

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
“РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ”

Факультет физико-математических и естественных наук

ОТЧЕТ

По лабораторной работе №10  
“Программирование в командном процессоре ОС UNIX. Командные  
файлы”

Выполнил:  
Студент группы: НПИбд-02-21  
Студенческий билет: №1032217060  
ФИО студента: Королев Адам Маратович  
Дата выполнения: 21.05.2022

Москва 2022

## **1 Цель работы:**

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

## 2 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;
- С-оболочка (или csh) — надстройка на оболочкой 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 С 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.

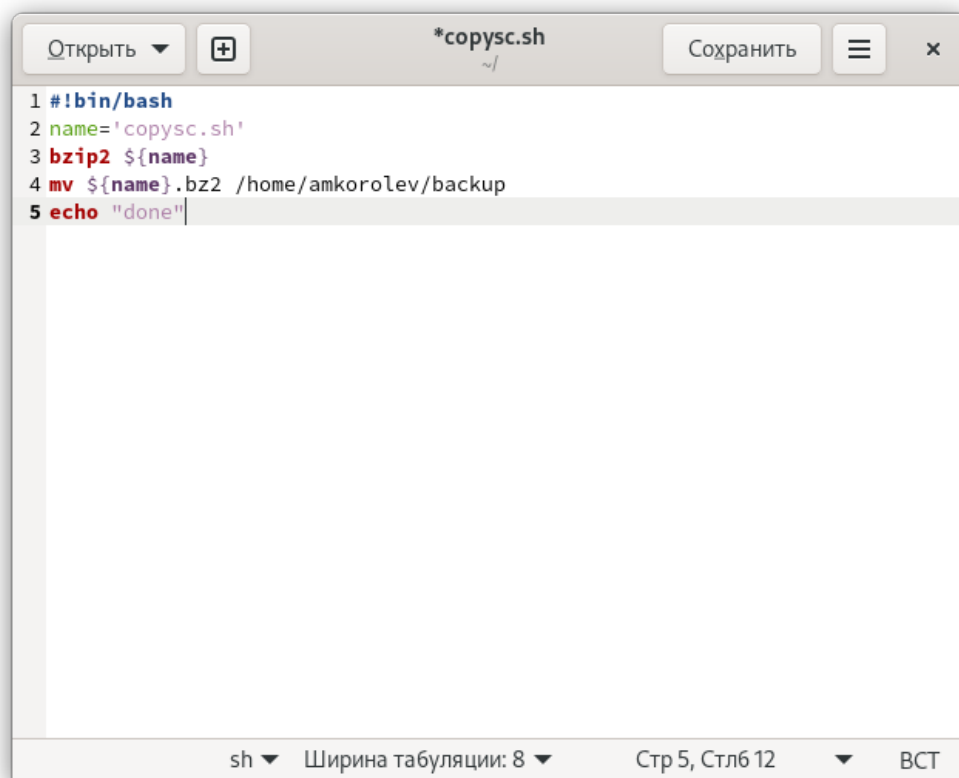
Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода.

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 Выполнение лабораторной работы:

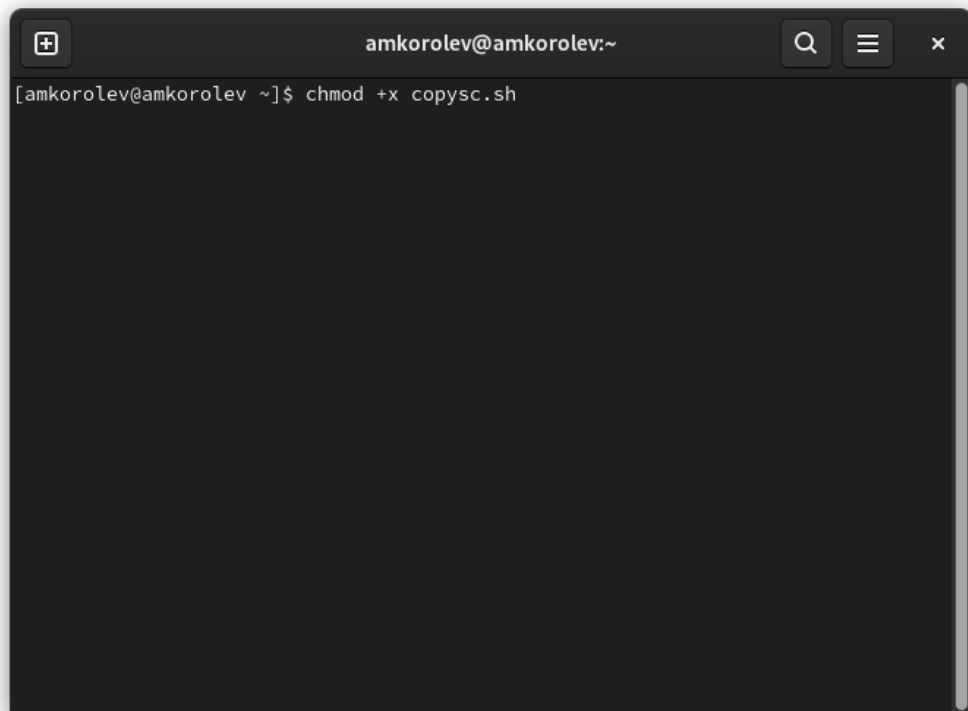
1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.

Пишем скрипт.

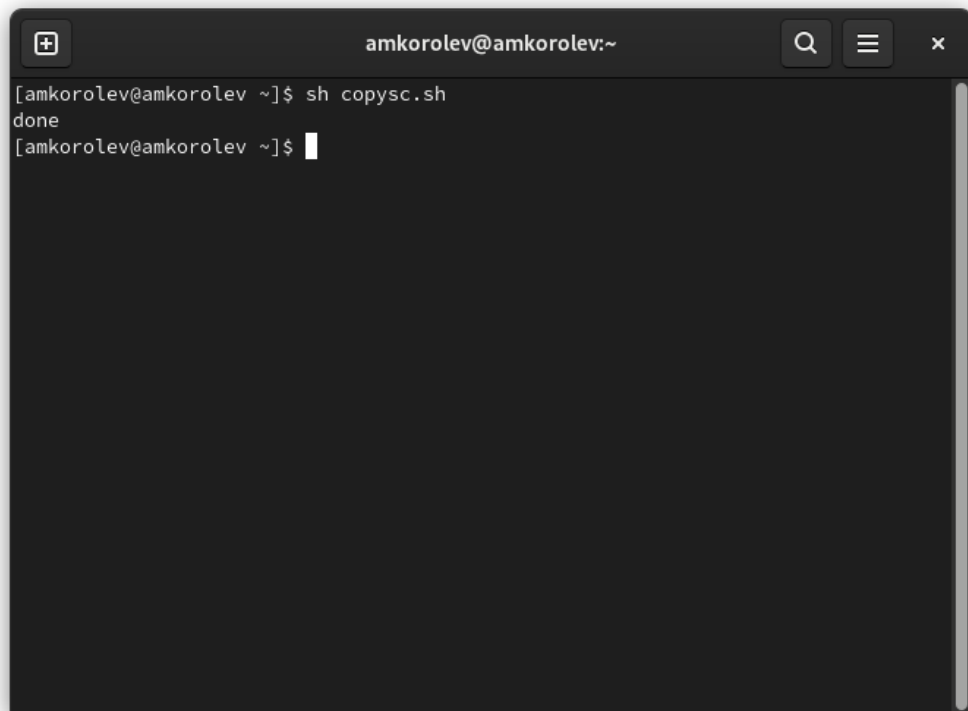


```
1 #!/bin/bash
2 name='copysc.sh'
3 bzip2 ${name}
4 mv ${name}.bz2 /home/amkorolev/backup
5 echo "done"
```

Даем права на исполнение.

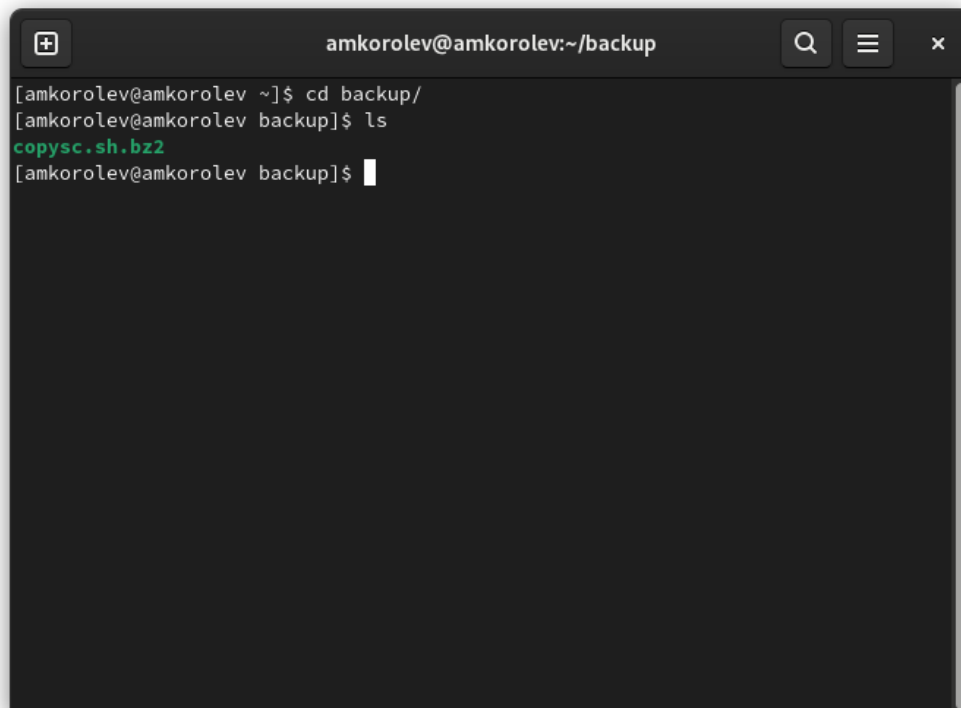


Выполняем скрипт

A terminal window with a dark background and light text. The title bar at the top reads 'amkorolev@amkorolev:~'. On the left of the title bar is a square button with a plus sign, and on the right are three buttons: a magnifying glass (search), a hamburger menu (three horizontal lines), and a close button (an 'x'). The terminal content shows the prompt '[amkorolev@amkorolev ~]\$' followed by the command 'sh copysc.sh'. The next line shows the output 'done'. The prompt '[amkorolev@amkorolev ~]\$' appears again, followed by a white cursor block. A vertical scrollbar is visible on the right side of the terminal window.

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

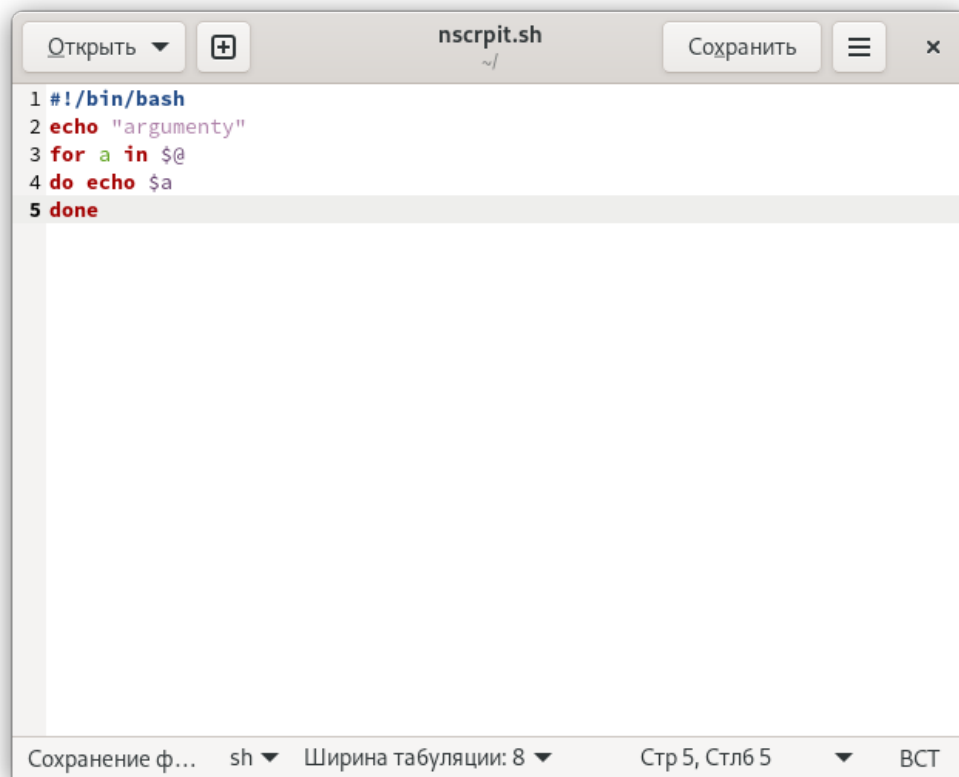
Проверяем результат

A terminal window with a dark background. The title bar shows 'amkorolev@amkorolev:~/backup'. The terminal content shows the user navigating to the 'backup' directory and running a script named 'copysc.sh.bz2'.

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

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.

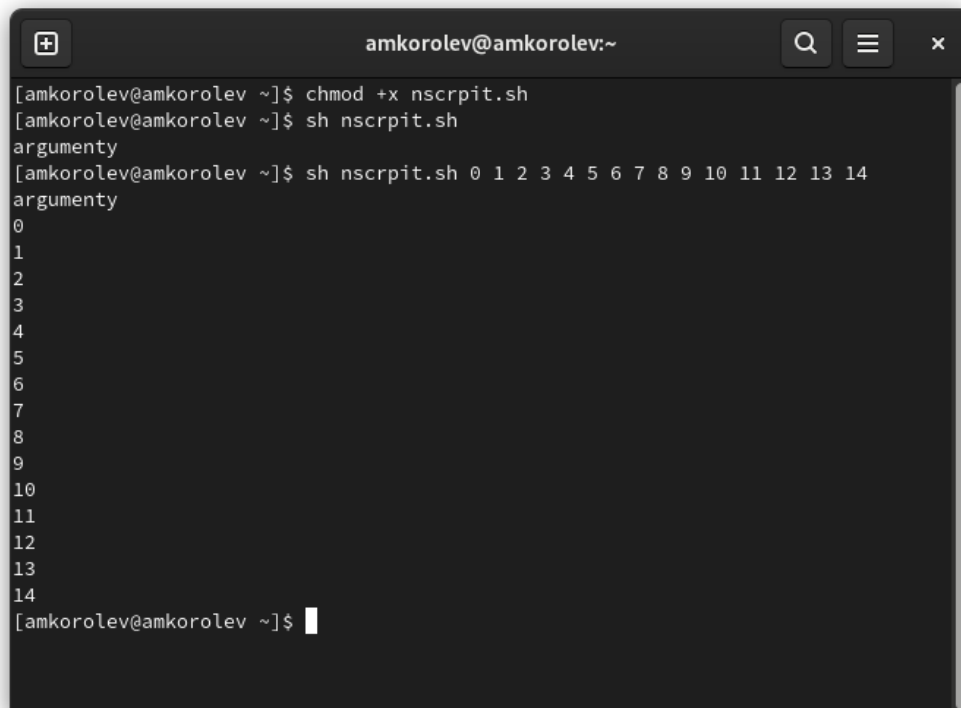
Пишем скрипт.



```
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

Даем права на исполнение и выполняем скрипт.

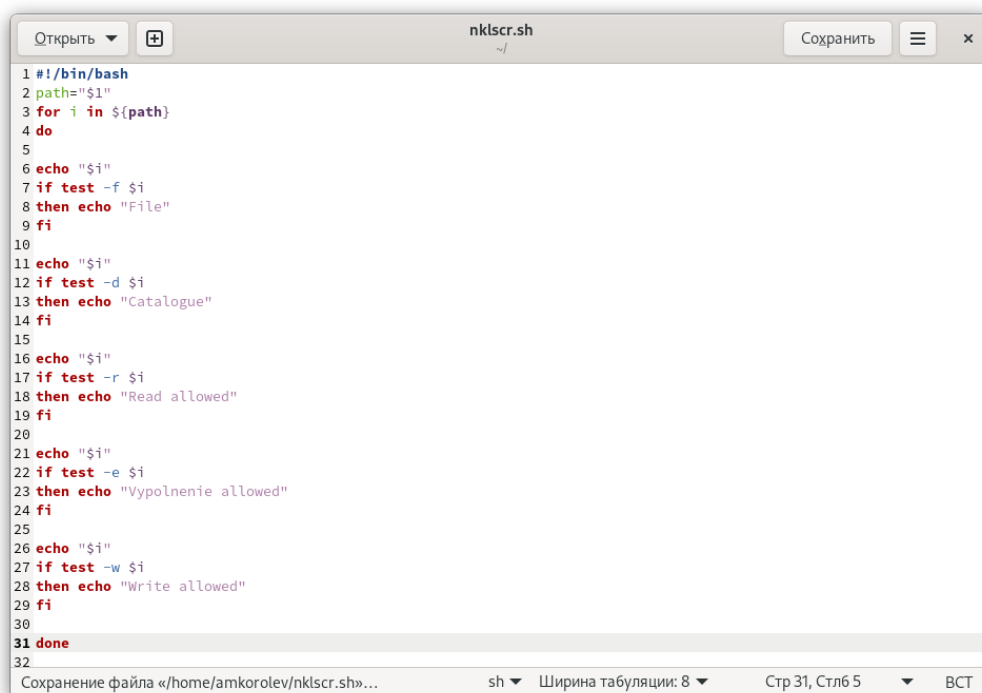
A terminal window titled 'amkorolev@amkorolev:~' with search, menu, and close buttons. It shows the execution of a script named 'nscrpit.sh'. The user first runs 'chmod +x nscrpit.sh' and then 'sh nscrpit.sh'. The script prints 'argumenty' and then a list of numbers from 0 to 14. The prompt returns to the user.

```
[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 ~]$
```

3. Написать командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

Пишем скрипт.

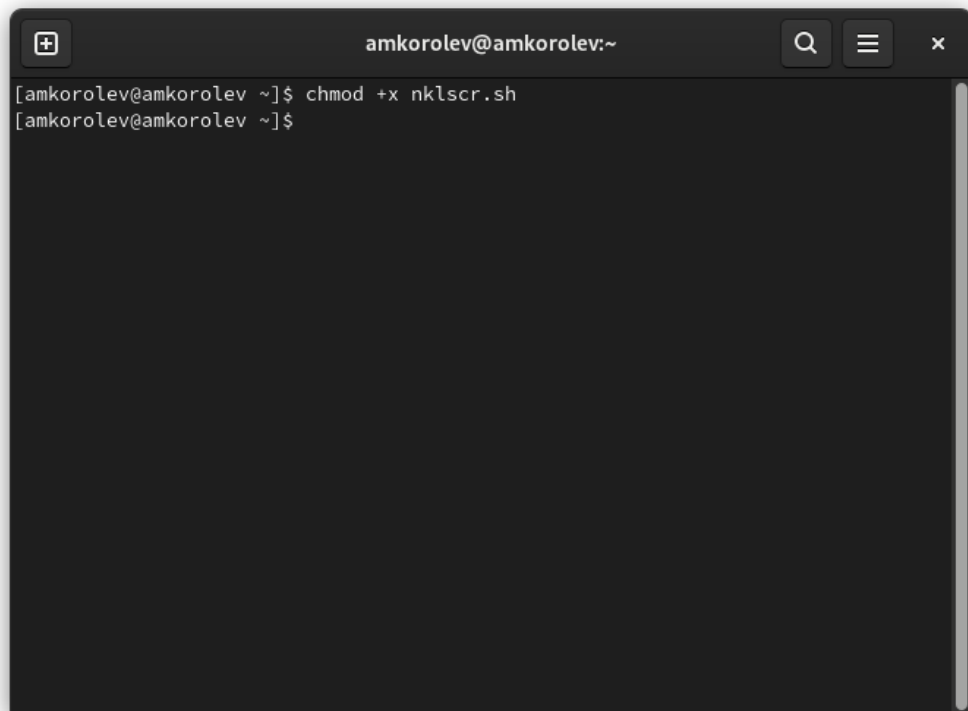




```
1 #!/bin/bash
2 path="$1"
3 for i in $(path)
4 do
5
6 echo "$i"
7 if test -f $i
8 then echo "File"
9 fi
10
11 echo "$i"
12 if test -d $i
13 then echo "Catalogue"
14 fi
15
16 echo "$i"
17 if test -r $i
18 then echo "Read allowed"
19 fi
20
21 echo "$i"
22 if test -e $i
23 then echo "Vypolnenie allowed"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

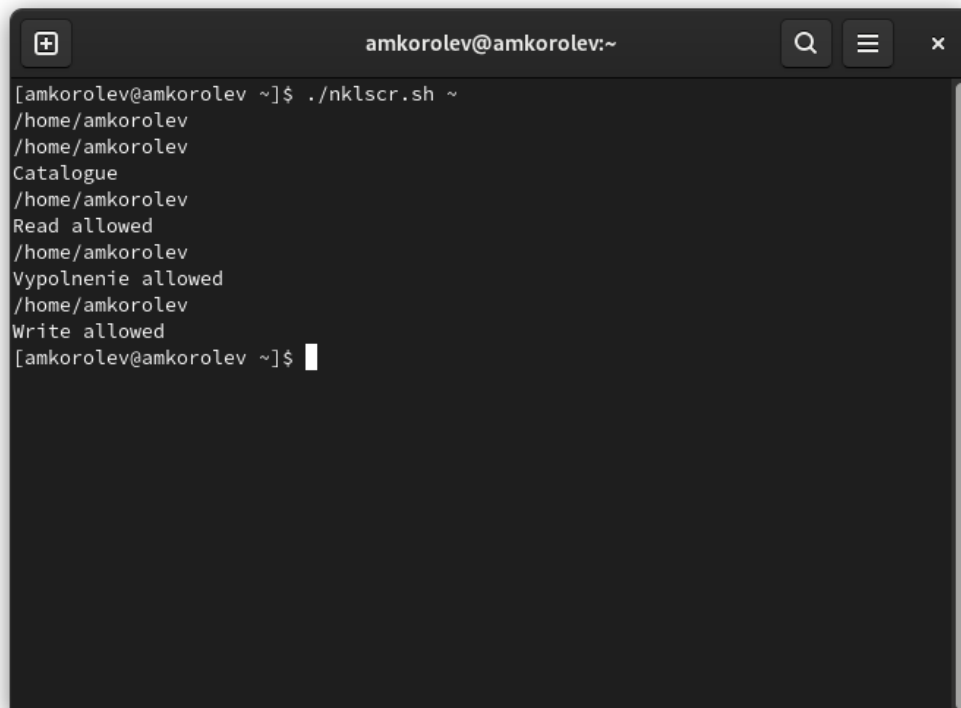
Сохранение файла «/home/amkorolev/nklscr.sh»... sh Ширина табуляции: 8 Стр 31, Стлб 5 ВСТ

Даем права на исполнение.

A terminal window with a dark background and light gray text. The window title bar shows 'amkorolev@amkorolev:~' and standard window controls. The terminal content shows the command 'chmod +x nklscr.sh' being executed successfully, with the prompt returning to the user.

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

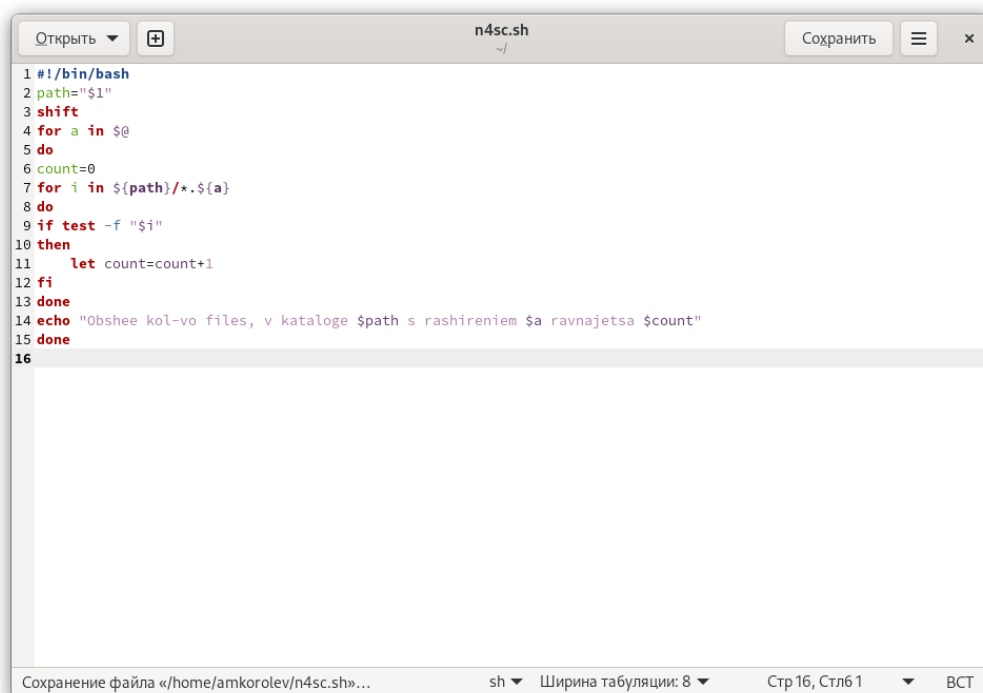
Выполняем скрипт

A terminal window titled 'amkorolev@amkorolev:~' with search, menu, and close buttons. It shows the execution of a script './nklscr.sh ~'. The script outputs the directory path '/home/amkorolev' three times, followed by 'Catalogue', and then permissions: 'Read allowed', 'Vypolnenie allowed', and 'Write allowed'. The prompt returns to the user.

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

4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

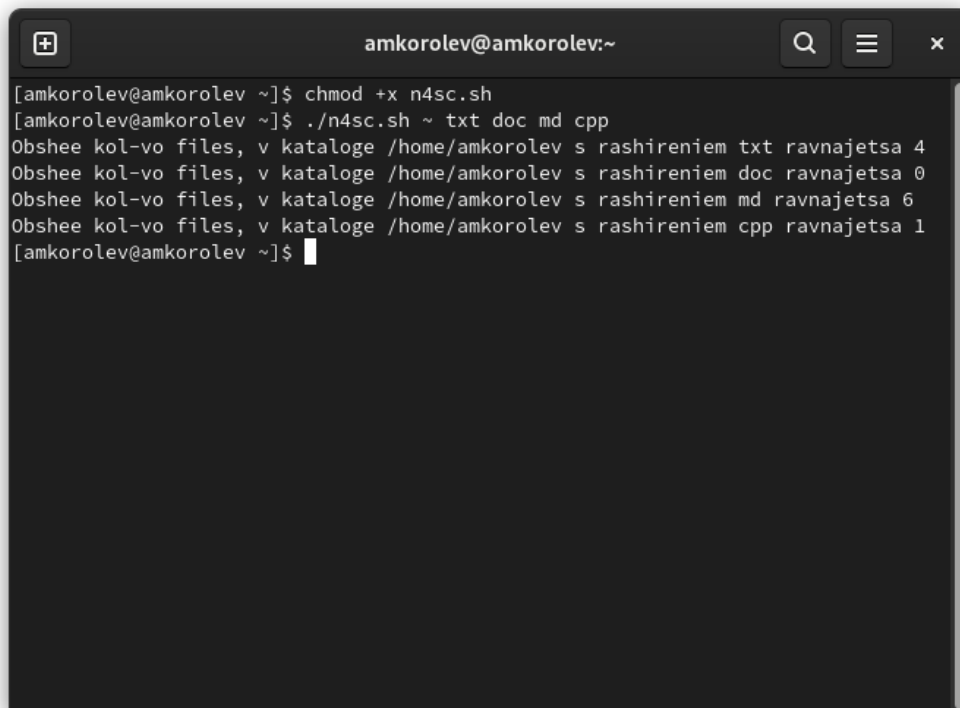
Пишем скрипт.



```
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 "Obshee kol-vo files, v kataloge $path s rashireniem $a ravnajetsa $count"
15 done
16
```

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

Даем права на исполнение и выполняем скрипт.

A terminal window titled 'amkorolev@amkorolev:~' with search, menu, and close buttons. It shows the execution of a script 'n4sc.sh' which counts files in 'txt', 'doc', 'md', and 'cpp' categories. The output shows 4 txt files, 0 doc files, 6 md files, and 1 cpp file.

```
[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 ~]$
```

## 4 Выводы:

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

## 5 Ответы на контрольные вопросы:

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

Командная оболочка – это программа, которая принимает ввод от пользователя и выполняет команды. Самая известная оболочка – это `bash`, но существуют и другие, например `zsh`, `csh`, `ksh`, `fish`, `busybox` и т.д. Разные командные оболочки могут различаться по своему синтаксису – так, `csh` имеет все возможности Bourne shell, но они доступны в дру-

гом, более C-подобном синтаксисе. Командные оболочки также могут отличаться по своему пользовательскому интерфейсу – fish имеет функционал вроде автодополнения из man-страниц, который делает ее более удобной для использования.

## 2. Что такое POSIX?

POSIX – стандарт для различных Unix-подобных систем, описывающий разные API и способы взаимодействия с системой. Следование этим стандартам при написании кода гарантирует, что этот код будет работать на любой системе, которая поддерживает этот стандарт.

## 3. Как определяются переменные и массивы в языке программирования bash?

```
VAR="hello world"
echo $var # > hello world
ARRAY=("this" "is" "an" "array")
echo ${ARRAY[3]} # > array
echo ${ARRAY[*]} # > this is an array
```

## 4. Каково назначение операторов let и read?

Команда let позволяет выполнять арифметические операции с переменными.

```
A=123
B=456
let "C = A + B"
echo $C # > 579
```

Команда read считывает ввод с стандартного ввода и присваивает его значение переменной.

```
echo "What is your name?"
read NAME
echo "Hello, $NAME"
```

## 5. Какие арифметические операции можно применять в языке программирования bash?

Самые полезные операции показаны ниже.

```
A=500
B=100
let "q = A + B"; echo $q # > 600
let "q = A - B"; echo $q # > 400
```

```

let "q = A * B"; echo $q # > 50000
let "q = A / B"; echo $q # > 5
let "q = A % B"; echo $q # > 0
A=5
B=2
let "q = A ** B"; echo $q # > 25
let "q = -A"; echo $q # > -5
let "q = A<<B"; echo $q # > 20
let "q = A>>B"; echo $q # > 1

```

#### 6. Что означает операция (( ))?

Эта операция позволяет выполнять арифметические действия, не используя команду let.

```

A=500
B=100
(( A++ ))
echo $(( A+B )) # > 601

```

#### 7. Какие стандартные имена переменных Вам известны?

- \$PATH – список путей, в которых следует искать программы команд
- \$HOME – домашний каталог
- \$SHELL – путь к используемой оболочке
- \$USER – имя пользователя
- \$HOSTNAME – имя хоста
- \$PWD – текущий каталог
- \$OLDPWD – предыдущий каталог
- \$PS1 – приглашение к вводу команды
- \$PS2 – приглашение к вводу продолжающей строки

#### 8. Что такое метасимволы?

Метасимволы – это символы, которые имеют особый смысл в контексте glob-последовательностей, например:

- \* - любое количество (включая 0) любых символов
- ? - любой один символ
- [ . . . ] - любой символ из перечисленных в скобках
- [ a - z ] - любой символ из диапазона букв

#### 9. Как экранировать метасимволы?

Для этого нужно написать перед этим символом обратный слеш: `*.*` соответствует всем файлам, имеющим расширение, а `*.*` соответствует только файлам, расширение которых равно одной звездочке.

10. Как создавать и запускать командные файлы?

Для этого нужно создать текстовый файл, на первой строке написать специальную последовательность `#!/`, а затем путь к интерпретатору (например, `#!/bin/bash`). После этого в файле можно написать команды. Для того, чтобы выполнить этот файл, нужно добавить разрешение на выполнение, используя команду `chmod +x`.

11. Как определяются функции в языке программирования bash?

```
greet () {  
    echo "Hello, $1!"  
}  
greet "Jim"
```

12. Каким образом можно выяснить, является файл каталогом или обычным файлом?

```
$file="/tmp/what"  
[ -d $file ] && echo "file is a directory" || echo "file is a normal"
```

13. Каково назначение команд `set`, `typeset` и `unset`?

Эти команды используются, чтобы управлять переменными – `set` задает новое значение переменной, `unset` удаляет переменную, а `typeset` задает значение переменной и ее тип.

14. Как передаются параметры в командные файлы?

Они оказываются в переменных `$1...$9`, а также в массиве `$@`. Оттуда их можно использовать внутри командного файла.

15. Назовите специальные переменные языка bash и их назначение.

- `$0` - имя исполняемого файла
- `$1` - первый аргумент командной строки
- `$2` - второй аргумент командной строки
- `$9` - девятый аргумент командной строки
- `$#` - количество аргументов командной строки
- `$@` - все аргументы командной строки
- `$$` - номер процесса
- `$?` - код, возвращенный последней выполненной командой