

Лабораторная работа №10. Программирование в командном процессоре ОС UNIX. Командные файлы

Подготовил:

Королев Адам Маратович

Группа: НПИбд-02-21

Студенческий билет: № 1032217060

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

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

- С-оболочка (или `csh`) - надстройка на оболочке Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд
- оболочка Корна (или `ksh`) - напоминает оболочку С, 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.

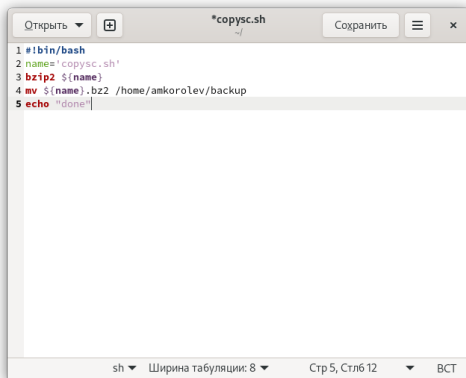
POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке `bash`. В других оболочках большинство команд будет совпадать с описанными ниже.

Выполнение лабораторной работы:

1. Написать скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в вашем домашнем каталоге.

При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.

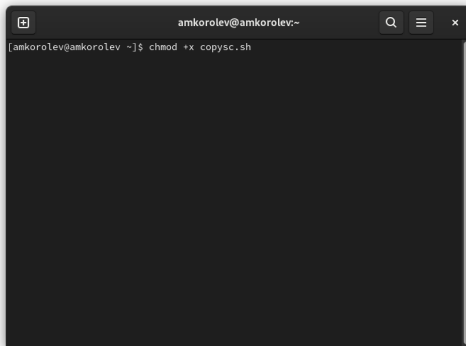
Пишем скрипт.



```
1 #!/bin/bash
2 name='copysc.sh'
3 bzip2 ${name}
4 mv ${name}.bz2 /home/amkorolev/backup
5 echo "done"
```

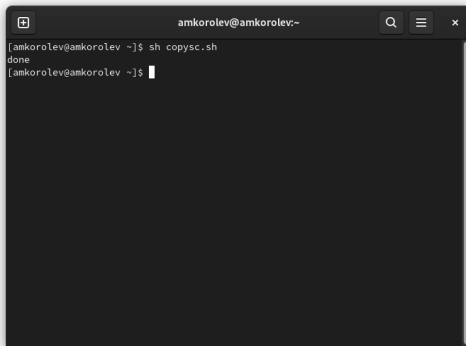
Figure 1: Пишем скрипт

Даем права на исполнение.

A terminal window with a dark background. The title bar shows 'amkorolev@amkorolev:~'. The command prompt is '[amkorolev@amkorolev ~]\$' and the command entered is 'chmod +x copysc.sh'.

```
amkorolev@amkorolev:~  
[amkorolev@amkorolev ~]$ chmod +x copysc.sh
```

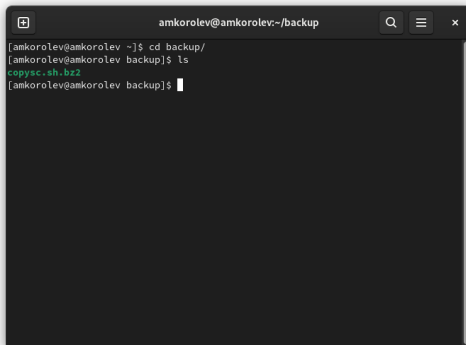
Figure 2: Даем права на исполнение

A terminal window with a dark background and light text. The title bar at the top reads 'amkorolev@amkorolev:~'. Below the title bar, the terminal shows the command '[amkorolev@amkorolev ~]\$ sh copysc.sh' followed by the output 'done' on the next line. The prompt '[amkorolev@amkorolev ~]\$' is followed by a cursor. The window has standard macOS-style window controls (red, yellow, green buttons) on the left and search, menu, and close buttons on the right.

```
amkorolev@amkorolev:~  
[amkorolev@amkorolev ~]$ sh copysc.sh  
done  
[amkorolev@amkorolev ~]$
```

Figure 3: Выполняем скрипт

Проверяем результат



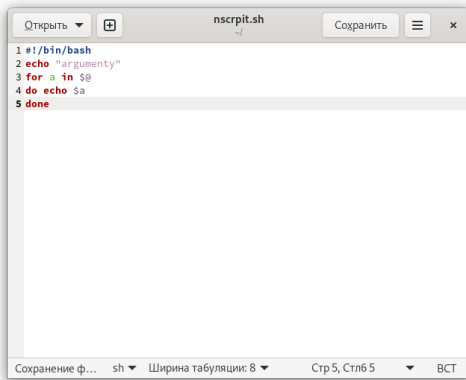
```
amkorolev@amkorolev:~/backup
[amkorolev@amkorolev ~]$ cd backup/
[amkorolev@amkorolev backup]$ ls
copysc.sh.bz2
[amkorolev@amkorolev backup]$
```

Figure 4: Проверяем результат

2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять.

Например, скрипт может
последовательно распечатывать
значения всех переданных
аргументов.

Пишем скрипт.

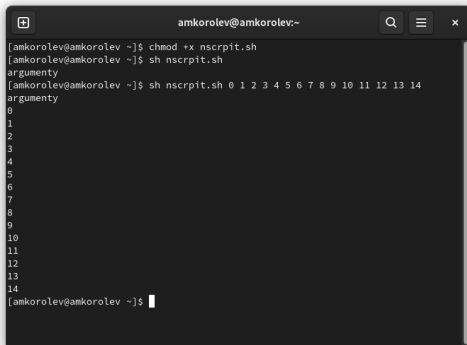


```
1 #!/bin/bash
2 echo "argumenty"
3 for a in $@
4 do echo $a
5 done
```

Сохранение ф... sh Ширина табуляции: 8 Стр 5, Стлб 5 ВСТ

Figure 5: Пишем скрипт

Даем права на исполнение и выполняем скрипт.

A terminal window titled 'amkorolev@amkorolev:~' with search, menu, and close buttons. It shows the following commands and output:

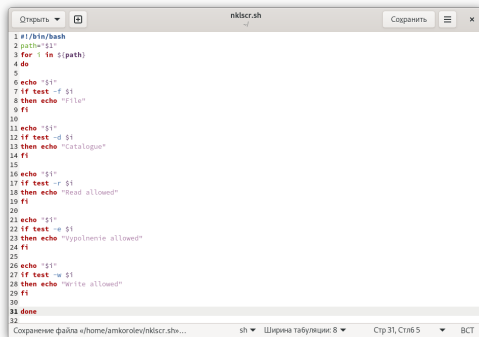
```
[amkorolev@amkorolev ~]$ chmod +x nscrpit.sh
[amkorolev@amkorolev ~]$ sh nscrpit.sh
argumenty
[amkorolev@amkorolev ~]$ sh nscrpit.sh 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
argumenty
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
[amkorolev@amkorolev ~]$
```

Figure 6: Даем права на исполнение и выполняем скрипт

3. Написать командный файл -
аналог команды ls.

3. Написать командный файл - аналог команды ls.

#Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

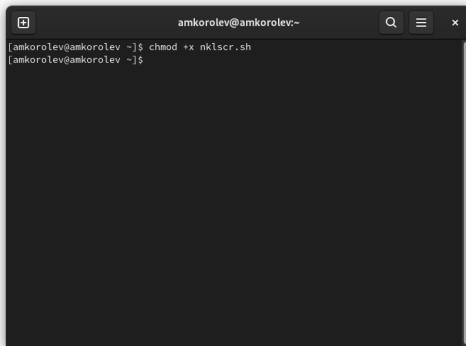


```
1 #!/bin/bash
2 path="$1"
3 for i in $(path)
4 do
5
6 echo "$i"
7 if test -f $i
8 then echo "File"
9 fi
10
11 echo "$i"
12 if test -d $i
13 then echo "Catalogue"
14 fi
15
16 echo "$i"
17 if test -r $i
18 then echo "Read allowed"
19 fi
20
21 echo "$i"
22 if test -w $i
23 then echo "Выполнение allowed"
24 fi
25
26 echo "$i"
27 if test -w $i
28 then echo "Write allowed"
29 fi
30
31 done
32
```

Сохранение файла «/home/amirolev/nklsr.shv... sh Ширина табуляции: 8 Стр 31, Стлб 5 ВСТ

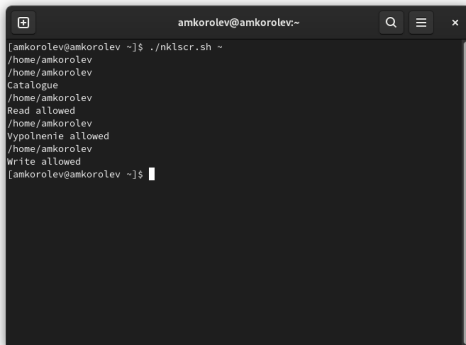
Figure 7: Пишем скрипт

Даем права на исполнение.

A terminal window with a dark background and light text. The title bar at the top reads 'amkorolev@amkorolev:~'. Below the title bar, the command '[amkorolev@amkorolev ~]\$ chmod +x nklscr.sh' has been entered and executed. The prompt '[amkorolev@amkorolev ~]\$' is visible on the line below.

```
amkorolev@amkorolev:~  
[amkorolev@amkorolev ~]$ chmod +x nklscr.sh  
[amkorolev@amkorolev ~]$
```

Figure 8: Даем права на исполнение

A terminal window titled 'amkorolev@amkorolev:~' with search, menu, and close buttons. It shows the execution of a script './nklscr.sh ~' which outputs several paths and permissions.

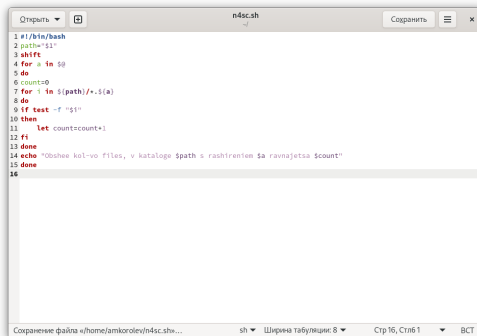
```
[amkorolev@amkorolev ~]$ ./nklscr.sh ~  
/home/amkorolev  
/home/amkorolev  
Catalogue  
/home/amkorolev  
Read allowed  
/home/amkorolev  
Vypolnenie allowed  
/home/amkorolev  
Write allowed  
[amkorolev@amkorolev ~]$
```

Figure 9: Выполняем скрипт

4. Написать командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории.

Путь к директории также
передаётся в виде аргумента
командной строки.

Пишем скрипт.

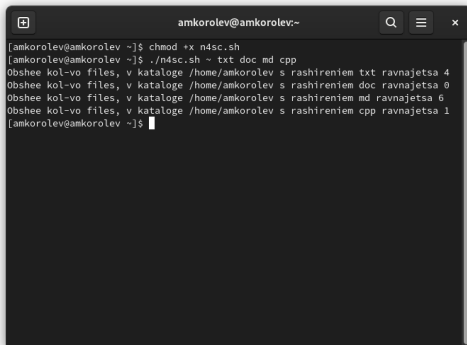


```
1 #!/bin/bash
2 path="$1"
3 shift
4 for a in $(
5 do
6 count=0
7 for i in $(path)/.*.$(a)
8 do
9 if test -f "$i"
10 then
11     let count=count+1
12 fi
13 done
14 echo "Общее кол-во files, в каталоге $path с расширением $a равно $count"
15 done
16
```

Сохранение файла «/home/amkorolev/n4sc.sh»... sh Ширина таблицы: 8 Стр 16, столб 1 ВСТ

Figure 10: Пишем скрипт

Даем права на исполнение и выполняем скрипт.

A terminal window titled 'amkorolev@amkorolev:~' with search, menu, and close icons. It shows the execution of a script 'n4sc.sh' with permissions. The output of the script is displayed in the terminal.

```
[amkorolev@amkorolev ~]$ chmod +x n4sc.sh
[amkorolev@amkorolev ~]$ ./n4sc.sh ~ txt doc md cpp
Obshee kol-vo files, v kataloge /home/amkorolev s rashireniem txt ravnajetsa 4
Obshee kol-vo files, v kataloge /home/amkorolev s rashireniem doc ravnajetsa 0
Obshee kol-vo files, v kataloge /home/amkorolev s rashireniem md ravnajetsa 6
Obshee kol-vo files, v kataloge /home/amkorolev s rashireniem cpp ravnajetsa 1
[amkorolev@amkorolev ~]$
```

Figure 11: Даем права на исполнение и выполняем скрипт

Выводы:

- В процессе выполнения работы изучил основы программирования в оболочке ОС UNIX/Linux. Научился писать небольшие командные файлы.