

# **Отчет по лабораторной работе 12**

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

Шалыгин Георгий Эдуардович, НФИбд-02-20

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Техническое обеспечение:</b>	<b>5</b>
<b>3</b>	<b>Условные обозначения и термины:</b>	<b>6</b>
<b>4</b>	<b>Теоретическое введение:</b>	<b>7</b>
<b>5</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
<b>6</b>	<b>Выводы</b>	<b>19</b>
<b>7</b>	<b>Библиография</b>	<b>20</b>

# List of Figures

5.1	Создание нового файла . . . . .	10
5.2	Текст 1 скрипта . . . . .	11
5.3	Результат поиска с ключами . . . . .	12
5.4	Результат поиска с ключами . . . . .	13
5.5	Результат поиска с ключами . . . . .	14
5.6	Текст программы .cpp . . . . .	15
5.7	Текст 3 скрипта . . . . .	15
5.8	Результат сравнения аргумента с 0 . . . . .	16
5.9	Текст 3 скрипта . . . . .	16
5.10	Результат создания tmp файлов . . . . .	17
5.11	Текст 4 скрипта . . . . .	17
5.12	Текст модифицированного 4 скрипта . . . . .	18
5.13	Результат создания архива . . . . .	18

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

Объект исследования: система UNIX.

Предмет исследования: программирование в UNIX.

## **2 Техническое обеспечение:**

- Характеристики техники: AMD Ryzen 5 3500U 2.1 GHz, 8 GB оперативной памяти, 50 GB свободного места на жёстком диске;
- ОС Windows 10 Home
- Git 2.31.1
- Google Chrome 91.0.4472.19
- VirtualBox 2.0
- CentOS 7

### 3 Условные обозначения и термины:

**Текстовым редактором**(text editor) называют программу, которая предназначена для редактирования (составления и изменения) файлов, содержащих только текст. [1]

**Командный язык** - это язык, на котором пользователь взаимодействует с системой в интерактивном режиме.

**Командный интерпретатор**, интерпретатор командной строки - компьютерная программа, часть операционной системы, обеспечивающая базовые возможности управления компьютером посредством интерактивного ввода команд через интерфейс командной строки или последовательного исполнения пакетных командных файлов.[3]

Подробнее в [2] и [3].

## 4 Теоретическое введение:

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая Подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже. [3]

## **Переменные в языке программирования bash**

Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда `mark=/usr/andy/bin` присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка символов.

Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}`. [3]

### **Использование арифметических вычислений. Операторы `let` и `read`**

Оболочка bash поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (`term`), обычно целочисленный.

Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7.

Команда `read` позволяет читать значения переменных со стандартного ввода:

```
echo "Please enter Month and Day of Birth ?"
```

```
read mon day trash [3]
```

### **Командные файлы и функции**

Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `bash командный_файл [аргументы]`

Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `chmod +x имя_файла`



### **Передача параметров в командные файлы и специальные переменные**

Пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1`.

В ходе интерпретации файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`.

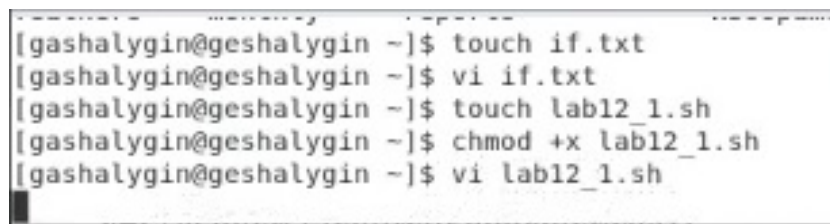
Команда `shift` позволяет удалять первый параметр и сдвигает все остальные на места предыдущих. При использовании в командном файле комбинации символов `$#` вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение.

### **Оператор цикла `for`**

В обобщённой форме оператор цикла `for` выглядит следующим образом: `for имя [in список-значений] do список-команд done`. [3]

## 5 Выполнение лабораторной работы

1. Создадим новый файл lab12.sh. И текстовый файл if.txt с текстом стихотворения architecture И. Бродского (рис. 5.1)



```
[gashalygin@geshalygin ~]$ touch if.txt
[gashalygin@geshalygin ~]$ vi if.txt
[gashalygin@geshalygin ~]$ touch lab12_1.sh
[gashalygin@geshalygin ~]$ chmod +x lab12_1.sh
[gashalygin@geshalygin ~]$ vi lab12_1.sh
```

Figure 5.1: Создание нового файла

2. Используя команды getoptс grep, напишем командный файл, который анализирует командную строку с ключами (рис. 5.2):

- -inputfile — прочитать данные из указанного файла;
- -outputfile — вывести данные в указанный файл;
- -ршаблон — указать шаблон для поиска;
- -C — различать большие и малые буквы;
- -n — выдавать номера строк.

```
gashalygin@gashalygin:~  
Файл Правка Вид Поиск Терминал Справка  
#!/bin/bash  
read=\*;find=\*;of=0;nf=0;cf=0;if=0;pf=0;  
while getopts i:o:p:Cn com  
do case $com in  
  i) if=1; read=$OPTARG;;  
  o) of=1; write=$OPTARG;;  
  p) pf=1; find=$OPTARG;;  
  C) cf=1;;  
  n) nf=1;;  
esac  
done  
if (let $if==1)&&(let $pf==1)  
then if (let $of==1)  
then if (let $cf==0)  
then if (let $nf==0)  
then grep -e $find -i $read >$write  
else grep -e$find -i -n $read >$write  
fi  
else if (let $nf==0)  
then grep -e $find $read >$write  
else grep -e $find -n $read >$write  
fi  
fi  
else if (let $cf==0)  
then if (let $nf==0)  
then grep -e ${find} -i ${read}  
else grep -e ${find} -i -n ${read}  
fi  
else if (let $nf==0)  
then grep -e ${find} ${read}  
else grep -e ${find} -n ${read}  
fi  
fi  
fi  
else echo "unkn. comm."  
fi  
fi  
fi  
fi  
fi
```

Figure 5.2: Текст 1 скрипта

3. Результат работы скрипта, он выводит строки по запросу поиска и ключам.  
(рис. 5.3 5.4 5.5).

```
[gashalygin@geshalygin ~]$ ./lab12_1.sh
unkn. comm.
[gashalygin@geshalygin ~]$ ./lab12_1.sh -p "you"
unkn. comm.
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "you"
only you alone, architecture,
You, in fact, then, what nature
and that - you.
You - vacuum Empress.
Faceted your crusting
in your hand crystal sparkles,
you - the next movement,
you - for more sverhpernatyh
Time for your temple, your rubbish
you brain pylish
You, roughly speaking, sated
and vice versa for you,
yours, architecture, ovary,
your loss, at least,
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "you" -o out.txt
[gashalygin@geshalygin ~]$ cat out.txt
only you alone, architecture,
You, in fact, then, what nature
and that - you.
You - vacuum Empress.
Faceted your crusting
in your hand crystal sparkles,
you - the next movement,
you - for more sverhpernatyh
Time for your temple, your rubbish
you brain pylish
You, roughly speaking, sated
and vice versa for you,
yours, architecture, ovary,
your loss, at least,
[gashalygin@geshalygin ~]$ █
```

Figure 5.3: Результат поиска с ключами

```

you - the next movement,
you - for more sverhpernatyh
Time for your temple, your rubbish
you brain pylish
You, roughly speaking, sated
and vice versa for you,
yours, architecture, ovary,
your loss, at least,
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "you" -C
only you alone, architecture,
and that - you.
Faceted your crusting
in your hand crystal sparkles,
you - the next movement,
you - for more sverhpernatyh
Time for your temple, your rubbish
you brain pylish
and vice versa for you,
yours, architecture, ovary,
your loss, at least,
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -C
You, in fact, then, what nature
You - vacuum Empress.
You, roughly speaking, sated
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -C -n
21:You, in fact, then, what nature
30:You - vacuum Empress.
75:You, roughly speaking, sated
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -n
12:only you alone, architecture,
21:You, in fact, then, what nature
28:and that - you.
30:You - vacuum Empress.
31:Faceted your crusting
32:in your hand crystal sparkles,
41:you - the next movement,
50:you - for more sverhpernatyh
62:Time for your temple, your rubbish
69:you brain pylish
75:You, roughly speaking, sated
85:and vice versa for you,
86:yours, architecture, ovary,
90:your loss, at least,
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -C -n

```

Figure 5.4: Результат поиска с ключами

```

and vice versa for you,
yours, architecture, ovary,
your loss, at least,
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "you" -C
only you alone, architecture,
and that - you.
Faceted your crusting
in your hand crystal sparkles,
you - the next movement,
you - for more sverhpernatyh
Time for your temple, your rubbish
you brain pylish
and vice versa for you,
yours, architecture, ovary,
your loss, at least,
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -C
You, in fact, then, what nature
You - vacuum Empress.
You, roughly speaking, sated
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -C -n
21:You, in fact, then, what nature
30:You - vacuum Empress.
75:You, roughly speaking, sated
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -n
12:only you alone, architecture,
21:You, in fact, then, what nature
28:and that - you.
30:You - vacuum Empress.
31:Faceted your crusting
32:in your hand crystal sparkles,
41:you - the next movement,
50:you - for more sverhpernatyh
62:Time for your temple, your rubbish
69:you brain pylish
75:You, roughly speaking, sated
85:and vice versa for you,
86:yours, architecture, ovary,
90:your loss, at least,
[gashalygin@geshalygin ~]$ ./lab12_1.sh -i if.txt -p "You" -o out.txt -C -n
[gashalygin@geshalygin ~]$ cat out.txt
21:You, in fact, then, what nature
30:You - vacuum Empress.
75:You, roughly speaking, sated
[gashalygin@geshalygin ~]$ █

```

Figure 5.5: Результат поиска с ключами

4. Напишем 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. (рис. 5.6)

```

#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
    int num;
    cout << "enter num\n";
    cin >> num;;
    if (num == 0)
        return 0;
    if (num>0) exit(1);
    if (num < 0) exit(2);
}

```

Figure 5.6: Текст программы .cpp

5. Командный файл вызывает эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено. (рис. 5.7)

```

Файл  Правка  Вид  Поиск  Терминал  Спра
#!/bin/bash
./12
case $? in
0) echo "its 0";;
1) echo ">0";;
2) echo "<0";;
*) echo "error";;
e

```

Figure 5.7: Текст 3 скрипта

6. Результат работы (рис. 5.8).

```

[gashalygin@geshalygin ~]$ g++ 12.cpp -o 12
[gashalygin@geshalygin ~]$ ./lab12_2.sh 10
enter num
10
>0
[gashalygin@geshalygin ~]$ ./lab12_2.sh 10
enter num
0
its 0
[gashalygin@geshalygin ~]$ ./lab12_2.sh 10
enter num
-2
<0

```

Figure 5.8: Результат сравнения аргумента с 0

7. Напишем командный файл, создающий N 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. 5.9)

```

#!/bin/bash
typeset -i count=0;
for f in *.tmp
do rm $f
done
for ((count=1; count<=$1+1; count++))
do
    touch $count.tmp
done

```

Figure 5.9: Текст 3 скрипта

8. Результат работы скрипта, сначала созданы 3 файла, затем - один, предыдущие удалены. (рис. 5.10)



```

[gasnalygin@gesnalygin ~]$ touch lab12.3.sh
[gashalygin@geshalygin ~]$ chmod +s lab12.3.sh
[gashalygin@geshalygin ~]$
[gashalygin@geshalygin ~]$ chmod +x lab12.3.sh
[gashalygin@geshalygin ~]$ vi lab12.3.sh
[gashalygin@geshalygin ~]$ lab12.3.sh 3
bash: lab12.3.sh: команда не найдена...
[gashalygin@geshalygin ~]$ ./lab12.3.sh 3
rm: невозможно удалить «*.tmp»: Нет такого файла или каталога
./lab12.3.sh: line 6: syntax error near unexpected token '('
./lab12.3.sh: line 6: `for(count=1;count<${1+1};count+)'
[gashalygin@geshalygin ~]$ vi lab12.3.sh
[gashalygin@geshalygin ~]$ ./lab12.3.sh 3
rm: невозможно удалить «*.tmp»: Нет такого файла или каталога
./lab12.3.sh: line 6: syntax error near unexpected token '('
./lab12.3.sh: line 6: `for(count=1;count<${1+1};count+)'
[gashalygin@geshalygin ~]$ vi lab12.3.sh
[gashalygin@geshalygin ~]$ ./lab12.3.sh 3
rm: невозможно удалить «*.tmp»: Нет такого файла или каталога
./lab12.3.sh: line 6: ((: count+: ошибка синтаксиса: ожидается операнд (error token is "+")
[gashalygin@geshalygin ~]$ vi lab12.3.sh
[gashalygin@geshalygin ~]$ ./lab12.3.sh 3
[gashalygin@geshalygin ~]$ ls | grep .tmp
1.tmp
2.tmp
3.tmp
[gashalygin@geshalygin ~]$ ./lab12.3.sh 1
[gashalygin@geshalygin ~]$ ls | grep .tmp
1.tmp
[gashalygin@geshalygin ~]$

```

Figure 5.10: Результат создания tmp файлов

9. Напишем командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории (рис. 5.11). Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). (рис. 5.12)

```

#!/bin/bash
cd $1
tar -cf arch.tar *

```

Figure 5.11: Текст 4 скрипта

```
Файл  Правка  Вид  Поиск  Терминал  Справка
#!/bin/bash
cd $1
tar -cf arch.tar $(find -type f -mtime -7)
```

Figure 5.12: Текст модифицированного 4 скрипта

10. Результат работы скрипта, создан arch.tar. (рис. 5.13)

```
[gashalygin@geshalygin ~]$ ./lab12_4.sh backup
[gashalygin@geshalygin ~]$ ls
12                backup      if.txt          lab12_3.sh      out.txt          ski.places      Изображения
12.cpp            conf.txt      lab11_3.sh      lab12_4.sh      pandoc-crossref test.txt        Музыка
l.tmp             equiplist    lab11_4.sh      may             pandoc-crossref.1 works          Общедоступные
abc1              feathers     lab11.sh        monthly         play             Видео          Рабочий стол
australia         file         lab12_1.sh      my_os           r               Документы     Шаблоны
australia, file.txt lab12_2.sh    my_os,         reports         Загрузки
```

```
[gashalygin@geshalygin ~]$ cd backup/
[gashalygin@geshalygin backup]$ ls
arch.tar  backup.sh.gz  lab11_2.sh  lab11_3.sh
[gashalygin@geshalygin backup]$
```

Figure 5.13: Результат создания архива

## **6 Выводы**

В процессе работы над лабораторной работы были изучены основы программирования в оболочке ОС UNIX, получен опыт написания более сложных командных файлов с использованием логических управляющих конструкций и циклов.

## 7 Библиография

1. <https://docs.altlinux.org/ru-RU/archive/2.3/html-single/junior/alt-docs-extras-linuxnovice/ch02s10.html>
2. <http://bourabai.kz/os/shells.htm>
3. Д.С. Кулябов, А.В. Королькова / Администрирование локальных систем. Лабораторные работы. — М.: Российский университет дружбы народов, 2017. — 119 с.