**Лабораторная работа №10**

## Аксенова Анастасия

**Содержание**

1. [Цель работы](#_bookmark0) 5
2. [Задание](#_bookmark1) 6
3. [Выполнение лабораторной работы](#_bookmark2) 7
4. [Библиография](#_bookmark25) 26
5. [Выводы](#_bookmark26) 27

**List of Tables**

**List of Figures**

[3.1 Работа с консолью](#_bookmark3) . . . . . . . . . . . . . . . . . . . . . . . . . . 7

[3.2 Информация о zip](#_bookmark4) . . . . . . . . . . . . . . . . . . . . . . . . . . 8

[3.3 Информация о bzip2](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . . . . 9

* 1. [Информация о tar](#_bookmark6) 10
  2. [Создание файла](#_bookmark7) 10
  3. [Скрипт №1](#_bookmark8) 11
  4. [Проверка работы скрипта](#_bookmark9) 12
  5. [Проверка работы скрипта](#_bookmark10) 12
  6. [Создание файла](#_bookmark11) 13
  7. [Открываем emacs](#_bookmark12) 13
  8. [Скрипт №2](#_bookmark13) 14
  9. [Проверка работы скрипта](#_bookmark14) 15
  10. [Проверка работы скрипта](#_bookmark15) 15
  11. [Создание файла](#_bookmark16) 16
  12. [Скрипт №3](#_bookmark17) 16
  13. [Скрипт №3](#_bookmark18) 17
  14. [Проверка работы скрипта](#_bookmark19) 17
  15. [Проверка работы скрипта](#_bookmark20) 18
  16. [Проверка работы скрипта](#_bookmark21) 18
  17. [Создание файла](#_bookmark22) 19
  18. [Скрипт №4](#_bookmark23) 19
  19. [Проверка работы скрипта](#_bookmark24) 20

# Цель работы

Цель данной лабораторной работы — Изучить основы программирования в обо- лочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

# Задание

* 1. Сделать отчёт по лабораторной работе №11 в формате Markdown.
  2. Изучить основы программирования в оболочке ОС UNIX/Linux.

# Выполнение лабораторной работы

1. Для начала я изучила команды архивации, используя команды «manzip»,

«manbzip2», «mantar» (алгоритм действий представлен на рис. [3.1](#_bookmark3) , [3.2](#_bookmark4) , [3.3](#_bookmark5) , [3.4](#_bookmark6) ).

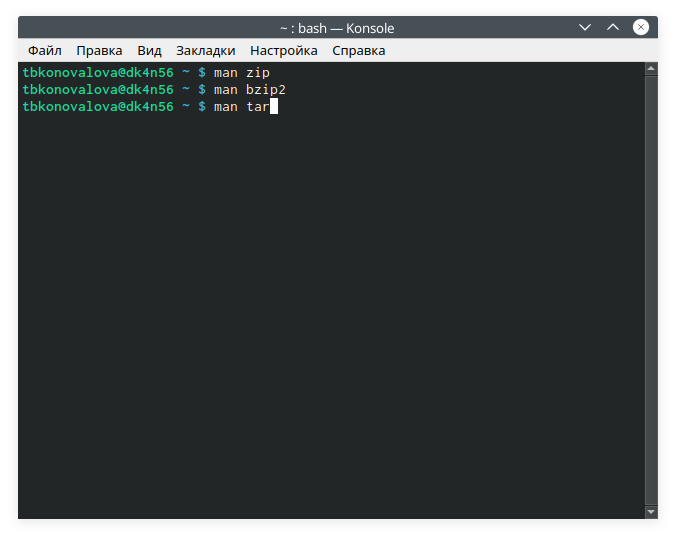


Figure 3.1: Работа с консолью

Синтаксис команды zip для архивации файла: zip [опции] [имя файла.zip] [файлы или папки, которые будем архивировать]

Синтаксис команды zip для разархивации/распаковки файла: unzip [опции] [файл\_архива.zip][файлы]-x[исключить]-d[папка]

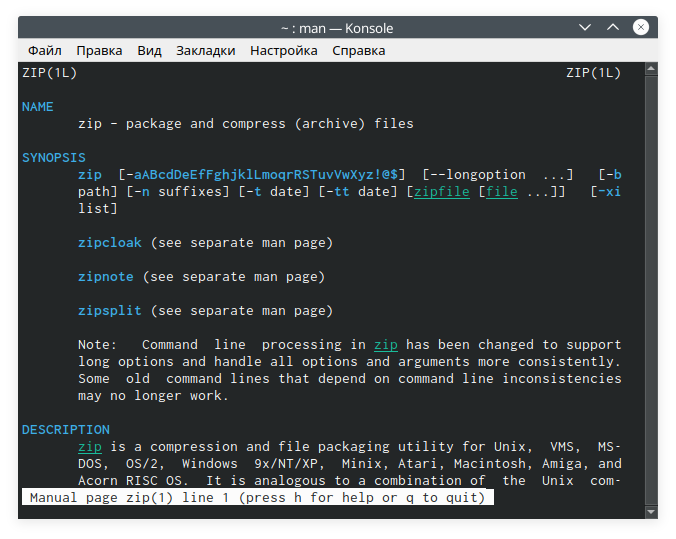


Figure 3.2: Информация о zip

Синтаксис команды bzip2 для архивации файла: bzip2 [опции] [имена файлов] Синтаксис команды bzip2 для разархивации/распаковки файла: bunzip2[опции]

[архивы.bz2]

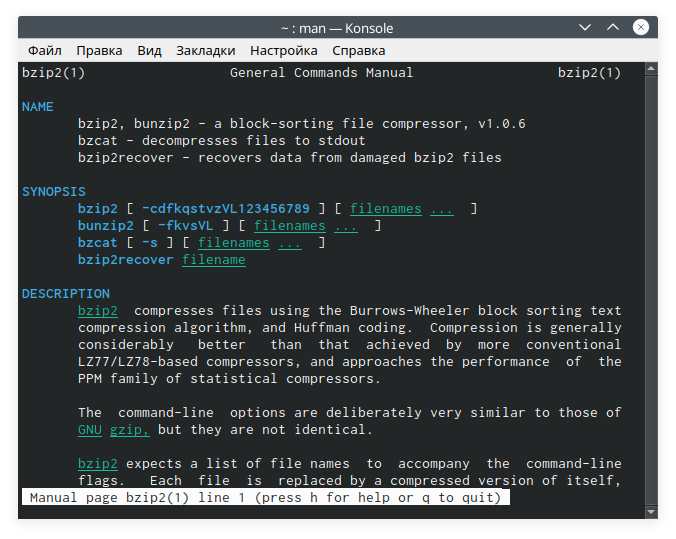


Figure 3.3: Информация о bzip2

Синтаксис команды tar для архивации файла: tar[опции][архив.tar][фай- лы\_для\_архивации]

Синтаксис команды tar для разархивации/распаковки файла: tar[опции][ар- хив.tar]

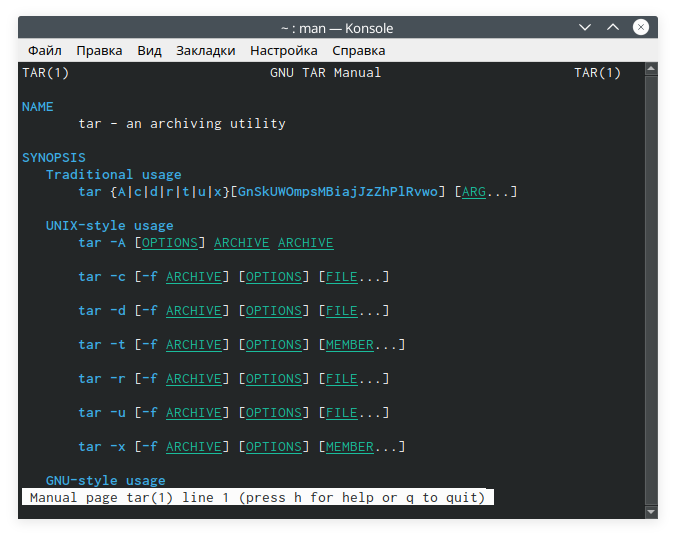


Figure 3.4: Информация о tar

Создала файл, в котором будуписать первый скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x»и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (Скриншот [3.5](#_bookmark7) ).

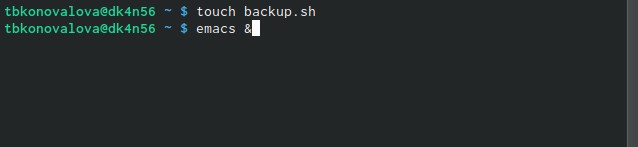


Figure 3.5: Создание файла

Написала скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директо- рию back up в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (Скриншот [3.6](#_bookmark8) ). При написании скрипта использовала архиватор bzip2.

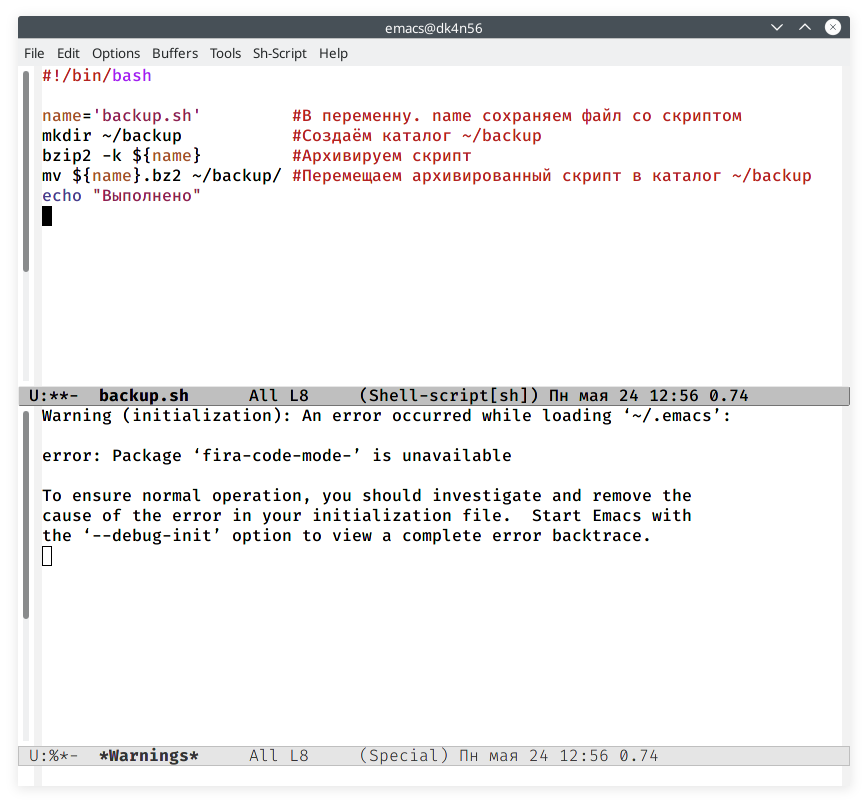


Figure 3.6: Скрипт №1

Проверила работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod+x\*.sh»). Проверила, появился ли каталог backup/, перейдя в него (команда «cd backup/»), посмотрела его со- держимое (команда «ls») и просмотрела содержимое архива (команда «bunzip2

-cbackup.sh.bz2») (алгоритм действий представлен на рис. [3.7](#_bookmark9) , [3.8](#_bookmark10) ).Скрипт рабо- тает корректно.

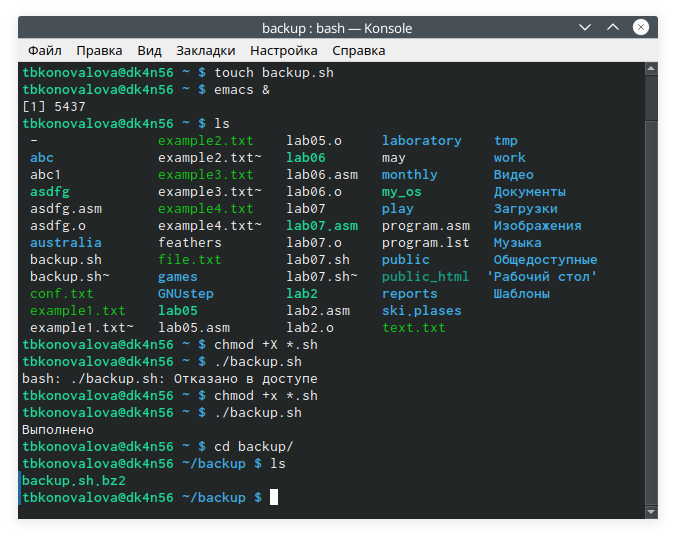


Figure 3.7: Проверка работы скрипта

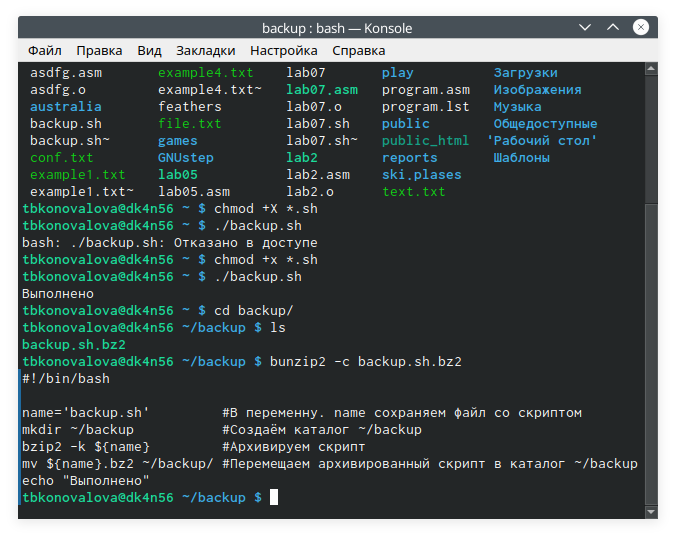


Figure 3.8: Проверка работы скрипта

* 1. Создала файл, в котором буду писать второйскрипт, и открыла его в редак- торе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и

«emacs &») (Скриншоты [3.9](#_bookmark11) , [3.10](#_bookmark12) ).

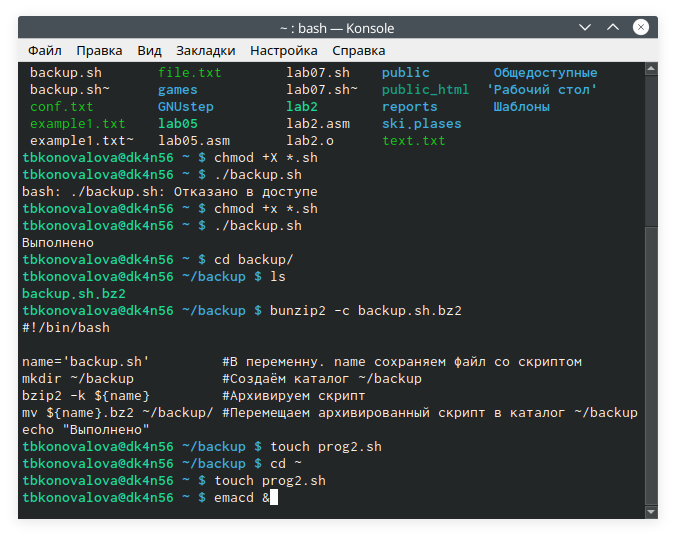


Figure 3.9: Создание файла

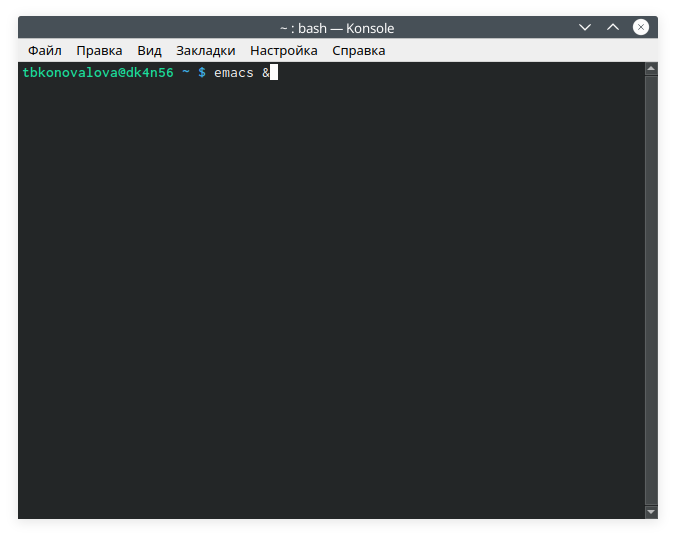


Figure 3.10: Открываем emacs

Написала пример командного файла, обрабатывающего любое произвольное

число аргументов командной строки, в том числе превышающее десять. Напри- мер, скрипт может последовательно распечатывать значения всех переданных аргументов (Рисунок [3.11](#_bookmark13) ).

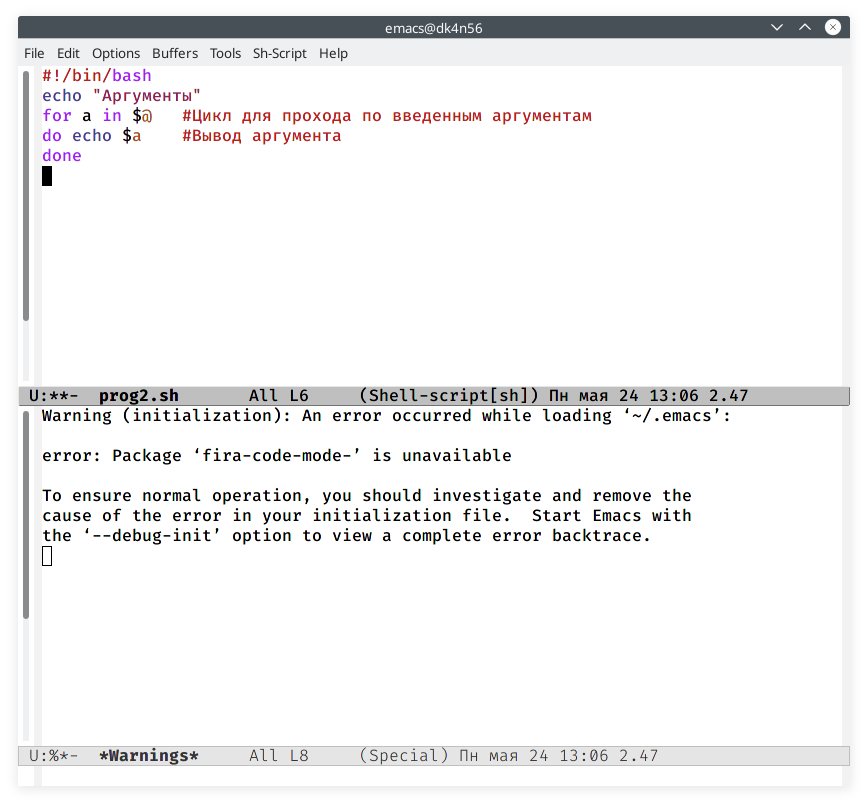


Figure 3.11: Скрипт №2

Проверила работу написанного скрипта (команды «./prog2.sh0 1 2 3 4» и

«./prog2.sh0 1 2 3 45 6 7 8 9 10 11»), предварительно добавив для него право на выполнение (команда «chmod+x\*.sh»). Вводила аргументы количество которых меньше 10 и больше 10 (алгоритм действий представлен на рис. [3.12](#_bookmark14) , [3.13](#_bookmark15) ). Скрипт работает корректно.

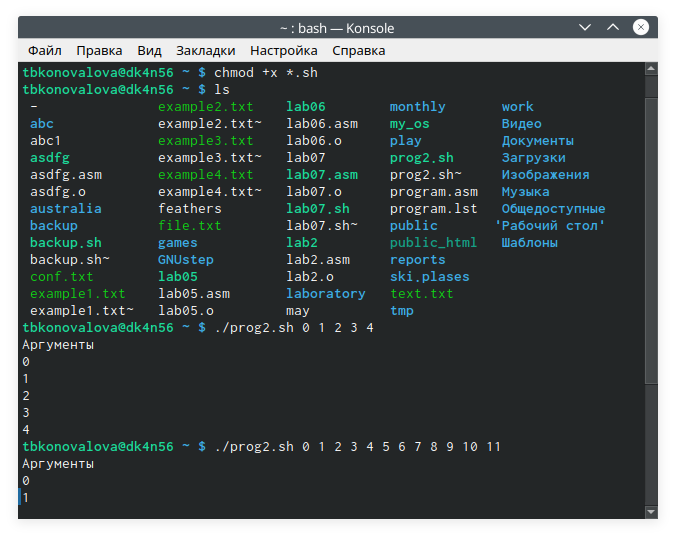


Figure 3.12: Проверка работы скрипта

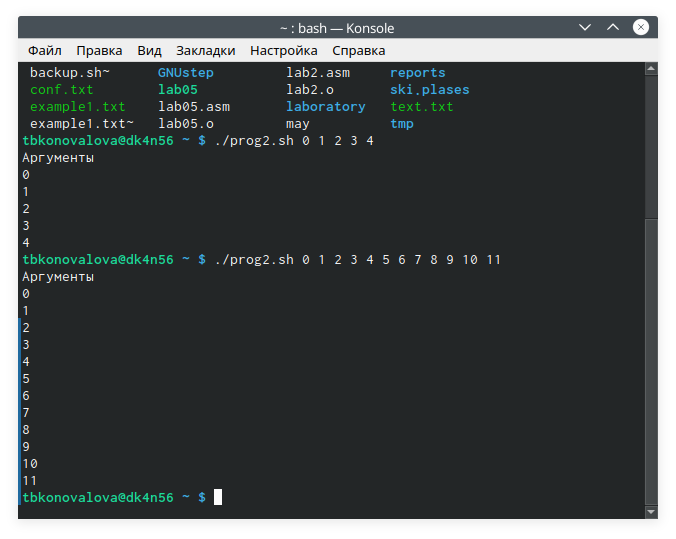


Figure 3.13: Проверка работы скрипта

* 1. Создала файл, в котором буду писать третий скрипт, и открыла его в редак- торе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touchprogls.sh» и

«emacs&») (алгоритм действий представлен на рис. [3.14](#_bookmark16) ).

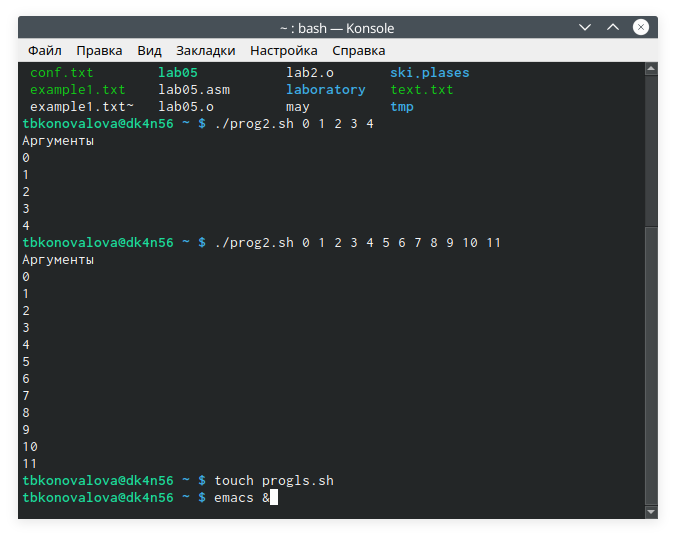


Figure 3.14: Создание файла

Написала командный файл − аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном катало- ге и выводить информацию о возможностях доступа к файлам этого каталога (Скриншоты [3.15](#_bookmark17) , [3.16](#_bookmark18) ).

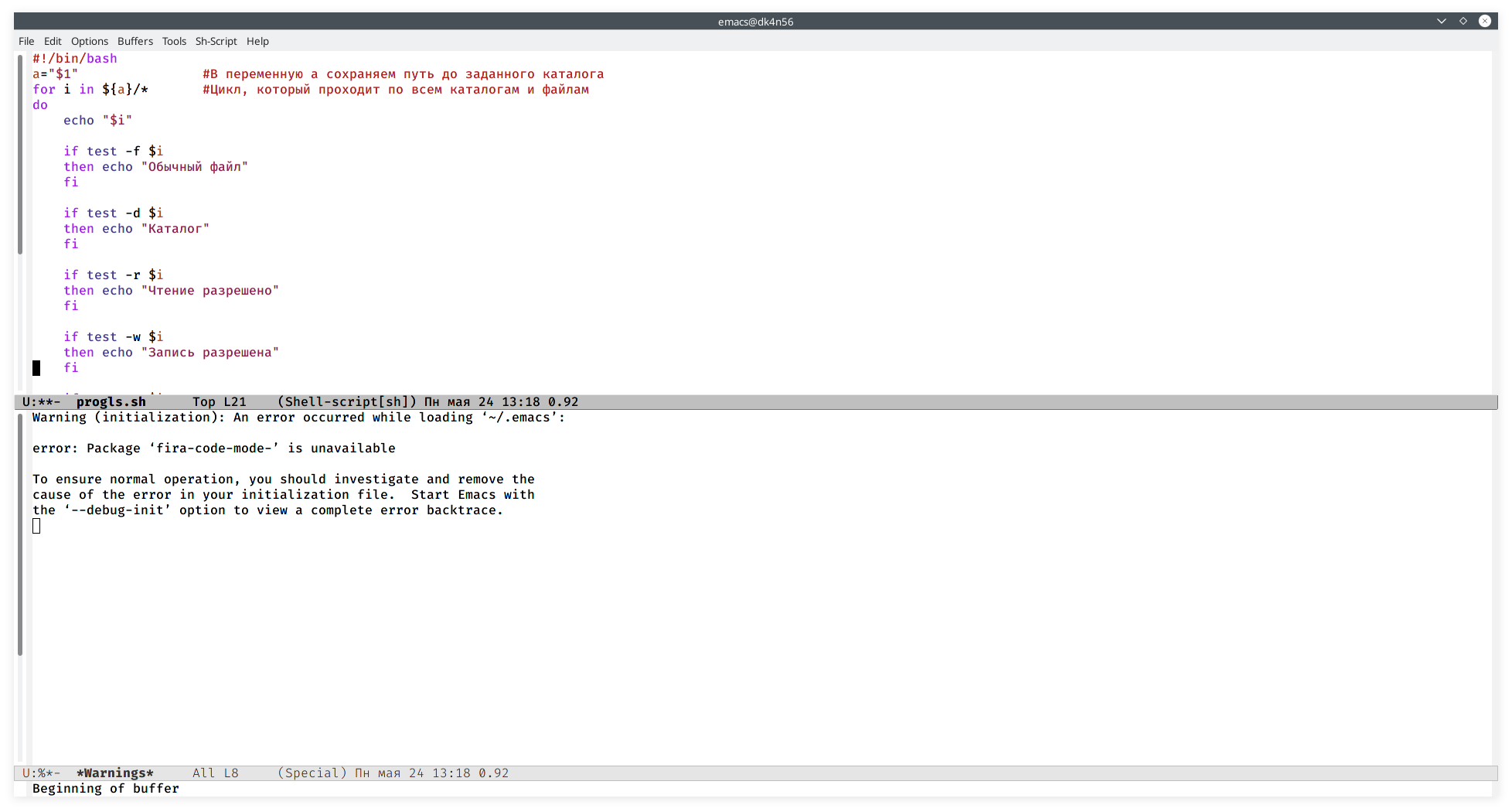


Figure 3.15: Скрипт №3



Figure 3.16: Скрипт №3

Далее проверила работу скрипта (команда «./progls.sh~»), предварительно доба- вив для него право на выполнение (команда «chmod+x\*.sh») (алгоритм действий представлен на рис. [3.17](#_bookmark19) , [3.18](#_bookmark20) , [3.19](#_bookmark21) ). Скрипт работает корректно.

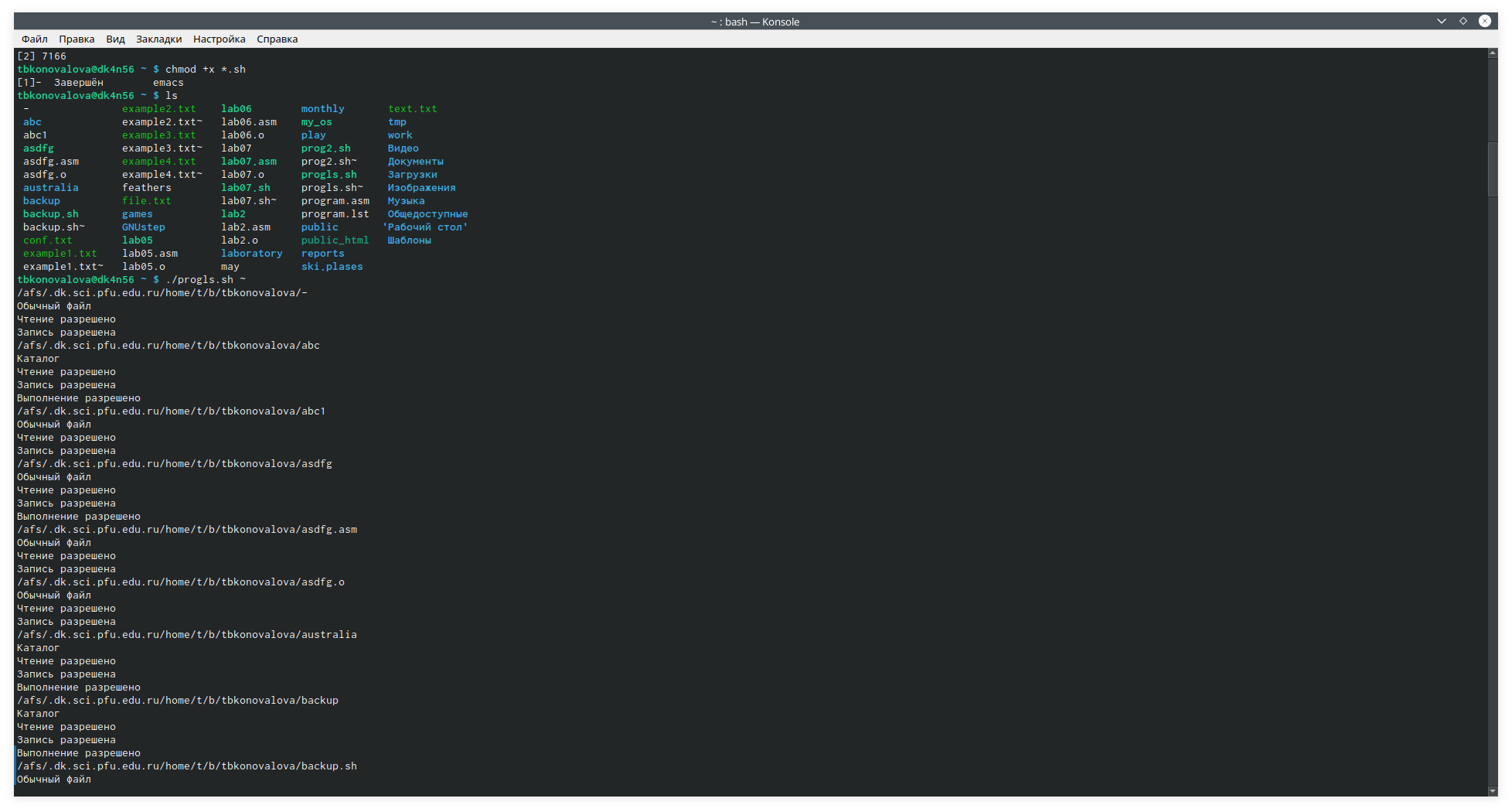


Figure 3.17: Проверка работы скрипта

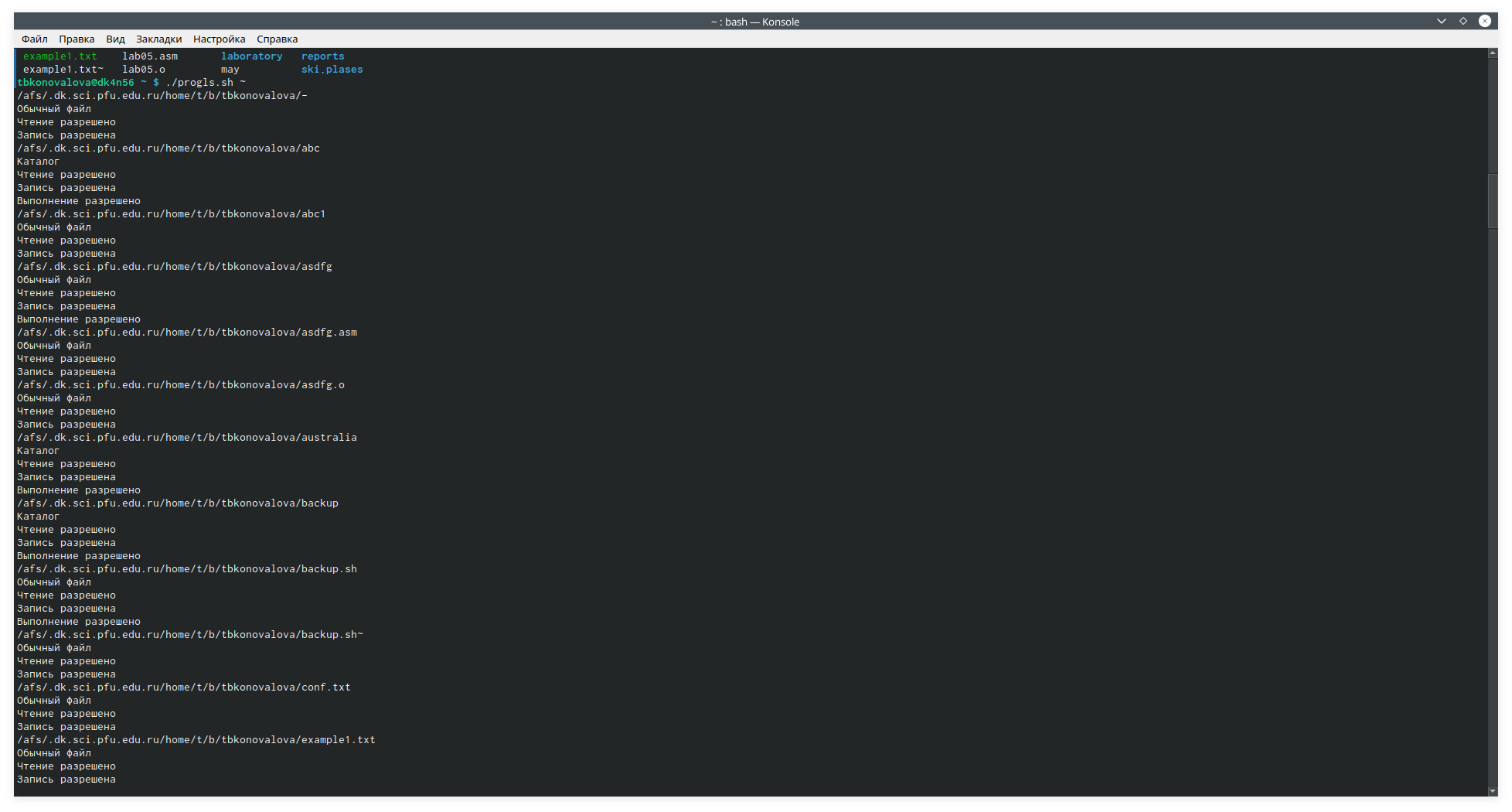


Figure 3.18: Проверка работы скрипта

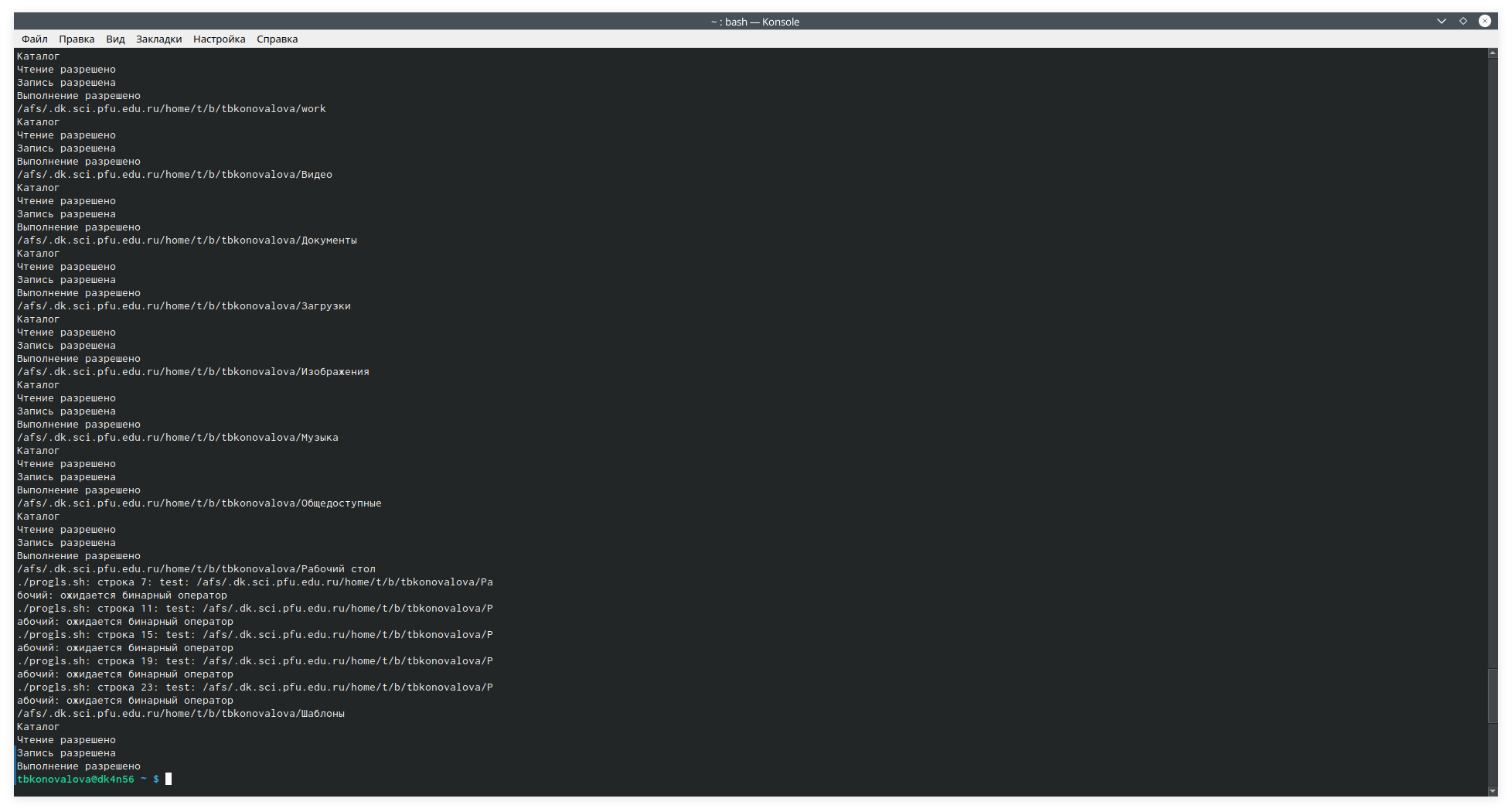


Figure 3.19: Проверка работы скрипта

* 1. Для четвертого скрипта создала файл (команда «touch format.sh») и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (Скриншот [3.20](#_bookmark22) ).

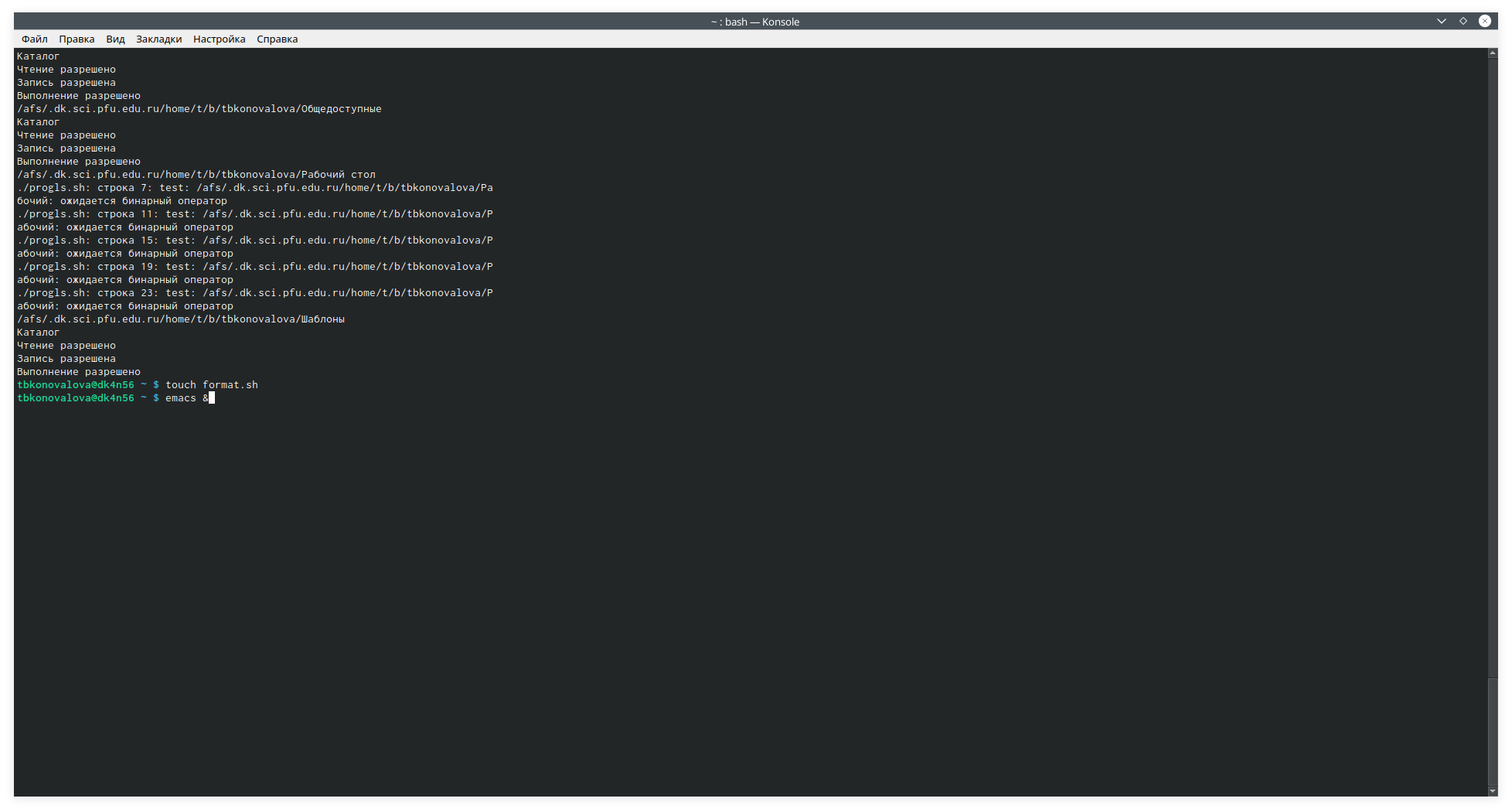


Figure 3.20: Создание файла

Написала 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 (Скриншот [3.21](#_bookmark23) ).

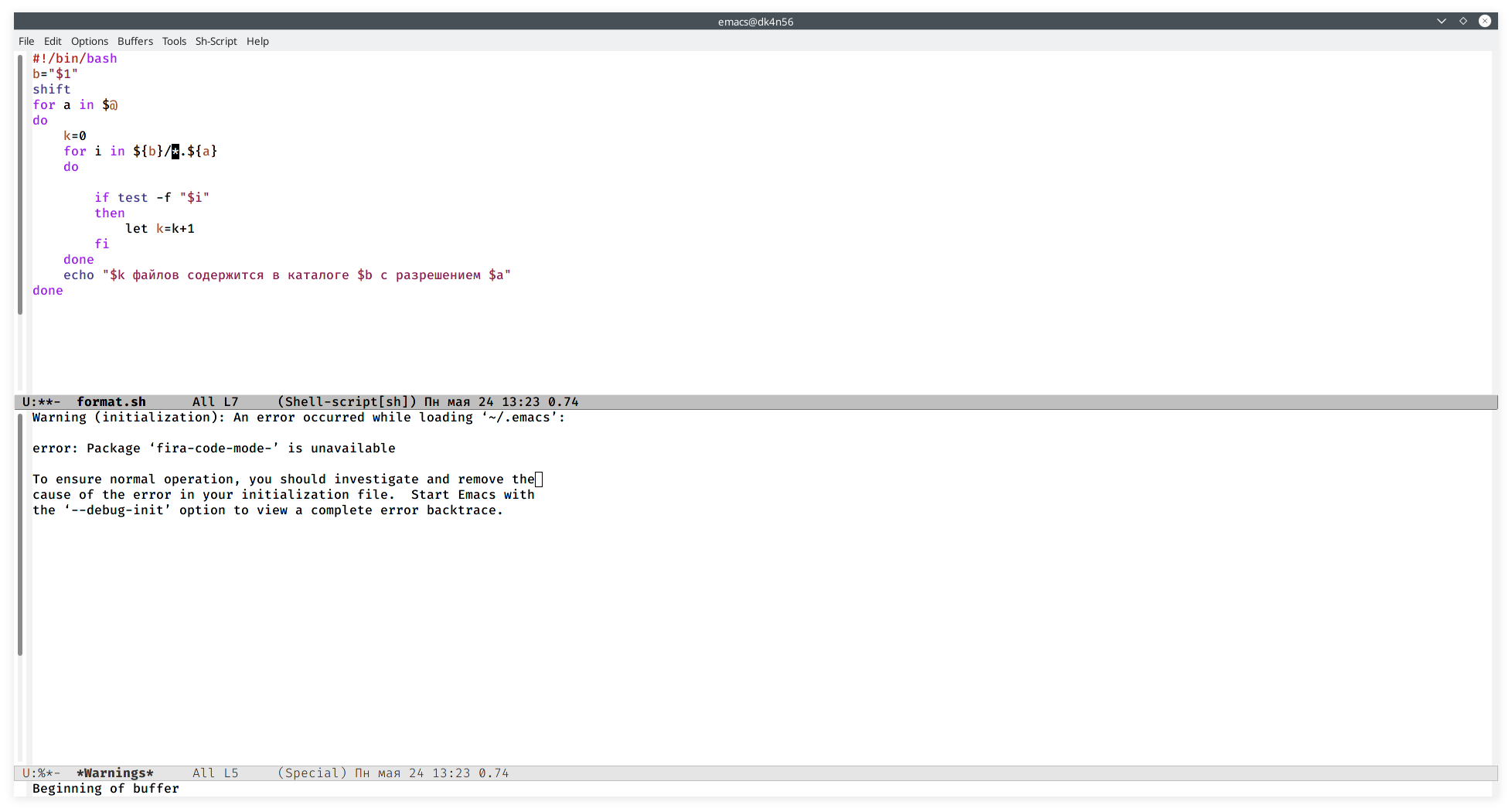


Figure 3.21: Скрипт №4

Проверила работу написанного скрипта (команда «./format.sh~ pdf sh txt doc»), предварительно добавив для него право на выполнение (команда «chmod+x\*.sh»), а также создав дополнительные файлы с разными расширениями (команда «touch file.pdf file1.doc file2.doc») (Рисунок [3.22](#_bookmark24) ).Скрипт работает корректно.

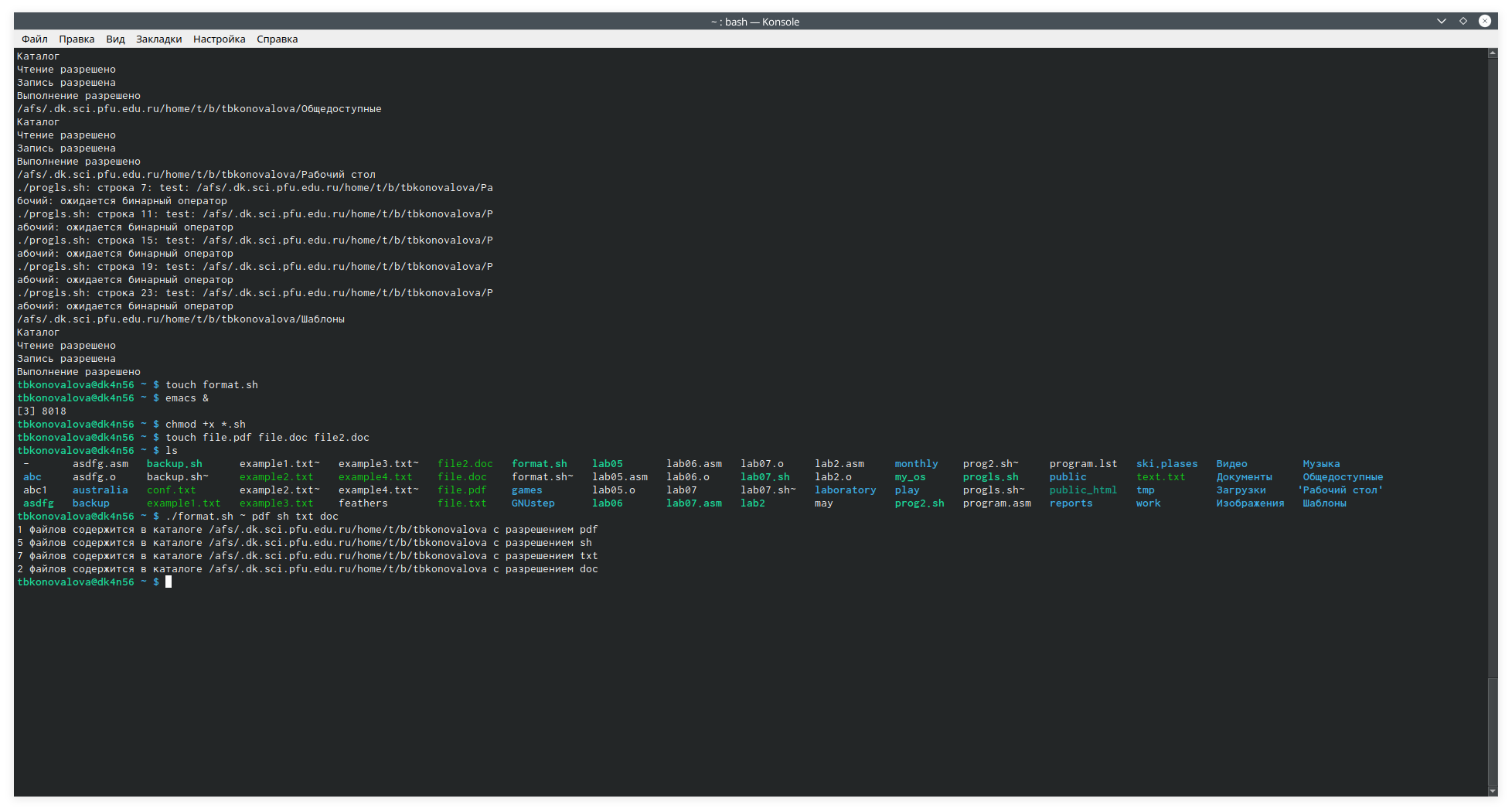


Figure 3.22: Проверка работы скрипта

Ответы на контрольные вопросы:

1. 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: 1. оболочка Борна (Bourneshellили sh) − стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; 2. С-оболочка (или csh) −надстрой- ка на оболочкой Борна, использующая Сподобный синтаксис команд с возмож- ностью сохранения истории выполнения команд; 3. Оболочка Корна (или ksh) − напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна; 4. BASH − сокращение от BourneAgainShell(опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разра- ботка компании FreeSoftwareFoundation).
2. POSIX (Portable Operating System Interface for Computer Environments ) − набор стандартов описания интерфейсов взаимодействия операционной систе- мы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electricaland Electronics Engineers) для обеспечения совместимости различных UNIX/Linux подобных операционных систем и переносимости при- кладных программ на уровне исходного кода. POSIX - совместимые оболочки

разработаны на базе оболочки Корна.

1. Командный процессор bash обеспечивает возможность использования пе- ременных типа строка символов. Имена переменных могут быть выбраны поль- зователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда «mark=/usr/andy/bin» присва- ивает значение строки символов /usr/andy/bin переменной mark типа строка символов. Значение, присвоенное некоторой переменной, может быть впослед- ствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует мета- символ ., «𝑚𝑣𝑎𝑓 𝑖𝑙𝑒{mark}» переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin. Оболочка bash позволяет работать с массивами. Для создания массива используется команда setс флагом -A. За флагом следует имя переменной, а затем список значений, разделённых про- белами. Например, «set -Astates Delaware Michigan “New Jersey”». Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента.
2. Оболочка bash поддерживает встроенные арифметические функции. Ко- манда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение − это еди- ничный терм (term), обычно целочисленный. Команда let берет два операнда и присваивает их переменной. Команда read позволяет читать значения пере- менных со стандартного ввода: «echo “Please enter Month and Day of Birth ?”»

«read mon day trash». В переменные monи day будут считаны соответствующие значения, введённые с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.

1. В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%).
2. В (( ))можно записывать условия оболочки bash, а также внутри двойных

скобок можно вычислять арифметические выражения и возвращать результат. 7). Стандартные переменные: 1. PATH: значением данной переменной явля-

ется список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зави- симости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневогоили текущего каталога.

1. PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. PS1 − это промптер командного процессора, по умолчанию его значение равно символу $ или #. Если какая-то интерактив- ная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >.
2. HOME: имя домашнего каталога пользователя. Если команда cdвводится без аргументов, то происходит переход в каталог,указанный в этой переменной.
3. IFS:последовательность символов, являющихся разделителями в команд- ной строке, например, пробел, табуляция и перевод строки (newline).
4. MAIL:командный процессор каждый раз перед выводом на экран промпте- ра проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего вво- да из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение Youhavemail(у Вас есть почта).
5. TERM: тип используемого терминала.
6. LOGNAME: содержит регистрационное имя пользователя, которое устанав- ливается автоматически при входе в систему.
7. Такие символы, как ’ < > \* ? | ” &, являются метасимволами и имеют для командного процессора специальный смысл.
8. Снятие специального смысла с метасимвола называется экранированием мета символа. Экранирование может быть осуществлено с помощью предшеству- ющего мета символу символа , который, в свою очередь, является мета символом. Для экранирования группы метасимволов нужно заключить её в одинарные ка- вычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , ”. Например, –echo\* выведет на экран символ *,–echoab’*|*’cd выведет на экран строку ab*|\*cd.
9. Последовательность команд может быть помещена в текстовый файл. Та- кой файл называется командным. Далее этот файл можно выполнить по команде:

«bash командный\_файл [аргументы]». Чтобы не вводить каждый раз последова- тельности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сдела- но с помощью команды «chmod +x имя\_файла». Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будтоон является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а про- грамма, написанная на языке программирования оболочки, и осуществить её интерпретацию.

1. Группу команд можно объединить в функцию. Для этого существует клю- чевое слово function, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды unsetcфлагом -f.
2. Чтобы выяснить, является ли файл каталогом или обычным файлом, необ- ходимо воспользоваться командами «test-f [путь до файла]» (для проверки, явля- ется ли обычным файлом) и «test -d[путь до файла]» (для проверки, является ли каталогом).
3. Команду «set» можно использовать для вывода списка переменных окру-

жения. В системах Ubuntu и Debia nкоманда «set» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при ра- боте с данными системами рекомендуется использовать команду «set| more». Команда «typeset» предназначена для наложения ограничений на переменные. Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.

1. 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. Использование комбинации символов $0 приводит к подстановке вместо неё имени данного командного файла.
2. Специальные переменные: 1. $\* −отображается вся командная строка или параметры оболочки; 2. $? −код завершения последней выполненной команды;

3. $$ −уникальный идентификатор процесса, в рамках которого выполняется командный процессор; 4. $! −номер процесса, в рамках которого выполняется по- следняя вызванная на выполнение в командном режиме команда; 5. $-−значение флагов командного процессора; 6. ${#*} −возвращает целое число −количествослов, которые были результатом $*; 7. ${#name} −возвращает целое значение длины строки в переменной name; 8. ${name[n]} −обращение к n-му элементу масси- ва; 9. ${name[\*]}−перечисляет все элементы массива, разделённые пробелом;

10. ${name[@]}−то же самое, но позволяет учитывать символы пробелы в самих переменных; 11. ${name:-value} −если значение переменной name не определено,

то оно будет заменено на указанное value; 12. ${name:value} −проверяется факт существования переменной; 13. ${name=value} −если name не определено, то ему присваивается значение value; 14. ${name?value} −останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке;

15. ${name+value} −это выражение работает противоположно ${name-value}. Если переменная определена, то подставляется value; 16. ${name#pattern} −представ- ляет значение переменной name с удалённым самым коротким левым образцом (pattern); 17. ${#name[\*]} и ${#name[@]}−эти выражения возвращают количество элементов в массиве name.

# Библиография

* 1. Программное обеспечение GNU/Linux. Лекция 3. FHS и процессы (Г. Курячий, МГУ);
  2. Программное обеспечение GNU/Linux. Лекция 4. Права доступа (Е. Алёхова, МГУ);
  3. Электронный ресурс: https://ru.wikibooks.org/wiki
  4. Электронный ресурс: <http://fedoseev.net/materials/courses/admin/ch02.html>

# Выводы

В ходе выполнения данной лабораторной работы я изучила основы программи- рования в оболочке ОС UNIX/Linuxи научилась писать небольшие командные файлы.