

Отчёт по лабораторной работе №13

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

Дауд Амжад

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	командный файл, который анализирует командную строку	7
3.2	программа, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю.	8
3.3	командный файл, создающий указанное число файлов	11
3.4	командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории.	12
4	Выводы	14
5	Ответы на контрольные вопросы	15
	Список литературы	19

Список иллюстраций

3.1	Код для анализов командной строки	7
3.2	право на исполнение	8
3.3	Запуск file1	8
3.4	Программа на языке Си	9
3.5	Командный файл программы на Си	10
3.6	Результаты программы	11
3.7	Командный файл для создания файлов	11
3.8	Создание файлов с помощью командного файла	12
3.9	Создание архива	12
3.10	Результаты кода	13

Список таблиц

1 Цель работы

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

2 Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку
2. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю.
3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N
4. Написать командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории.

3 Выполнение лабораторной работы

3.1 командный файл, который анализирует командную строку

Создаю файл file1 и в нем написала код, который анализирует командную строку с ключами -i (прочитать данные из указанного файла), -o (вывести данные в указанный файл), -p (указать шаблон для поиска), -C (различать большие и малые буквы), -n (выдавать номера строк) используя команды getoptс grep:



```
1 while getopt "i:o:p:C:n" opt
2 do
3 case $opt in
4 i)inputfile="$OPTARG";;
5 o)outputfile="$OPTARG";;
6 p)template="$OPTARG";;
7 c)register="$OPTARG";;
8 n)number="";;
9 esac
10 done
11
12 grep -n "$template" "$inputfile.txt" > "$outputfile.txt"
```

Рис. 3.1: Код для анализования командной строки

```
while getopt "i:o:p:C:n" opt
do
case $opt in
i)inputfile="$OPTARG";;
o)outputfile="$OPTARG";;
p)template="$OPTARG";;
```

```
c)register="$OPTARG";;
n)number="";;
esac
done
```

```
grep -n "$template" "$inputfile.txt" > "$outputfile.txt"
```

Далее я установила права на исполнение и запустила файл:

```
[amjaddawud@vbox ~]$ gedit file1
(gedit:2535): Gtk-WARNING **: 12:23:26.624: Calling org.freedesktop.portal.Inhibit on
Method: No such interface "org.freedesktop.portal.Inhibit" on
[amjaddawud@vbox ~]$ cp file1 file1.txt
[amjaddawud@vbox ~]$ chmod
chmod: missing operand
Try 'chmod --help' for more information.
[amjaddawud@vbox ~]$ chmod +x file1.txt
```

Рис. 3.2: право на исполнение

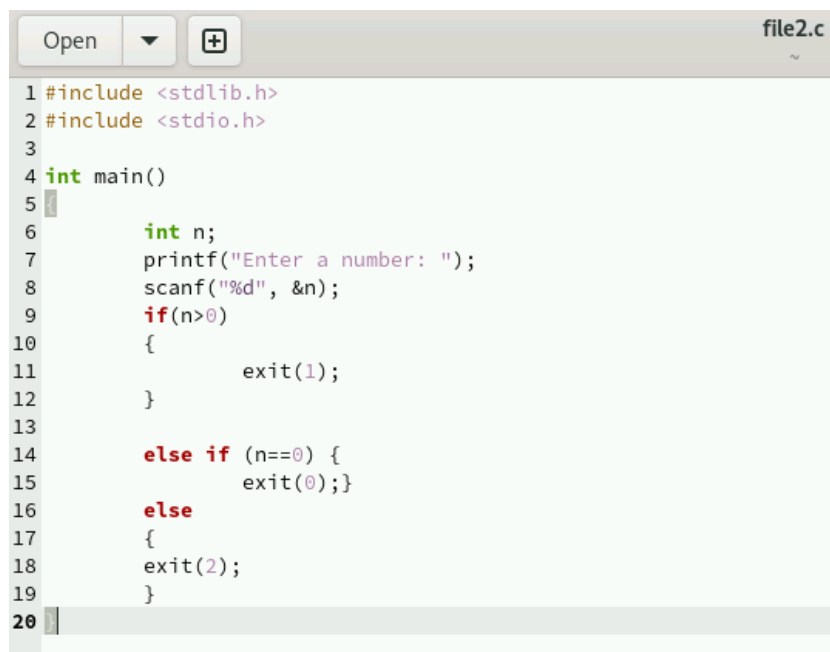
```
[amjaddawud@vbox ~]$ ./file1.txt -i file1 -o output -p n etconf -C -n
[amjaddawud@vbox ~]$ cat output.txt
1:while getopts "i:o:p:C:n" opt
3:case $opt in
4:i)inputfile="$OPTARG";;
8:n)number="";;
10:done
12:grep -n "$template" "$inputfile.txt" > "$outputfile.txt"
```

Рис. 3.3: Запуск file1

3.2 программа, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю.

Написала на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа заверша-

ется с помощью функции `exit(n)`, передавая информацию о коде завершения в оболочку:



```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main()
5 {
6     int n;
7     printf("Enter a number: ");
8     scanf("%d", &n);
9     if(n>0)
10    {
11        exit(1);
12    }
13
14    else if (n==0) {
15        exit(0);}
16
17    else
18    {
19        exit(2);
20    }
```

Рис. 3.4: Программа на языке Си

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int n;
```

```
    printf("Enter a number: ");
```

```
    scanf("%d", &n);
```

```
    if(n>0)
```

```
    {
```

```
        exit(1);
```

```
    }
```

```
    else if (n==0) {
```

```

        exit(0);}
else
{
    exit(2);
}
}

```

Далее создала командный файл который вызывает эту программу и, проанализировав с помощью команды \$?, выдает сообщение о том, какое число было введено:

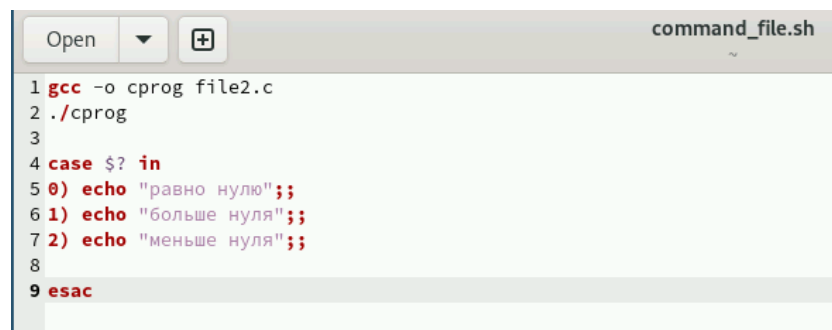


Рис. 3.5: Командный файл программы на Си

```

gcc -o cprog file2.c
./cprog

```

```

case $? in
0) echo "равно нулю";;
1) echo "больше нуля";;
2) echo "меньше нуля";;

esac

```

Создала исполняемый файл и запустила:

```

[amjaddawud@vbox ~]$ gedit file2.c

(gedit:4461): Gtk-WARNING **: 12:31:36.474: Calling org.freedesktop
Method: No such interface "org.freedesktop.portal.Inhibit" on obj
[amjaddawud@vbox ~]$ gedit command_file.sh

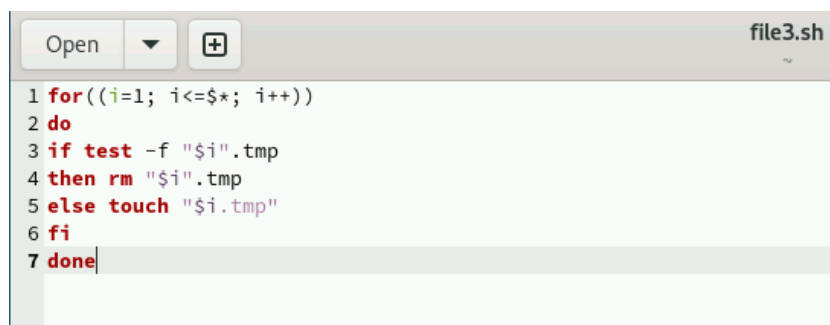
(gedit:5577): Gtk-WARNING **: 12:36:15.643: Calling org.freedesktop
Method: No such interface "org.freedesktop.portal.Inhibit" on obj
[amjaddawud@vbox ~]$ chmod +x file.sh
chmod: cannot access 'file.sh': No such file or directory
[amjaddawud@vbox ~]$ chmod +x command_file.sh
[amjaddawud@vbox ~]$ ./command_file.sh
Enter a number: 7
больше нуля

```

Рис. 3.6: Результаты программы

3.3 командный файл, создающий указанное число файлов

Я написала командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до ☐. Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют):



```

1 for((i=1; i<=$*; i++))
2 do
3 if test -f "$i".tmp
4 then rm "$i".tmp
5 else touch "$i".tmp
6 fi
7 done

```

Рис. 3.7: Командный файл для создания файлов

```

for((i=1; i<=$*; i++))
do
if test -f "$i".tmp
then rm "$i".tmp

```

```
else touch "$i.tmp"
fi
done
```

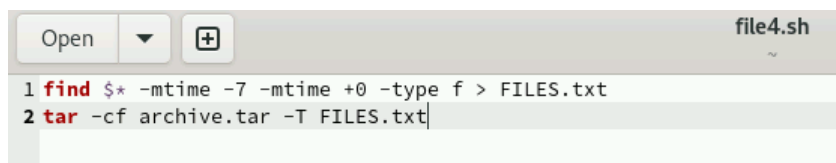
Создала исполняемый файл и запустила:

```
[amjaddawud@vbox ~]$ chmod +x file3.sh
[amjaddawud@vbox ~]$ ./file3.sh 3
[amjaddawud@vbox ~]$ ls
1.tmp      bin          Downloads   file3.sh
2.tmp      command_file.sh feathers    file.txt
3.tmp      conf.txt     file1       fun
abc1       cprog        file1.sh~   games
australia  Desktop      file1.txt   lab
backup     Documents    file2.c     '#lab07.sh#'
```

Рис. 3.8: Создание файлов с помощью командного файла

3.4 командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории.

создала командный файл, который с помощью команды tar 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).



```
Open [v] [x] file4.sh
1 find $* -mtime -7 -mtime +0 -type f > FILES.txt
2 tar -cf archive.tar -T FILES.txt
```

Рис. 3.9: Создание архива

```
find $* -mtime -7 -mtime +0 -type f > FILES.txt
```

```
tar -cf archive.tar -T FILES.txt
```

```
[amjaddawud@vbox ~]$ gedit file4.sh

(gedit:10448): Gtk-WARNING **: 12:53:25.373: Calling on
nMethod: No such interface "org.freedesktop.portal.Inhi
[amjaddawud@vbox ~]$ chmod +x files4.sh
chmod: cannot access 'files4.sh': No such file or direc
[amjaddawud@vbox ~]$ ./file4.sh /home/amjaddawud/work
bash: ./file4.sh: Permission denied
[amjaddawud@vbox ~]$ ls work/
blog os study
[amjaddawud@vbox ~]$ ./file4.sh /home/amjaddawud/work/
bash: ./file4.sh: Permission denied
[amjaddawud@vbox ~]$ ./file4.sh /
afs/      boot/      etc/      lib/      lost+fo
bin/      dev/      home/    lib64/    media/
[amjaddawud@vbox ~]$ ./file4.sh /home/amjaddawud/work/
bash: ./file4.sh: Permission denied
[amjaddawud@vbox ~]$ chmod +x file4.sh
[amjaddawud@vbox ~]$ ./file4.sh /home/amjaddawud/work/
tar: Removing leading `/' from member names
tar: Removing leading `/' from hard link targets
[amjaddawud@vbox ~]$ ls work/
blog os study
[amjaddawud@vbox ~]$ ls
1.tmp      backup      Documents  file2.c
2.tmp      bin         Downloads  file3.sh
3.tmp      command_file.sh feathers    file4.sh
abc1       conf.txt   file1      FILES.txt
archive.tar cprog      file1.sh~  file.txt
australia  Desktop    file1.txt  fun
```

Рис. 3.10: Результаты кода

4 Выводы

При выполнении проделанной работы я научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Ответы на контрольные вопросы

1. Осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable`. 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`. Предположим, необходимо распознать командную строку следующего формата: `testprog -ifile_in.txt -ofile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае:

```
while getopts o:i:Ltr optletter do
case $optletter in
o) oflag = 1; oval = OPTARG;;
i) iflag=1; ival=$OPTARG;;
L) Lflag=1;;
t) tflag=1;;
r) rflag=1;;
*) echo Illegal option $optletter
esac
done
```

 Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равна `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упомянутый аргумент. Функция `getopts` также по-

нимает переменные типа массив, следовательно, можно использовать ее 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`. `echo prog.?` – 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 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, не равный нулю (т. е. ложь).
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.

Список литературы

Архитектура коипьютеров