

Отчёт по лабораторной работе №13

Операционные системы

Аветисян Алина Эдуардовна

Содержание

1	Цель работы	1
2	Задание	1
3	Теоретическое введение.....	2
4	Выполнение лабораторной работы.....	2
5	Выводы.....	6
6	Ответы на контрольные вопросы.....	7
	Список литературы	9

1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

- 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;
 - `-ршаблон` — указать шаблон для поиска;
 - `-С` — различать большие и малые буквы;
 - `-п` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-р`.
- 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. Команд- ный 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).

3 Теоретическое введение

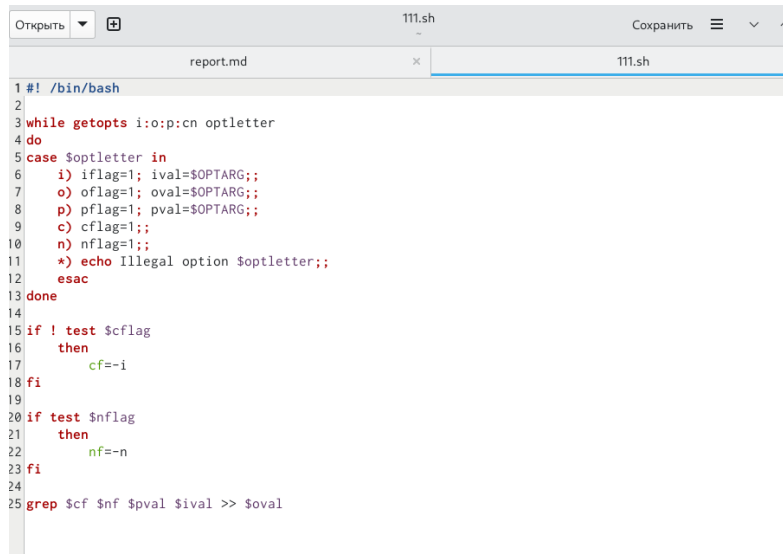
4 Выполнение лабораторной работы

Создаю файл с разрешением на исполнение (рис. fig. 1).

```
aeavetisyan@dk2n24 ~$ touch 111.sh
aeavetisyan@dk2n24 ~$ chmod +x 111.sh
aeavetisyan@dk2n24 ~$ bash 111.sh -p улир -i input.txt -o output.txt -c -n
aeavetisyan@dk2n24 ~$
```

Рис. 1: Создание файла

Командный файл, с командами getoptс и грег, 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 -p (рис. fig. 2).



```
Открыть 111.sh Сохранить
report.md 111.sh
1 #!/bin/bash
2
3 while getoptс i:o:p:cn optletter
4 do
5 case $optletter in
6   i) iflag=1; ival=$OPTARG;;
7   o) oflag=1; oval=$OPTARG;;
8   p) pflag=1; pval=$OPTARG;;
9   c) cflag=1;;
10  n) nflag=1;;
11  *) echo illegal option $optletter;;
12  esac
13 done
14
15 if ! test $cflag
16 then
17   cf=-i
18 fi
19
20 if test $nflag
21 then
22   nf=-n
23 fi
24
25 grep $cf $nf $pval $ival >> $oval
```

Рис. 2: Код программы

```

#!/bin/bash

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 ! test $cflag
then
    cf=-i
fi

if test $nflag
then
    nf=-n
fi

grep $cf $nf $pval $ival >> $oval

```

Результат работы программы в файле output.txt (рис. fig. 3).

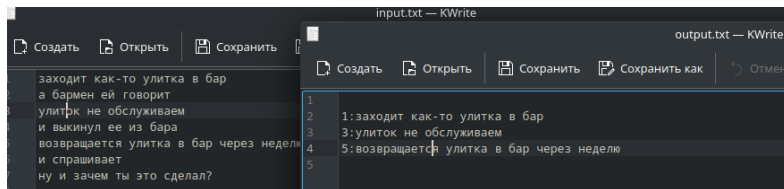


Рис. 3: Результат работы программы

Создаю исполняемый файл для второй программы, также создаю файл 12.c для программы на Си (рис. fig. 4).

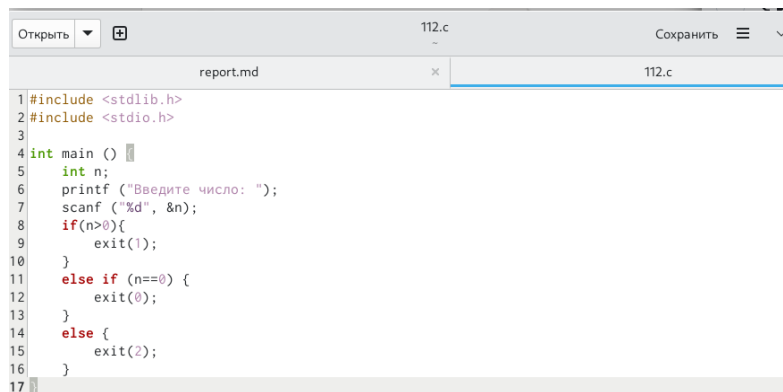
```

aeavetisyan@dk2n24 ~ $ touch 112.sh
aeavetisyan@dk2n24 ~ $ chmod +x 112.sh
aeavetisyan@dk2n24 ~ $ touch 112.cpp
aeavetisyan@dk2n24 ~ $ touch 112.sh
aeavetisyan@dk2n24 ~ $

```

Рис. 4: Создание файла

Пишу программу на языке Си, 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 (рис. fig. 5).



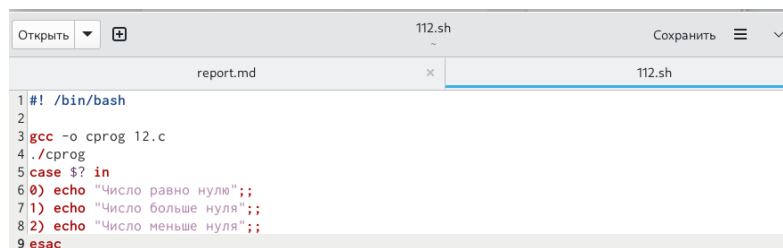
```
1#include <stdlib.h>
2#include <stdio.h>
3
4int main () {
5    int n;
6    printf ("Введите число: ");
7    scanf ("%d", &n);
8    if(n>0){
9        exit(1);
10    }
11    else if (n==0) {
12        exit(0);
13    }
14    else {
15        exit(2);
16    }
17}
```

Рис. 5: Код программы на Си

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

int main () {
    int n;
    printf ("Введите число: ");
    scanf ("%d", &n);
    if(n>0){
        exit(1);
    }
    else if (n==0) {
        exit(0);
    }
    else {
        exit(2);
    }
}
```

Командный файл должен вызывать эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено (рис. fig. 6).



```
1#!/bin/bash
2
3gcc -o cprog 12.c
4./cprog
5case $? in
60) echo "Число равно нулю";;
71) echo "Число больше нуля";;
82) echo "Число меньше нуля";;
9esac
```

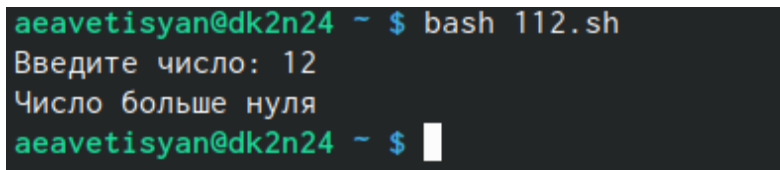
Рис. 6: Код программы

```
#!/bin/bash

gcc -o cprog 12.c
./cprog
case $? in
0) echo "Число равно нулю";;
1) echo "Число больше нуля";;
```

```
2) echo "Число меньше нуля";;  
esac
```

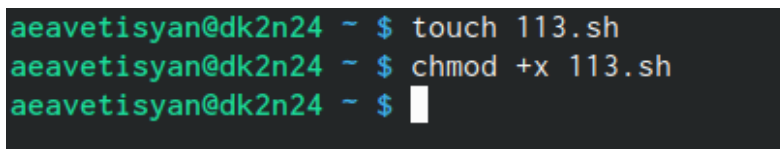
Программа работает корректно (рис. fig. 7).



```
aeavetisyan@dk2n24 ~ $ bash 112.sh  
Введите число: 12  
Число больше нуля  
aeavetisyan@dk2n24 ~ $
```

Рис. 7: Результат работы программы

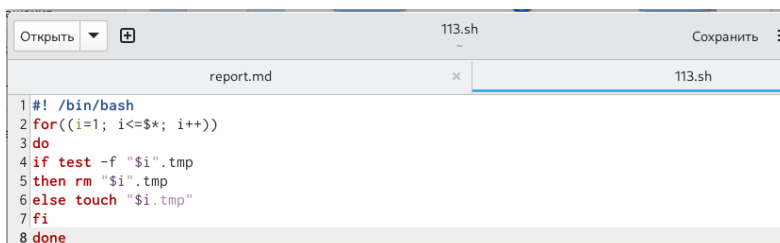
Создаю исполняемый файл для третьей программы (рис. fig. 8).



```
aeavetisyan@dk2n24 ~ $ touch 113.sh  
aeavetisyan@dk2n24 ~ $ chmod +x 113.sh  
aeavetisyan@dk2n24 ~ $
```

Рис. 8: Создание файла

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) (рис. fig. 9).



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

Рис. 9: Код программы

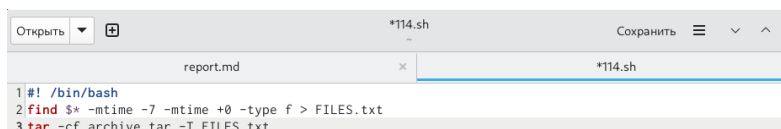
```
#!/bin/bash  
for((i=1; i<=$*; i++))  
do  
if test -f "$i".tmp  
then rm "$i".tmp  
else touch "$i.tmp"  
fi  
done
```

Проверяю, что программа создала файлы и удалила их при соответствующих запросах (рис. fig. 10).

```
aeavetisyan@dk2n24 ~ $ bash 113.sh 4
aeavetisyan@dk2n24 ~ $ ls
111.sh  2.tmp  input.txt  prog4.sh  Видео  'Рабочий стол'
112.cpp 3.tmp  1.log     project  Документы  Шаблоны
112.sh  4.tmp  output.txt public  Загрузки
113.sh  backup prog1.sh  public_html  Изображения
12.c   cprog  prog2.sh  tmp        Музыка
1.tmp  GNUstep prog3.sh  work       Общедоступные
aeavetisyan@dk2n24 ~ $ bash 113.sh 4
aeavetisyan@dk2n24 ~ $ ls
111.sh  backup  output.txt  project  Видео  Общедоступные
112.cpp cprog   prog1.sh    public   Документы  'Рабочий стол'
112.sh  GNUstep prog2.sh    public_html  Загрузки  Шаблоны
113.sh  input.txt prog3.sh    tmp        Изображения
12.c   1.log   prog4.sh    work       Музыка
aeavetisyan@dk2n24 ~ $
```

Рис. 10: Результат работы программы

Создаю исполняемый файл для четвертой программы. Это 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`) (рис. fig. 11).



```
#!/bin/bash
find $* -mtime -7 -mtime +0 -type f > FILES.txt
tar -cf archive.tar -T FILES.txt
```

Рис. 11: Код программы

```
#!/bin/bash
find $* -mtime -7 -mtime +0 -type f > FILES.txt
tar -cf archive.tar -T FILES.txt
```

Проверяю работу программы (рис. fig. 12).

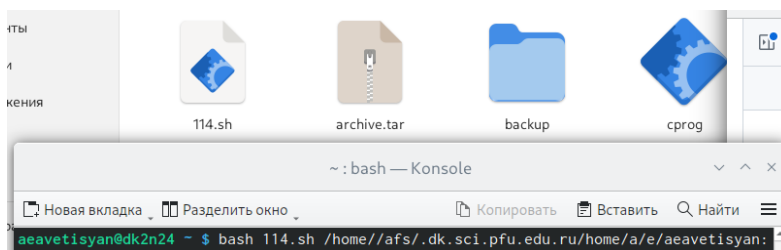


Рис. 12: Результат работы программы

5 Выводы

При выполнении данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX, научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

6 Ответы на контрольные вопросы

1. Каково предназначение команды getopts?

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 $optletter in o) iflag = 1; oval =OPTARG;; i) iflag=1; ival=$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 понимает переменные типа массив, следовательно, можно использовать ее 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: `-` – 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. Какие операторы управления действиями вы знаете?

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов 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. 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, не равный нулю (т.е. ложь).

6. Что означает строка `if test -f mans/i.$s`, встреченная 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 вернёт 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.

Список литературы