

Отчёт по лабораторной работе №11

**Программирование в командном процессоре ОС UNIX. Ветвления и
циклы**

Дарья Сергеевна Кочина

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	18

Список иллюстраций

4.1	Создание файла	7
4.2	Скрипт №1	8
4.3	Скрипт №1	8
4.4	Скрипт №1	8
4.5	Предоставление прав доступа	9
4.6	Проверка работы программы	9
4.7	Создание файлов	10
4.8	Работа в файле	10
4.9	Работа в файле	10
4.10	Работа в файле	11
4.11	Создание файлов	11
4.12	Скрипт	12
4.13	Проверка работы программы	13
4.14	Создание файла	13
4.15	Скрипт	14

1 Цель работы

Целью данной лабораторной работы является изучение основ программирования в оболочке ОС UNIX. А также приобретение практических навыков написания более сложные командных файлов с использованием логических управляющих конструкций и циклов.

2 Задание

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

3 Теоретическое введение

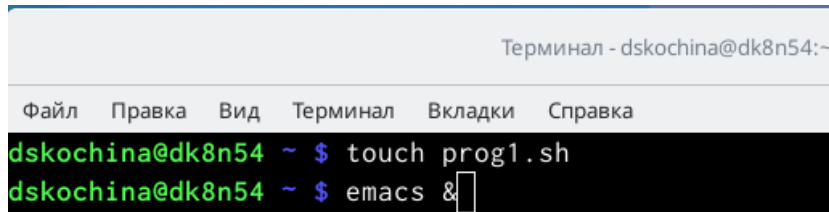
Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg ...]`

Флаги – это опции командной строки, обычно помеченные знаком минус; Например, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`.

4 Выполнение лабораторной работы

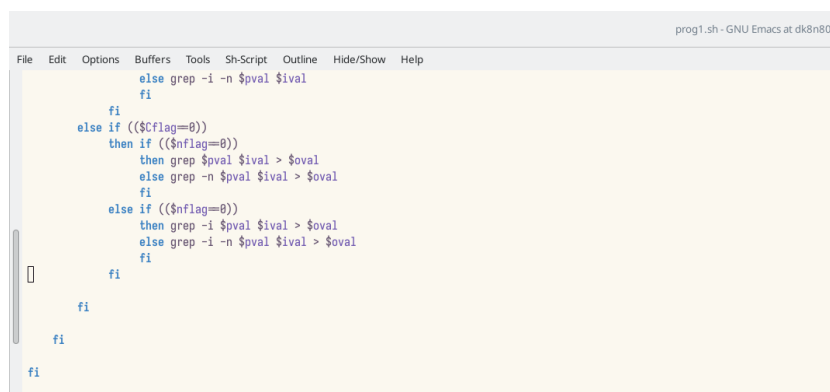
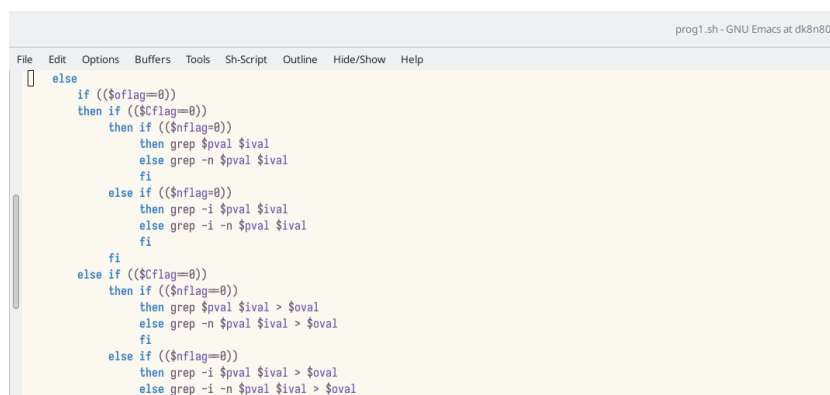
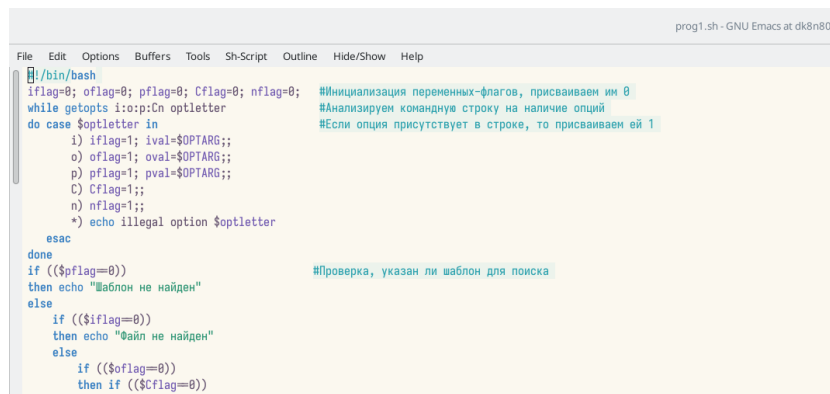
1. Используя команды `getopts` `grep`, написала командный файл, который анализирует командную строку с ключами:

-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 я создала файл `prog1.sh` и написала соответствующие скрипты. (рис. [4.1], [4.2], [4.3], [4.4])



The image shows a terminal window titled "Терминал - dskochina@dk8n54:~". The window has a menu bar with "Файл", "Правка", "Вид", "Терминал", "Вкладки", and "Справка". The terminal content shows two commands being entered at the prompt "dskochina@dk8n54 ~ \$":
1. `touch prog1.sh`
2. `emacs &` (with a cursor at the end of the command)

Рис. 4.1: Создание файла



2. Проверила работу написанного скрипта, используя различные опции (на-

пример, команда «./prog.sh -Ia1.txt -oa2.txt -pcapital -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. [4.5], [4.6])

```
dskochina@dk8n54 ~ $ touch a1.txt a2.txt
dskochina@dk8n54 ~ $ chmod +x prog1.sh
dskochina@dk8n54 ~ $
```

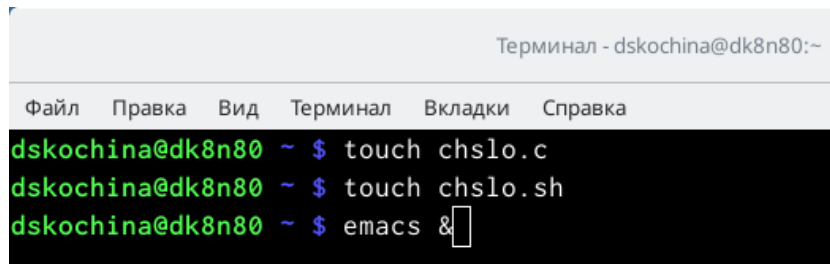
Рис. 4.5: Предоставление прав доступа

```
dskochina@dk8n80 ~ $ emacs &
[1] 3813
dskochina@dk8n80 ~ $ cat a1.txt
water abc abcs
asd
prog1
water water
[1]+  Завершён          emacs
dskochina@dk8n80 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -n
dskochina@dk8n80 ~ $ cat a2.txt
1:water abc abcs
4:water water
dskochina@dk8n80 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -n
dskochina@dk8n80 ~ $ cat a2.txt
1:water abc abcs
4:water water
dskochina@dk8n80 ~ $ ./prog1.sh -i a1.txt -C -n
Шаблон не найден
dskochina@dk8n80 ~ $ ./prog1.sh -o a2.txt -p water -C -n
Файл не найден
dskochina@dk8n80 ~ $
```

Рис. 4.6: Проверка работы программы

3. Написала 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. Для данной задачи я создала 2 файла: `chslo.c` и `chislo.sh`

и написала соответствующие скрипты. (команды «touch prog2.sh» и «emacs &»). (рис. [4.7], [4.8], [4.9], [4.10])

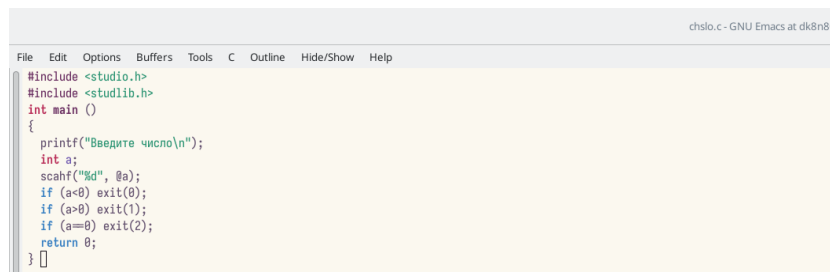


Терминал - dskochina@dk8n80:~

Файл Правка Вид Терминал Вкладки Справка

```
dskochina@dk8n80 ~ $ touch chslo.c
dskochina@dk8n80 ~ $ touch chslo.sh
dskochina@dk8n80 ~ $ emacs &
```

Рис. 4.7: Создание файлов

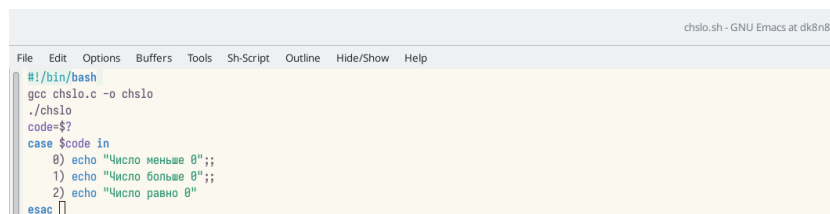


chslo.c - GNU Emacs at dk8n80

File Edit Options Buffers Tools C Outline Hide/Show 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;
}
```

Рис. 4.8: Работа в файле

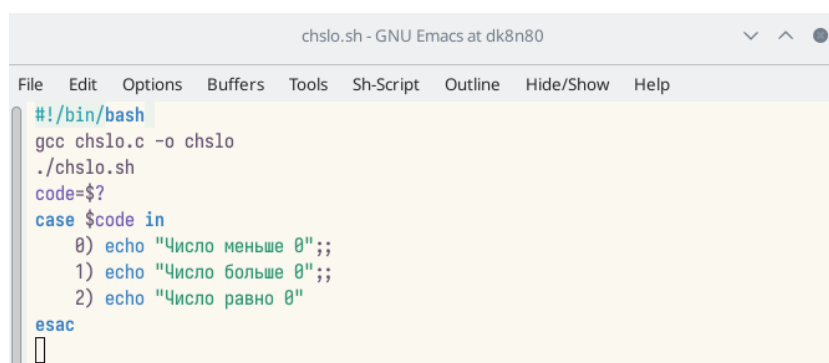


chslo.sh - GNU Emacs at dk8n80

File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help

```
#!/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
```

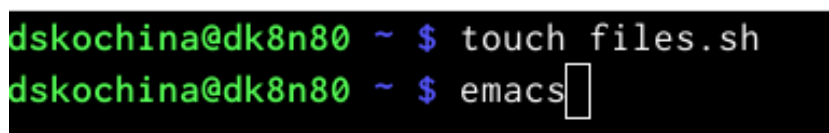
Рис. 4.9: Работа в файле



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


Рис. 4.10: Работа в файле

4. Проверила работу написанных скриптов (команда «./chislo.sh»), предварительно добавив право на исполнение файла (команда «chmod+x chislo.sh»). Скрипты работают корректно.
5. 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: files.sh. и написала соответствующий скрипт. (рис. [4.11], [4.12])



```
dskochina@dk8n80 ~ $ touch files.sh
dskochina@dk8n80 ~ $ emacs
```

Рис. 4.11: Создание файлов



```
files.sh - GNU Emacs at dk8n80
File Edit Options Buffers Tools Sh-Script Outline Hide/Show 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. 4.12: Скрипт

6. Далее я проверила работу написанного скрипта (команда «./files.sh»), предварительно добавив право на исполнение файла (команда «chmod+x files.sh»). Сначала я создала три файла (команда «./files.sh-cabc#.txt3»), удовлетворяющие условию задачи, а потом удалила их (команда «./files.sh-rabc#.txt3»). (рис. [4.13])

```

dskochina@dk8n80 ~ $ emacs
dskochina@dk8n80 ~ $ chmod +x files.sh
dskochina@dk8n80 ~ $ ls
a1.txt      chslo.sh      GNUstep      public      Изображения
a2.txt      chslo.sh~     my_os        public_html Музыка
backup      file2.doc     prog1.sh     ski.plases Общедоступные
backup.sh   file.doc      prog1.sh~    tmp         'Рабочий стол'
backup.sh~  file.pdf     prog2.sh     work        Шаблоны
bin         files.sh      prog2.sh~    Видео
chslo.c     format.sh    progl.s.sh   Документы
chslo.c~    format.sh~   progl.s.sh~ Загрузки

dskochina@dk8n80 ~ $ ./files.sh -c abc#.txt 3
dskochina@dk8n80 ~ $ ls
a1.txt      backup.sh~    file.doc     prog1.sh     public_html  Изображения
a2.txt      bin           file.pdf     prog1.sh~    ski.plases   Музыка
abc1.txt    chslo.c      files.sh     prog2.sh     tmp          Общедоступные
abc2.txt    chslo.c~     format.sh    prog2.sh~    work         'Рабочий стол'
abc3.txt    chslo.sh     format.sh~   progl.s.sh   Видео        Шаблоны
backup      chslo.sh~    GNUstep     progl.s.sh~  Документы
backup.sh   file2.doc    my_os       public       Загрузки

dskochina@dk8n80 ~ $ ./files.sh -r abc#.txt 3
dskochina@dk8n80 ~ $ ls
a1.txt      chslo.sh      GNUstep      public      Изображения
a2.txt      chslo.sh~     my_os        public_html Музыка
backup      file2.doc     prog1.sh     ski.plases Общедоступные
backup.sh   file.doc      prog1.sh~    tmp         'Рабочий стол'
backup.sh~  file.pdf     prog2.sh     work        Шаблоны
bin         files.sh      prog2.sh~    Видео
chslo.c     format.sh    progl.s.sh   Документы
chslo.c~    format.sh~   progl.s.sh~ Загрузки

dskochina@dk8n80 ~ $ █

```

Рис. 4.13: Проверка работы программы

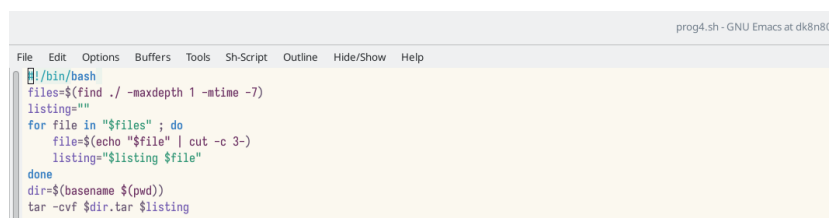
7. Написала командный файл, который с помощью команды `tag` запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`). Для данной задачи я создала файл: `prog4.sh` и написала соответствующий скрипт. (рис. [4.14], [4.15])

```

dskochina@dk8n80 ~ $ touch prog4.sh
dskochina@dk8n80 ~ $ emacs █

```

Рис. 4.14: Создание файла



```
prog4.sh - GNU Emacs at dk8n80
File Edit Options Buffers Tools Sh-Script Outline Hide/Show 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
```

Рис. 4.15: Скрипт

8. Далее я проверила 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 Рисунков, файлы, измененные более недели назад, заархивированы не были. Скрипт работает корректно.

Ответы на контрольные вопросы:

1. Команда getoptс осуществляет синтаксический анализ командной строки, выделяя флаги, используется для объявления переменных. Синтаксис команды следующий: getoptс 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 getoptс может распознать аргумент, то она возвращает истину. Принято включать getoptс в цикл while и анализировать введенные данные с помощью оператора case. Функция getoptс включает две специальные переменные среды –OPTARG и OPTIND. Если ожидается дополнительное значение, то OPTARG устанавливается в значение этого аргумента. Функция getoptс 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, 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. 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. А также приобрела практические навыки написания более сложные командных файлов с использованием логических управляющих конструкций и циклов.