +x pr4.sh) и запустим его в этом каталоге (рис.10). Файл работает исправно.



```
ukmorofova@dk4n58 ~ $ cd catalog
ukmorofova@dk4n58 ~/catalog $ ls
files.sh format.sh lab07.sh pr1.sh
ukmorofova@dk4n58 ~/catalog $ ~/pr4.sh
files.sh
format.sh
pr1.sh
ukmorofova@dk4n58 ~/catalog $
```

Рис. 0.10.: Выполнение

Выводы

Я научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Контрольные вопросы

1). 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, 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:

–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, `1.1 echo` – выведет имена всех файлов текущего каталога, что представляет собой простей-

ший аналог команды `ls`; 1.2. `ls.c`–выведет все файлы с последними двумя символами, совпадающими с.с. 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). 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.