

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №10 =====

дисциплина: Операционные системы

Студент: Койфман Кирилл Дмитриевич

Группа: НПИбд-01-21

### Введение.

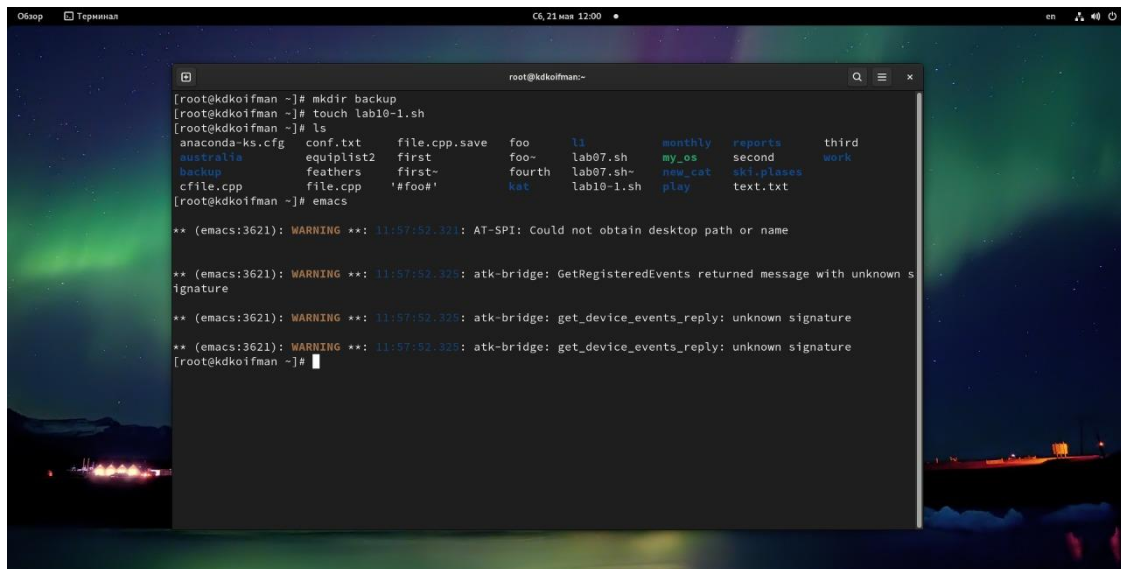
---

### Цель работы.

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

### Задачи.

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.
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.
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.
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. ## Ход работы. ## 1 задание. Напишем 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 (рис.1,2,3).

A terminal window titled "Терминал" with a dark background and a colorful aurora-like wallpaper. The user is at the root prompt of a machine named "kdkoifman". They execute a series of commands: "mkdir backup", "touch lab10-1.sh", and "ls". The "ls" command shows a directory listing with files like "anaconda-ks.cfg", "conf.txt", "file.cpp.save", "foo", "li", "monthly", "reports", "third", "australia", "equiplist2", "first", "foo-", "lab07.sh", "my\_os", "second", "work", "backup", "feathers", "first-", "fourth", "lab07.sh-", "new\_cat", "ski.places", "cfile.cpp", "file.cpp", "'#foo#'", "kat", "lab10-1.sh", "play", and "text.txt". Finally, they run "emacs", which opens the Emacs editor. The Emacs status bar at the bottom shows "root@kdkoifman:~". There are also some warning messages from the system in the background.

```
root@kdkoifman:~# mkdir backup
root@kdkoifman:~# touch lab10-1.sh
root@kdkoifman:~# ls
anaconda-ks.cfg  conf.txt      file.cpp.save  foo    li      monthly  reports  third
australia        equiplist2    first          foo-   lab07.sh my_os    second   work
backup           feathers      first-         fourth lab07.sh- new_cat  ski.places
cfile.cpp        file.cpp      '#foo#'        kat    lab10-1.sh play     text.txt
root@kdkoifman:~# emacs

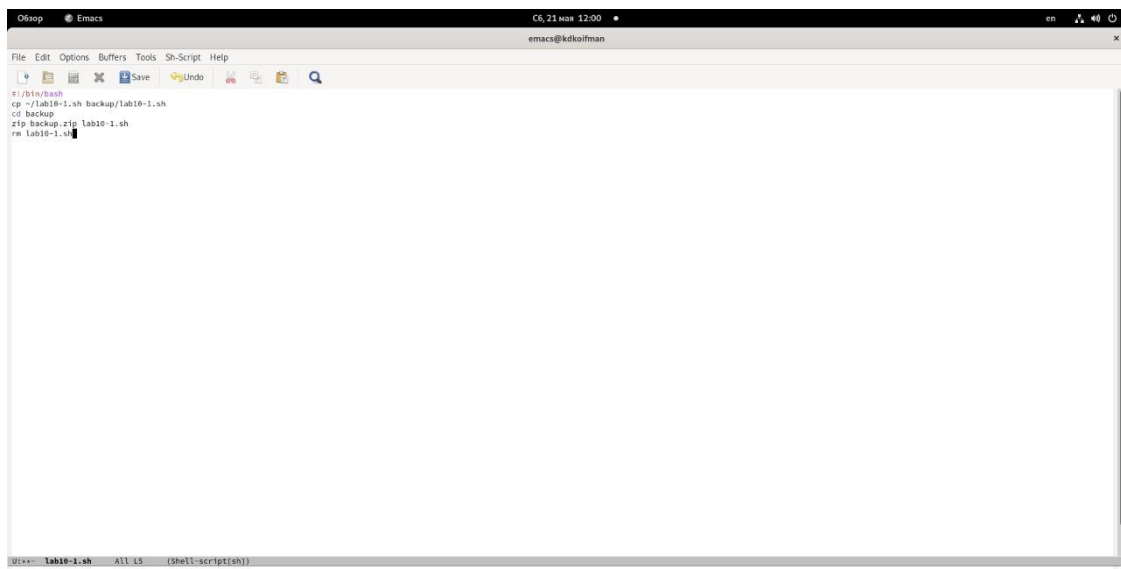
** (emacs:3621): WARNING **: 11:57:52.321: AT-SPI: Could not obtain desktop path or name

** (emacs:3621): WARNING **: 11:57:52.325: atk-bridge: GetRegisteredEvents returned message with unknown signature

** (emacs:3621): WARNING **: 11:57:52.325: atk-bridge: get_device_events_reply: unknown signature

** (emacs:3621): WARNING **: 11:57:52.325: atk-bridge: get_device_events_reply: unknown signature
root@kdkoifman:~#
```

рис.1(создадим директорию backup, файл lab10-1.sh и откроем его в редакторе emacs)

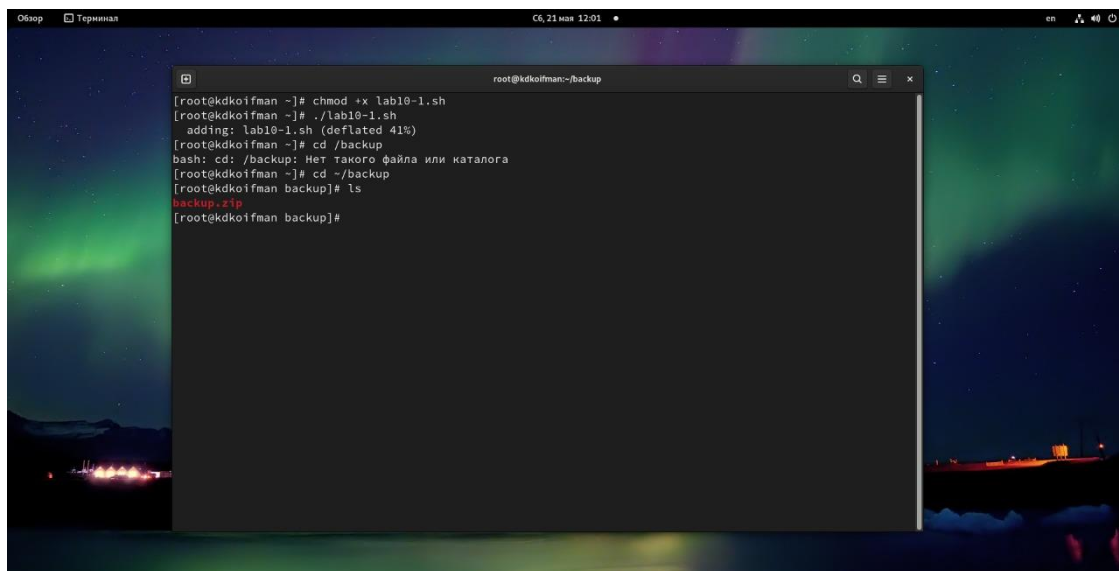
An Emacs editor window titled "Emacs" with a light gray background. The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". The toolbar shows icons for file operations and editing. The main text area contains the following script code:

```
#!/bin/bash
cp ~/lab10-1.sh backup/lab10-1.sh
cd backup
zip backup.zip lab10-1.sh
rm lab10-1.sh
```

The status bar at the bottom shows "lab10-1.sh", "All LS", and "(shell-script/sh)".

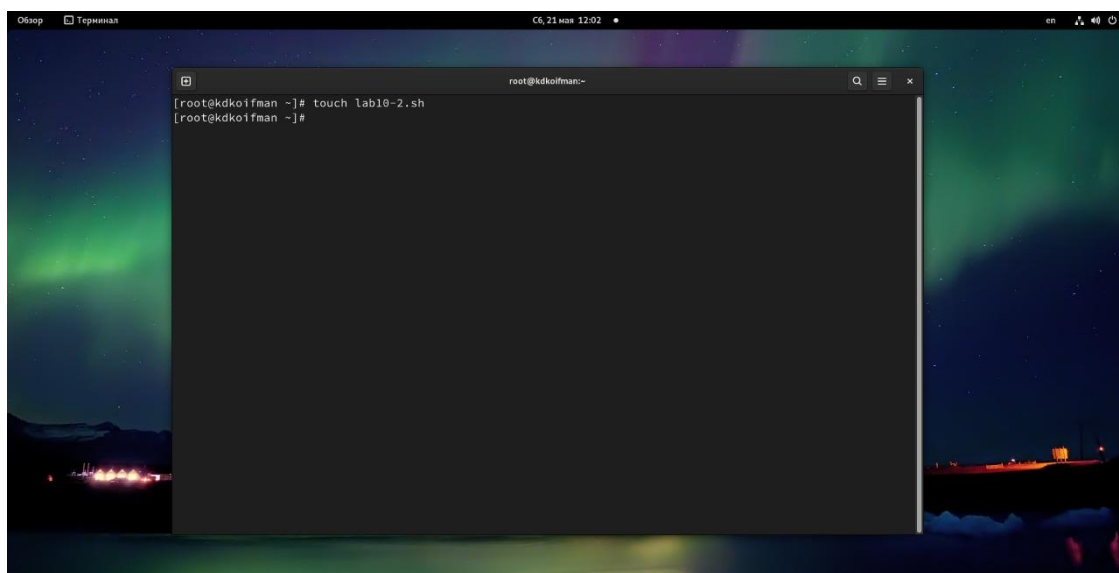
```
#!/bin/bash
cp ~/lab10-1.sh backup/lab10-1.sh
cd backup
zip backup.zip lab10-1.sh
rm lab10-1.sh
```

рис.2(код программы)

A terminal window titled 'root@kdkoifman:~/backup' is shown against a desktop background of a night sky with aurora borealis. The terminal displays the following commands and output:

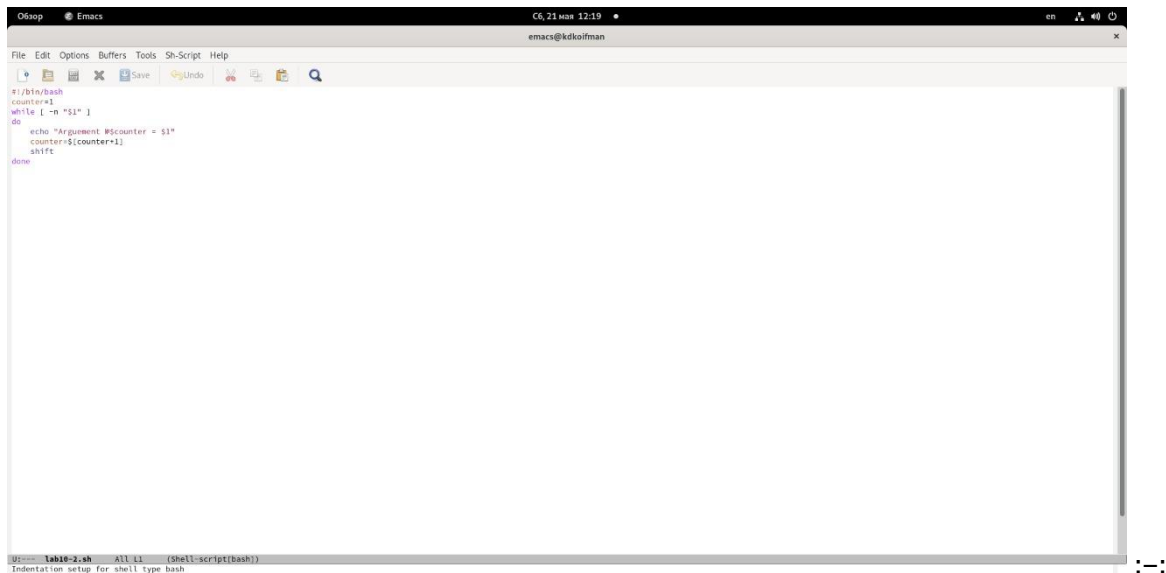
```
[root@kdkoifman ~]# chmod +x lab10-1.sh
[root@kdkoifman ~]# ./lab10-1.sh
adding: lab10-1.sh (deflated 41%)
[root@kdkoifman ~]# cd /backup
bash: cd: /backup: Нет такого файла или каталога
[root@kdkoifman ~]# cd ~/backup
[root@kdkoifman backup]# ls
backup.zip
[root@kdkoifman backup]#
```

рис.3(предоставим всем право на выполнение файла *lab10-1.sh* и выполним его) ## 2 задание. Напишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов. (рис.4,5,6).

A terminal window titled 'root@kdkoifman:~' is shown against the same desktop background. The terminal displays the following commands:

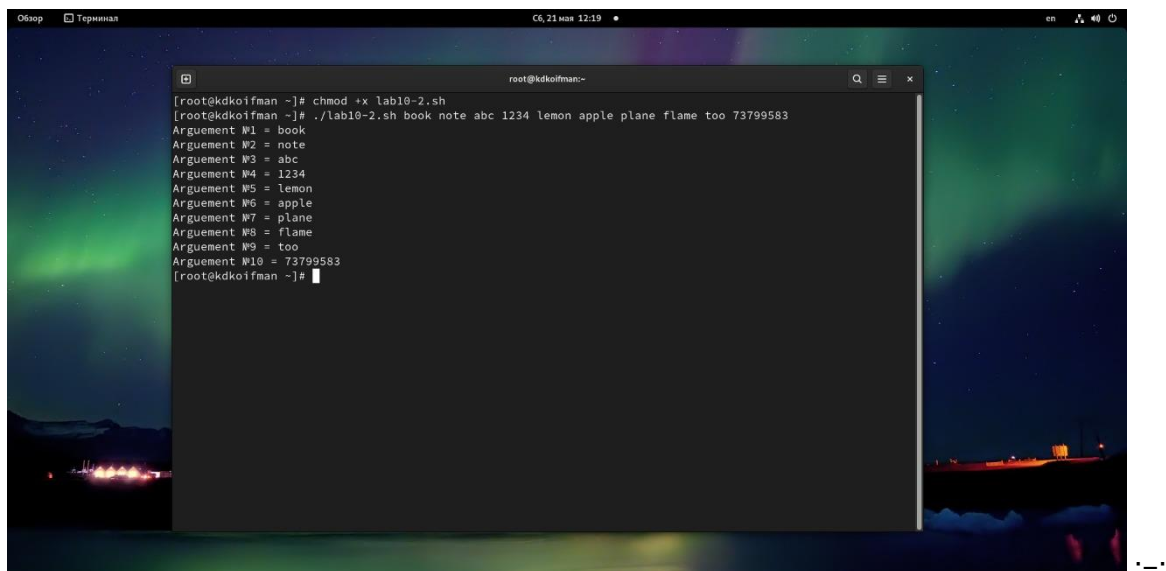
```
[root@kdkoifman ~]# touch lab10-2.sh
[root@kdkoifman ~]#
```

рис.4(создадим файл *lab10-2.sh* и откроем его в редакторе *etacs*)



```
#!/bin/bash
counter=1
while [ -n "$1" ]
do
    echo "Argument #counter = $1"
    counter=$((counter+1))
    shift
done
```

рис.5(код программы)



```
[root@kdkoifman ~]# chmod +x lab10-2.sh
[root@kdkoifman ~]# ./lab10-2.sh book note abc 1234 lemon apple plane flame too 73799583
Argument #1 = book
Argument #2 = note
Argument #3 = abc
Argument #4 = 1234
Argument #5 = lemon
Argument #6 = apple
Argument #7 = plane
Argument #8 = flame
Argument #9 = too
Argument #10 = 73799583
[root@kdkoifman ~]#
```

рис.6(предоставим всем право на выполнение файла lab10-2.sh и выполним его) ## 3 задание. Напишем 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. (рис.7,8,9).

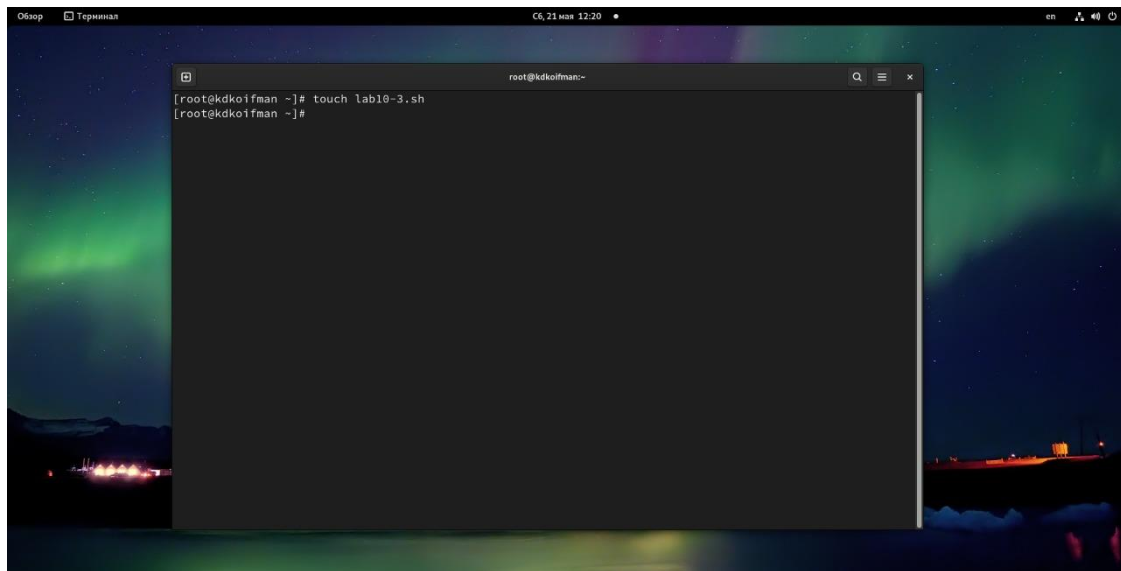


рис.7(создадим файл `lab10-3.sh` и откроем его в редакторе `emacs`)

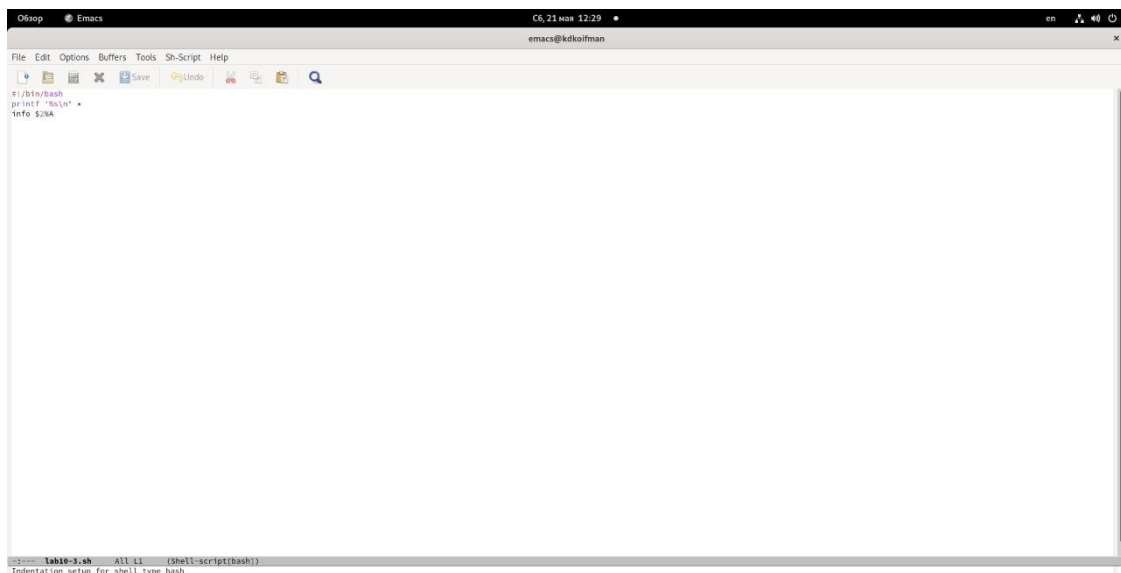
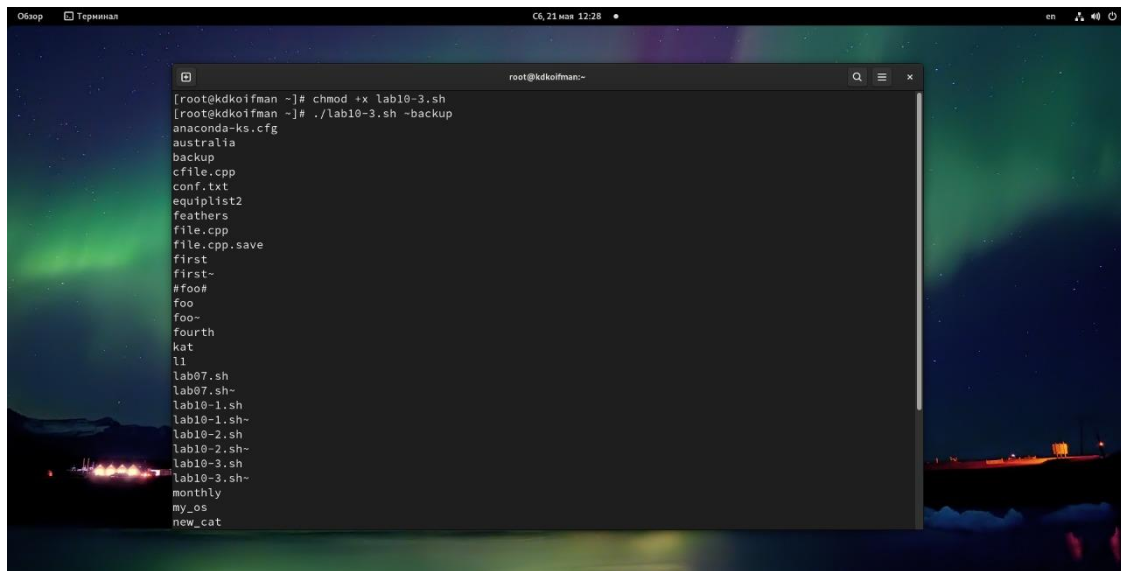
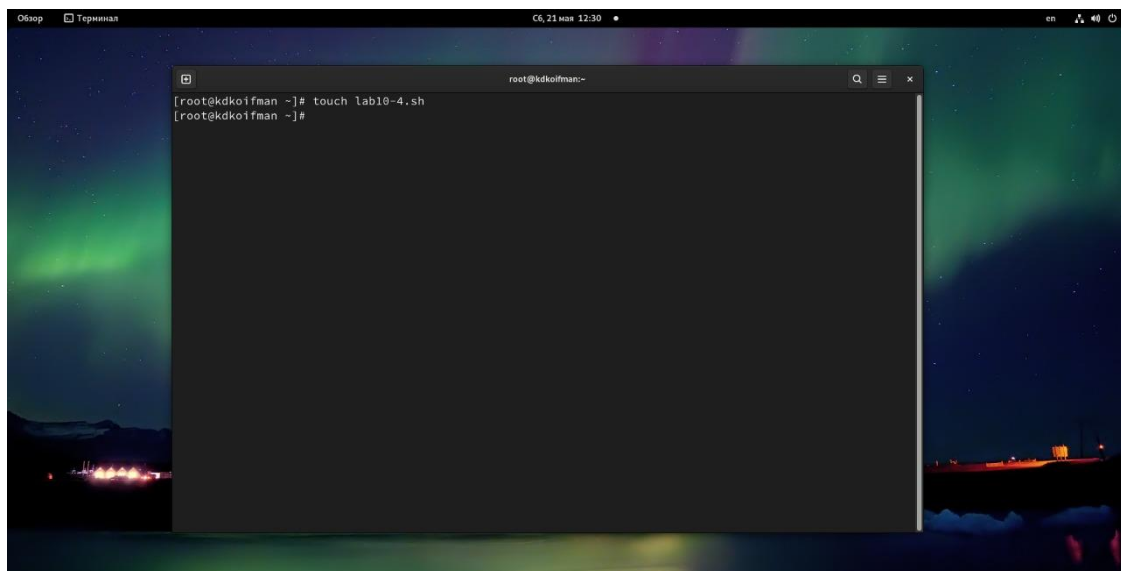


рис.8(код программы)

A terminal window titled 'Обзор' and 'Терминал' with a dark background and a green aurora wallpaper. The terminal shows the user 'root@kdkoifman' at the prompt. The user enters 'chmod +x lab10-3.sh' and then './lab10-3.sh -backup'. The command lists various files in the current directory, including 'anaconda-ks.cfg', 'australia', 'backup', 'cfile.cpp', 'conf.txt', 'equiplist2', 'feathers', 'file.cpp', 'file.cpp.save', 'first', 'first-', '#foo#', 'foo', 'foo-', 'fourth', 'kat', 'll', 'lab07.sh', 'lab07.sh-', 'lab10-1.sh', 'lab10-1.sh-', 'lab10-2.sh', 'lab10-2.sh-', 'lab10-3.sh', 'lab10-3.sh-', 'monthly', 'my\_os', and 'new\_cat'.

```
root@kdkoifman ~]# chmod +x lab10-3.sh
root@kdkoifman ~]# ./lab10-3.sh -backup
anaconda-ks.cfg
australia
backup
cfile.cpp
conf.txt
equiplist2
feathers
file.cpp
file.cpp.save
first
first-
#foo#
foo
foo-
fourth
kat
ll
lab07.sh
lab07.sh-
lab10-1.sh
lab10-1.sh-
lab10-2.sh
lab10-2.sh-
lab10-3.sh
lab10-3.sh-
monthly
my_os
new_cat
```

рис.9(предоставим всем право на выполнение файла *lab10-3.sh* и выполним его) ## 4 задание. Напишем 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. (рис.10,11,12).

A terminal window titled 'Обзор' and 'Терминал' with a dark background and a green aurora wallpaper. The terminal shows the user 'root@kdkoifman' at the prompt. The user enters 'touch lab10-4.sh' and then '#'.

```
root@kdkoifman ~]# touch lab10-4.sh
root@kdkoifman ~]#
```

рис.10(создадим файл *lab10-4.sh* и откроем его в редакторе *etacs*)

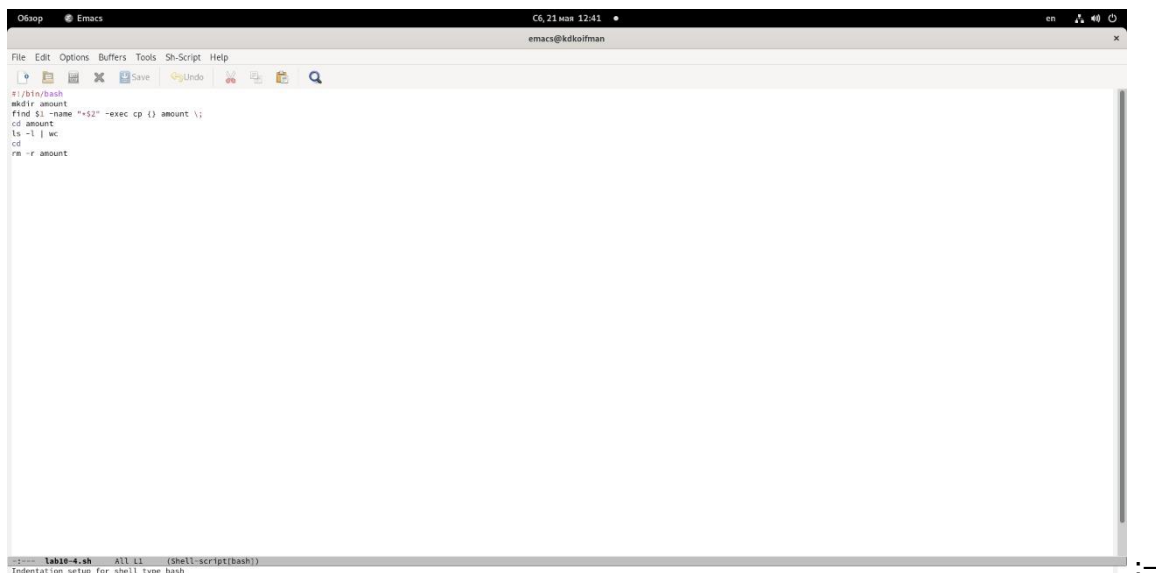


рис.11(код программы)

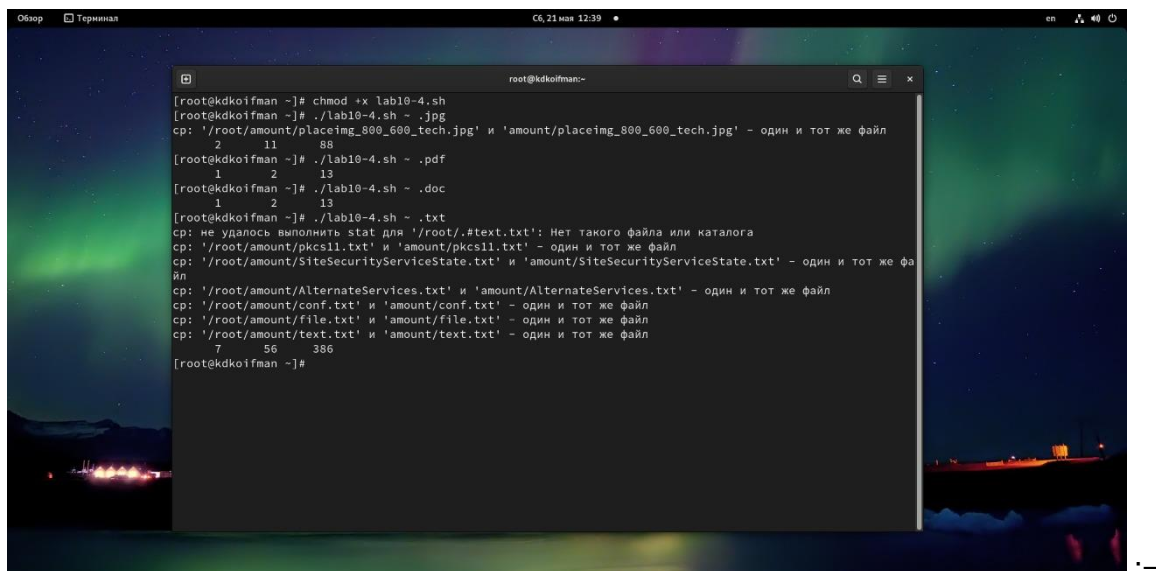


рис.12(предоставим всем право на выполнение файла lab10-4.sh и выполним его)

## Вывод.

В ходе проделанной работы мной были освоены основные навыки программирования в оболочке ОС UNIX/Linux, которые были закреплены при написании небольших командных файлов.

## Контрольные вопросы.

1. Командная оболочка — 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 C-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).
2. 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.
  3. 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. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол \$. 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}. Оболочка bash позволяет работать с массивами. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами.
  4. Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. 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, например:



```
echo "Please enter Month and Day of Birth ?"  
read mon day trash
```

В переменные mon и day будут считаны соответствующие значения, введенные с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать её.

5. Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. (()) используется для записи условия оболочки bash(внутри двойных скобок также можно проводить математические операции и возвращать результат вычислений)
7. - HOME — имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.

- IFS — последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line).

- MAIL — командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта). - TERM — тип используемого терминала.

- LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

Переменные PS1 и PS2 предназначены для отображения промптера командного процессора. PS1 — это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2.

8. Такие символы, как ' < > \* ? | " &, являются метасимволами и имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола.
9. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом.
10. 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
```

11. ретацию. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями:

- `-f` — перечисляет определённые на текущий момент функции;
- `-ft` — при последующем вызове функции иницирует её трассировку;
- `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек;
- `-fu` — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции.

12. Команда `test` создана специально для использования в командных файлах. Единственная функция той команды заключается в выработке кода завершения. Например, команда

```
test -f file
```

возвращает нулевой код завершения (истина), если файл `file` существует, и ненулевой код завершения (ложь) в противном случае: – `test -f file` — истина, если файл `file` существует; – `test -d file` — истина, если файл `file` является каталогом.

13. Команда `set` предназначена для вывода списка переменных окружения. Команда `typeset` предназначена для наложения какого-либо ограничения на переменную. Команда `unset` используется для удаления переменной из окружения командной оболочки.
14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов `$i`, где  $0 < i < 10$ , вместо неё будет осуществлена подстановка значения параметра с порядковым номером  $i$ , т.е. аргумента командного файла с порядковым номером  $i$ .

15. - `$*` — отображается вся командная строка или параметры оболочки;
- `$?` — код завершения последней выполненной команды;
  - `$$` — уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
  - `#!` — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
  - `$-` — значение флагов командного процессора;
  - `${#}` — *возвращает целое число — количество слов, которые были результатом `$`*;
  - `${#name}` — возвращает целое значение длины строки в переменной `name`;
  - `${name[n]}` — обращение к `n`-му элементу массива;
  - `${name[*]}` — перечисляет все элементы массива, разделённые пробелом;
  - `${name[@]}` — то же самое, но позволяет учитывать символы пробелы в самих переменных;
  - `${name:-value}` — если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
  - `${name:value}` — проверяется факт существования переменной;
  - `${name=value}` — если `name` не определено, то ему присваивается значение `value`;
  - `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
  - `${name+value}` — это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`;
  - `${name#pattern}` — представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`);
  - `*${#name[*]}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.