

# **Отчёт по лабораторной работе №13**

**Дисциплина: Операционные системы**

Верниковская Екатерина Андреевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>4</b>	<b>Ответы на контрольные вопросы</b>	<b>17</b>
<b>5</b>	<b>Выводы</b>	<b>21</b>
<b>6</b>	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

3.1	Создание файла lab13_1.sh и добавление прав на исполнение . . .	8
3.2	Написанная программа для lab13_1.sh . . . . .	9
3.3	Создание txt файла . . . . .	10
3.4	txt файл . . . . .	10
3.5	Проверка работы командного файла lab13_1.sh (1) . . . . .	11
3.6	Проверка работы командного файла lab13_1.sh (2) . . . . .	11
3.7	Проверка работы командного файла lab13_1.sh (3) . . . . .	11
3.8	Создание файла lab13_2.sh и добавление прав на исполнение . . .	11
3.9	Написанная программа для lab13_2.sh . . . . .	12
3.10	Написанная программа на си . . . . .	12
3.11	Проверка работы командного файла lab13_2.sh . . . . .	13
3.12	Создание файла lab13_3.sh и добавление прав на исполнение . . .	14
3.13	Написанная программа для lab13_3.sh . . . . .	14
3.14	Проверка работы командного файла lab13_3.sh (1) . . . . .	15
3.15	Проверка работы командного файла lab13_3.sh (2) . . . . .	15
3.16	Создание файла lab13_4.sh и добавление прав на исполнение . . .	15
3.17	Написанная программа для lab13_4.sh . . . . .	16
3.18	Проверка работы командного файла lab13_4.sh . . . . .	16

## 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 с ключами:
  - `-iinputfile` — прочитать данные из указанного файла;
  - `-ooutputfile` — вывести данные в указанный файл;
  - `-р`шаблон — 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`.
2. 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. Командный 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 Выполнение лабораторной работы

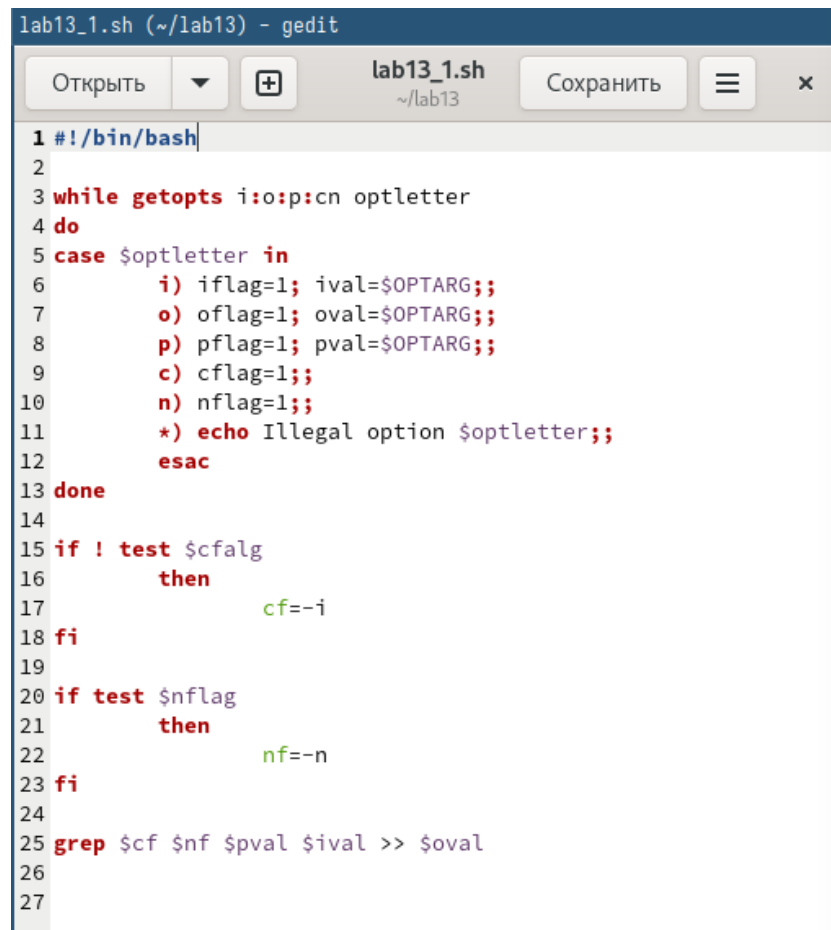
Создаю файл для первого задания с расширением sh и делаю его исполняемым (рис. 3.1)

```
[eavernikovskaya@eavernikovskaya ~]$ touch lab13_1.sh  
[eavernikovskaya@eavernikovskaya ~]$ chmod +x lab13_1.sh  
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.1: Создание файла lab13\_1.sh и добавление прав на исполнение

Открываю файл lab13\_1.sh в текстовом редакторе gedit и пишу командный файл, который будет анализировать командную строку с ключами (см. в задании №1) (рис. 3.2)





```
lab13_1.sh (~/.lab13) - gedit
Открыть  lab13_1.sh  Сохранить
1 #!/bin/bash
2
3 while getopts 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
26
27
```

Рис. 3.2: Написанная программа для lab13\_1.sh

Программа для задания №1:

```
#!/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

```

Далее создаю файл input.txt с любым текстом (рис. 3.3), (рис. 3.4)

```

[eavernikovskaya@eavernikovskaya ~]$ touch input.txt
[eavernikovskaya@eavernikovskaya ~]$

```

Рис. 3.3: Создание txt файла

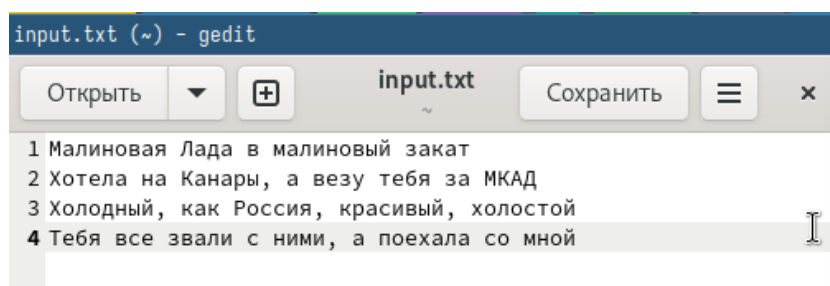


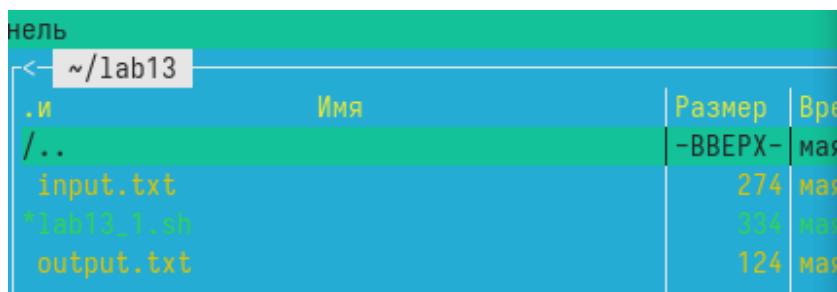
Рис. 3.4: txt файл

Далее запускаю файл с помощью bash и проверяю работу командного файла. Во время работы программы создался файл output.txt с нужным содержимым

(рис. 3.5), (рис. 3.6), (рис. 3.7)

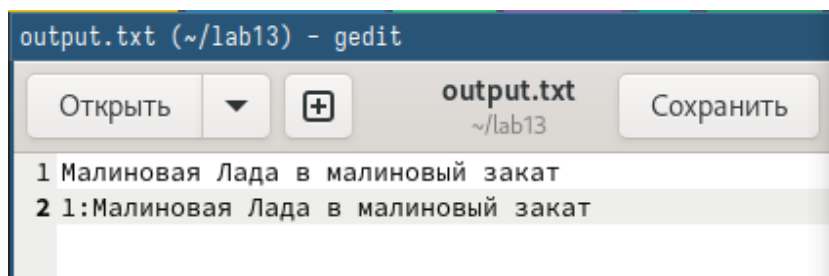
```
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_1.sh -p малинов -i input.txt -o output.txt -c -n
[eavernikovskaya@eavernikovskaya lab13]$
```

Рис. 3.5: Проверка работы командного файла lab13\_1.sh (1)



Имя	Размер	Вре
../	-ВВЕРХ-	мая
input.txt	274	мая
*lab13_1.sh	334	мая
output.txt	124	мая

Рис. 3.6: Проверка работы командного файла lab13\_1.sh (2)



```
output.txt (~/.lab13) - gedit
Открыть  output.txt  Сохранить
~/.lab13
1 Малиновая Лада в малиновый закат
2 1:Малиновая Лада в малиновый закат
```

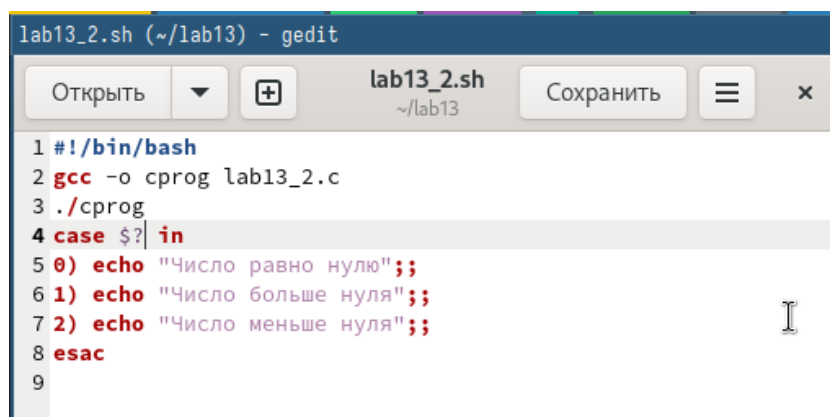
Рис. 3.7: Проверка работы командного файла lab13\_1.sh (3)

Создаю файл для второго задания с расширением sh и делаю его исполняемым (рис. 3.8)

```
[eavernikovskaya@eavernikovskaya lab13]$ touch lab13_2.sh
[eavernikovskaya@eavernikovskaya lab13]$ chmod +x lab13_2.sh
[eavernikovskaya@eavernikovskaya lab13]$
```

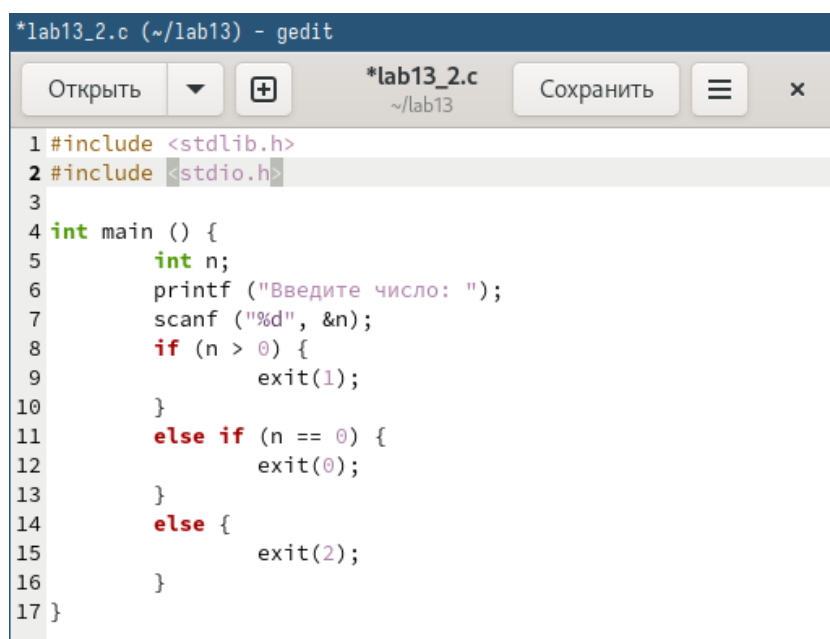
Рис. 3.8: Создание файла lab13\_2.sh и добавление прав на исполнение

Открываю файл lab13\_2.sh в текстовом редакторе gedit и пишу командный файл и программу на языке си, которая будет выводить число и определять, является ли оно больше нуля, меньше нуля или равно нулю. (рис. 3.9), (рис. 3.10)



```
lab13_2.sh (~/.lab13) - gedit
1 #!/bin/bash
2 gcc -o cprog lab13_2.c
3 ./cprog
4 case $? in
5 0) echo "Число равно нулю";;
6 1) echo "Число больше нуля";;
7 2) echo "Число меньше нуля";;
8 esac
9
```

Рис. 3.9: Написанная программа для lab13\_2.sh



```
*lab13_2.c (~/.lab13) - gedit
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int 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. 3.10: Написанная программа на си

Программа для задания №2:

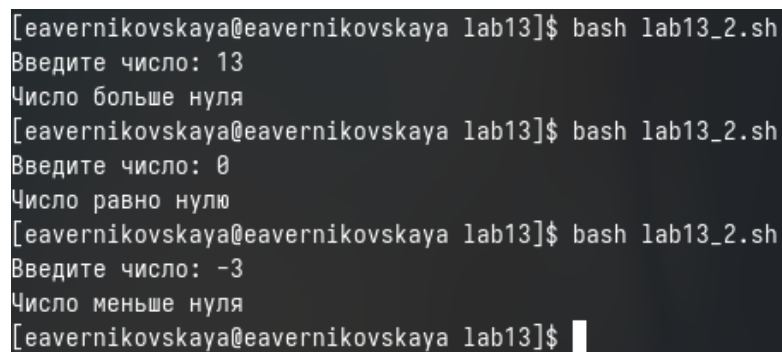
```
#!/bin/bash
gcc -o cprog lab13_2.c
./cprog
case $? in
0) echo "Число равно нулю";;
```

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

Программа на языке си

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

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.11)



```
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_2.sh  
Введите число: 13  
Число больше нуля  
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_2.sh  
Введите число: 0  
Число равно нулю  
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_2.sh  
Введите число: -3  
Число меньше нуля  
[eavernikovskaya@eavernikovskaya lab13]$
```

Рис. 3.11: Проверка работы командного файла lab13\_2.sh

Создаю файл для третьего задания с расширением sh и делаю его исполняемым (рис. 3.12)

```
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_2.sh
Введите число: 13
Число больше нуля
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_2.sh
Введите число: 0
Число равно нулю
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_2.sh
Введите число: -3
Число меньше нуля
[eavernikovskaya@eavernikovskaya lab13]$
```

Рис. 3.12: Создание файла lab13\_3.sh и добавление прав на исполнение

Открываю файл lab13\_3.sh в текстовом редакторе gedit и пишу 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. 3.13)



```
lab13_3.sh (~/.lab13) - gedit
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
```

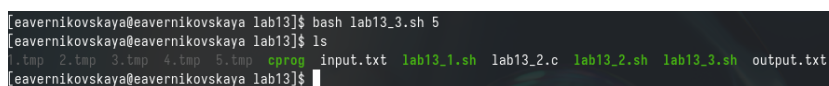
Рис. 3.13: Написанная программа для lab13\_3.sh

Программа для задания №3:

```
#!/bin/bash
for((i=1; i<=$*; i++))
```

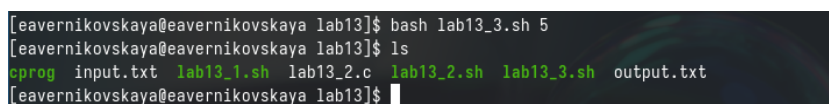
```
do
if test -f "$i".tmp
then rm "$i".tmp
else touch "$i.tmp"
fi
done
```

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.14), (рис. 3.15)



```
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_3.sh 5
[eavernikovskaya@eavernikovskaya lab13]$ ls
.tmp 2.tmp 3.tmp 4.tmp 5.tmp cprog input.txt lab13_1.sh lab13_2.c lab13_2.sh lab13_3.sh output.txt
[eavernikovskaya@eavernikovskaya lab13]$
```

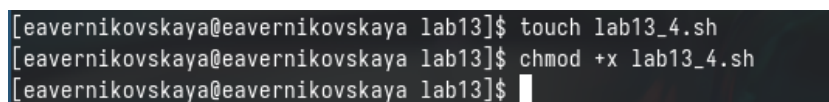
Рис. 3.14: Проверка работы командного файла lab13\_3.sh (1)



```
[eavernikovskaya@eavernikovskaya lab13]$ bash lab13_3.sh 5
[eavernikovskaya@eavernikovskaya lab13]$ ls
cprog input.txt lab13_1.sh lab13_2.c lab13_2.sh lab13_3.sh output.txt
[eavernikovskaya@eavernikovskaya lab13]$
```

Рис. 3.15: Проверка работы командного файла lab13\_3.sh (2)

Создаю файл для четвёртого задания с расширением sh и делаю его исполняемым (рис. 3.16)



```
[eavernikovskaya@eavernikovskaya lab13]$ touch lab13_4.sh
[eavernikovskaya@eavernikovskaya lab13]$ chmod +x lab13_4.sh
[eavernikovskaya@eavernikovskaya lab13]$
```

Рис. 3.16: Создание файла lab13\_4.sh и добавление прав на исполнение

Открываю файл lab13\_4.sh в текстовом редакторе gedit и пишу командный файл, который с помощью команды tar будет запаковывать в архив все файлы в указанной директории. (рис. 3.17)



Рис. 3.17: Написанная программа для lab13\_4.sh

Программа для задания №4:

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

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.18)

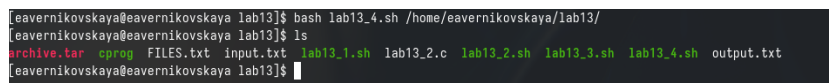


Рис. 3.18: Проверка работы командного файла lab13\_4.sh



## 4 Ответы на контрольные вопросы

### 1. Каково предназначение команды getopt?

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 -infile_in.txt -outfile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае: 

```
while getopts o:i:Ltr optletter do
case
xxxxxxx)xxxxxx = 1; xxxxx =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 man/i.$s`, встреченная в командном файле?

Строка `if test -f man/i.`, ``s/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет 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.

## 5 Выводы

В ходе выполнения лабораторной работы мы изучили основы программирования в оболочке ОС UNIX а также научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 6 Список литературы

Не пользовалась сайтами.