Лабораторная работа №10

Ханина Людмила Константиновна

Table of Contents

# Цель работы

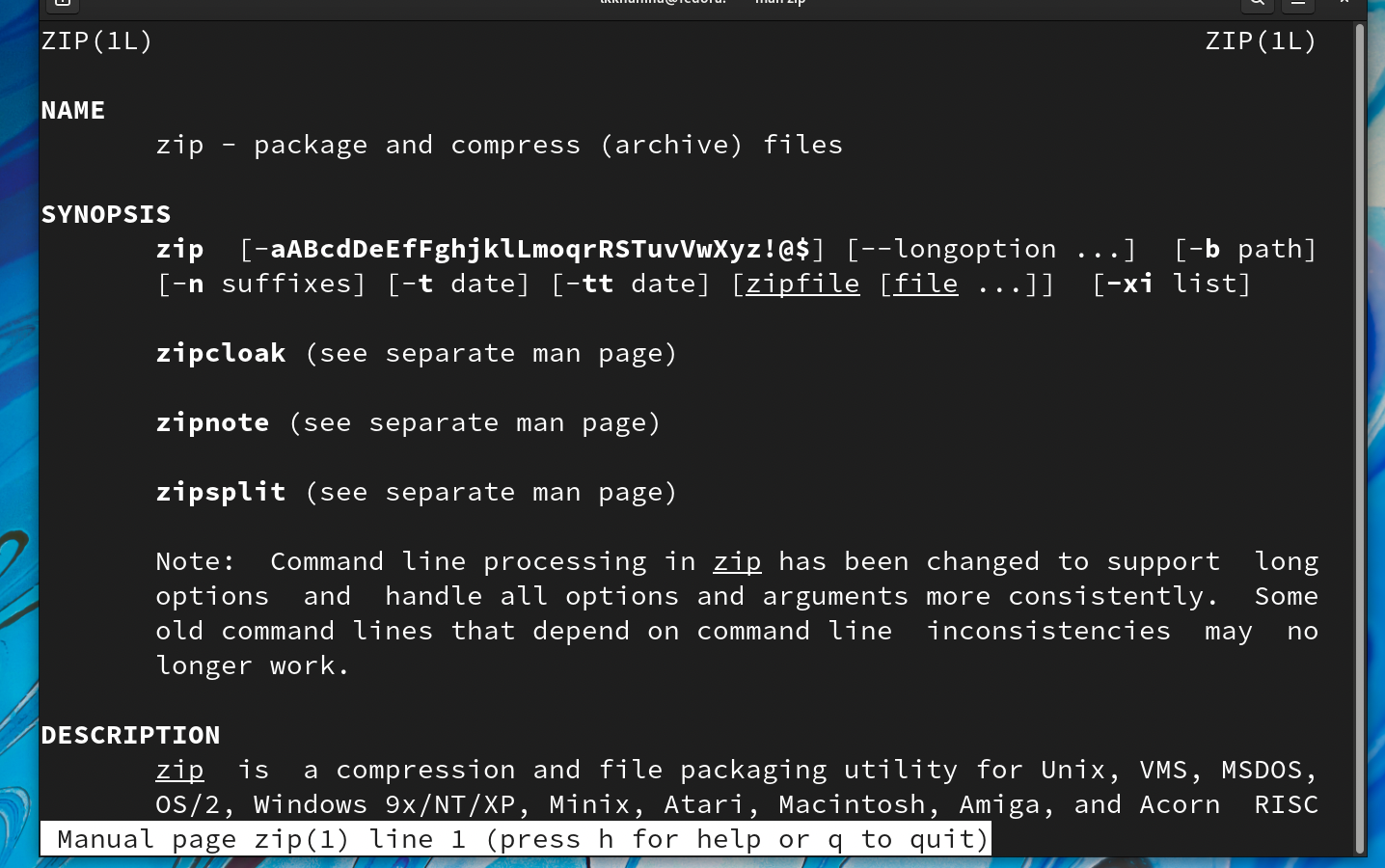
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

# Задание

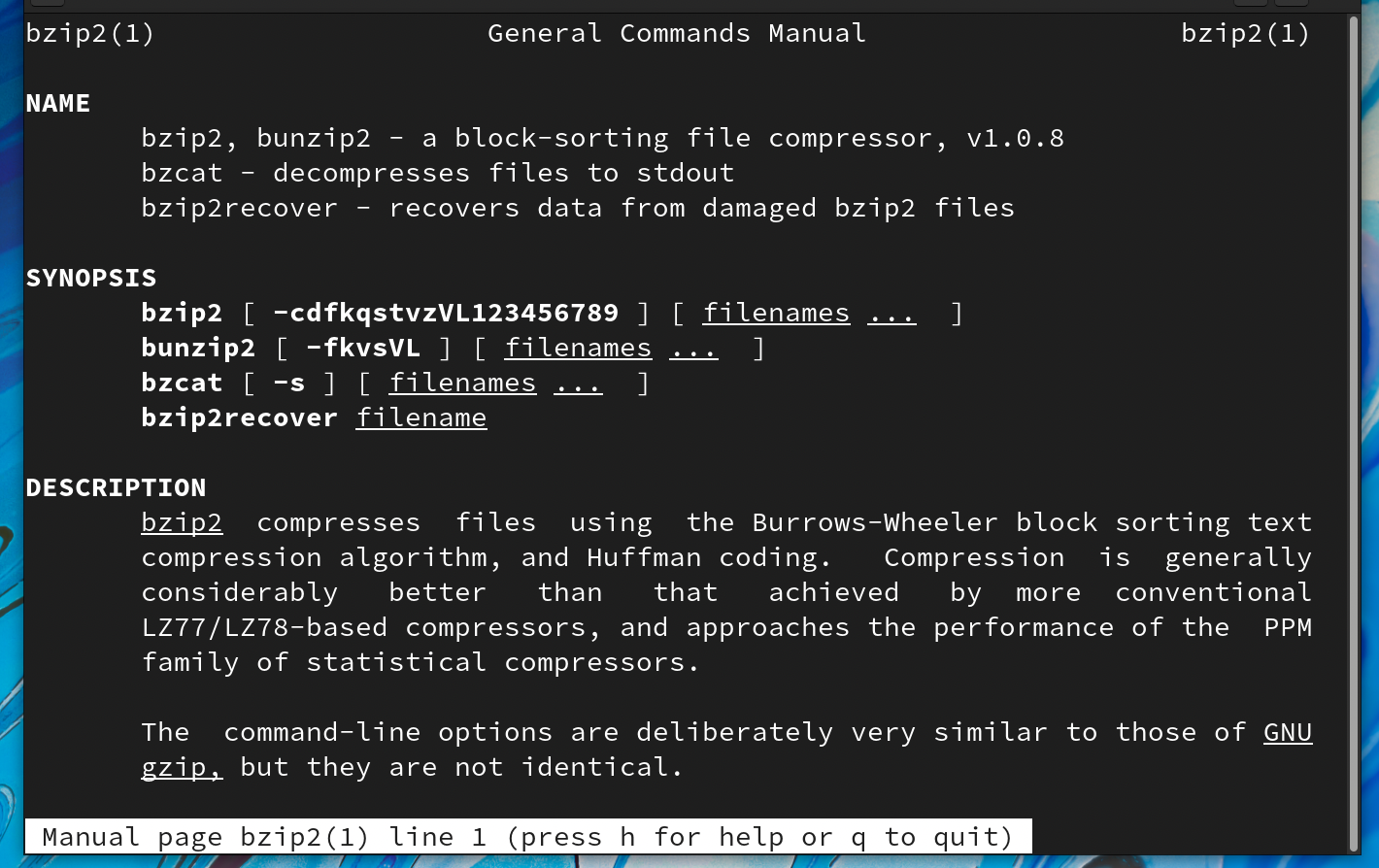
* Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд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

# Выполнение лабораторной работы

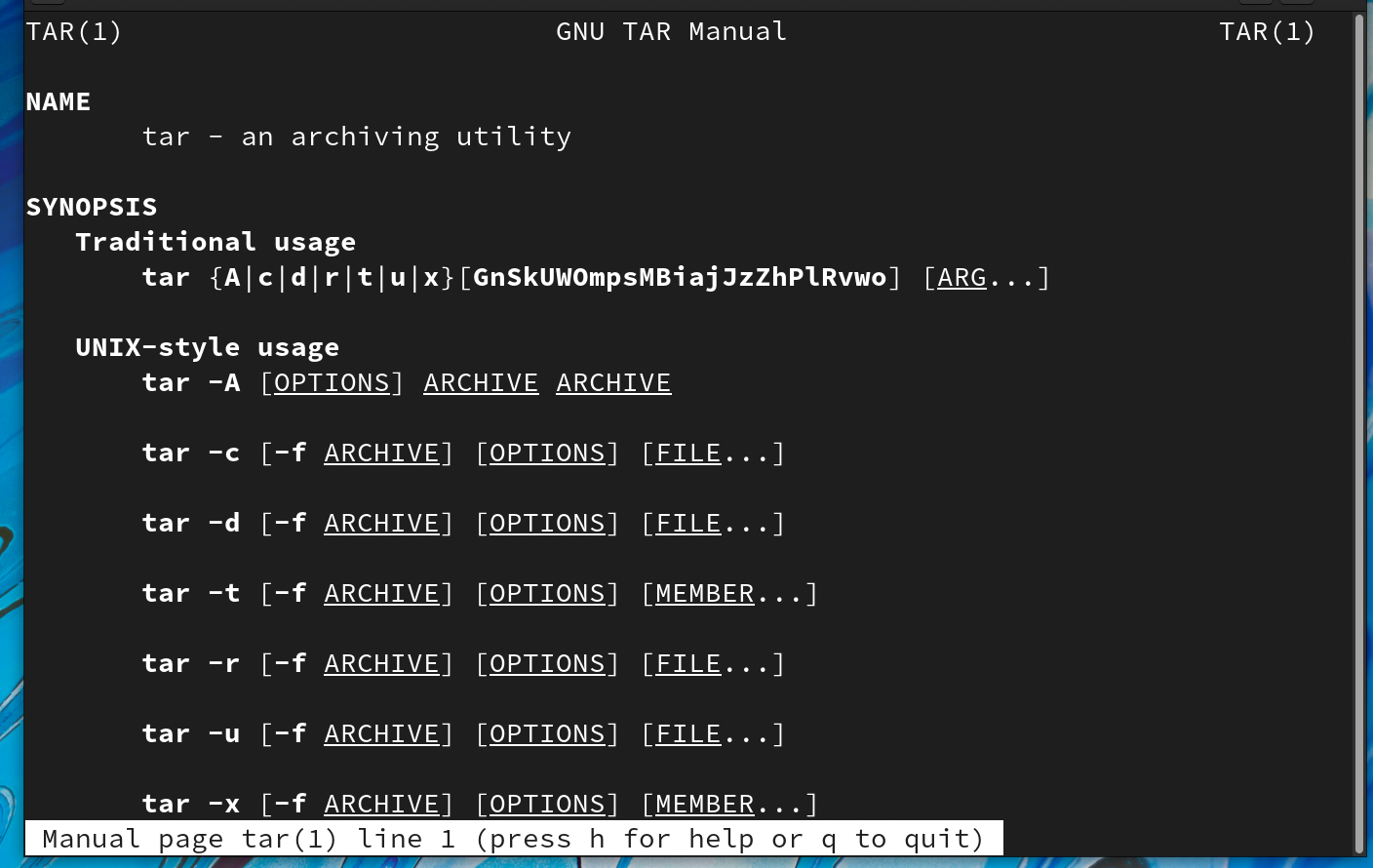
1. C помощью команды man узнаем информацию про zip, bzip2, tar.



zip

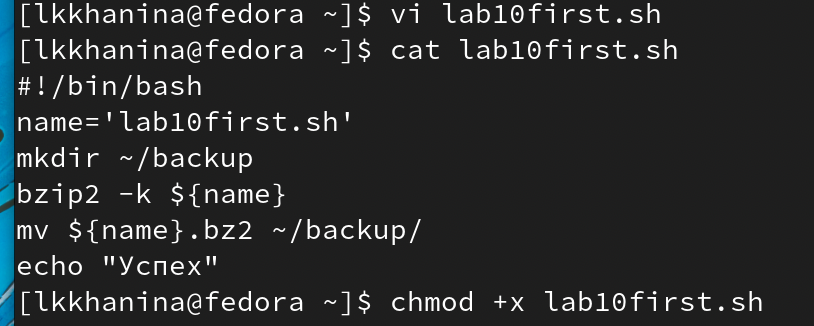


bzip2



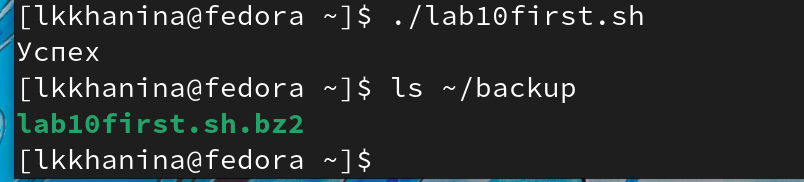
tar

1. Создаем файл lab01first.sh, в котором будем писать скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в нашем домашнем каталоге. Файл будет архивироваться с помощью архиватора bzip2. Перед запуском изменим права доступа, чтобы иметь возможность запускать скрипт.

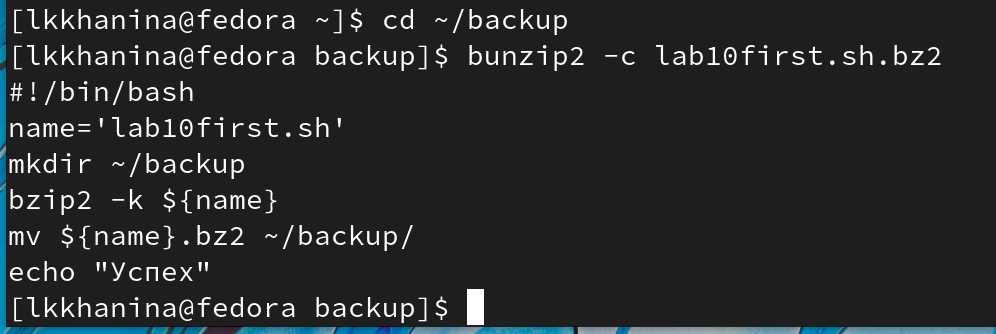


Первый скрипт

1. Посмотрим корректность исполнения скрипта. Для этого зайдем в появившийся каталог backup и посмотрим текст архивированного файла. Все хорошо.

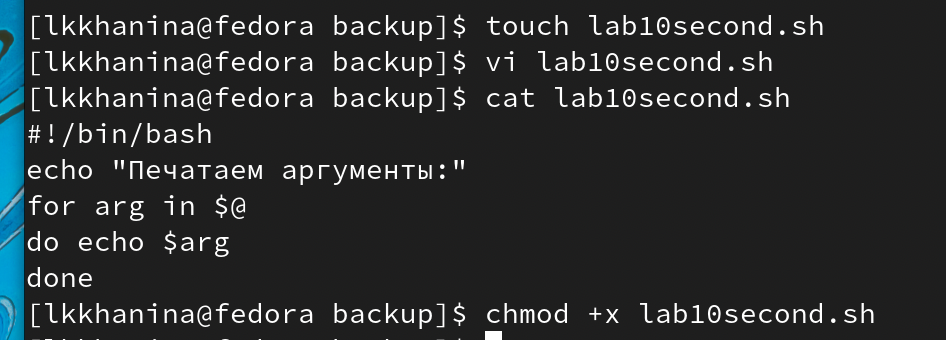


Запускаем первый скрипт



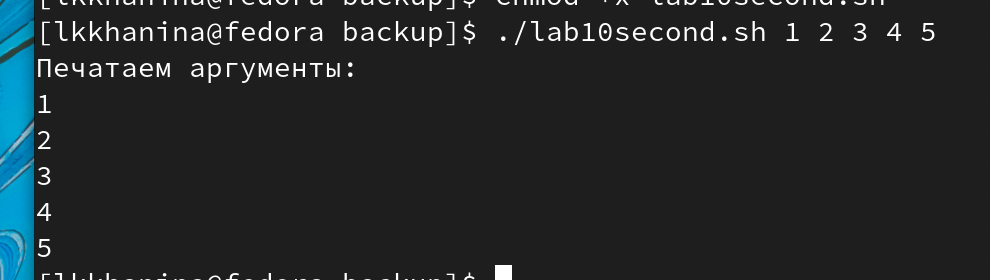
Содержание архивированного файла

1. Далее создаем файл lab10second, в котором будет второй скрипт, обрабатывающий любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов. Сразу изменим права доступа.



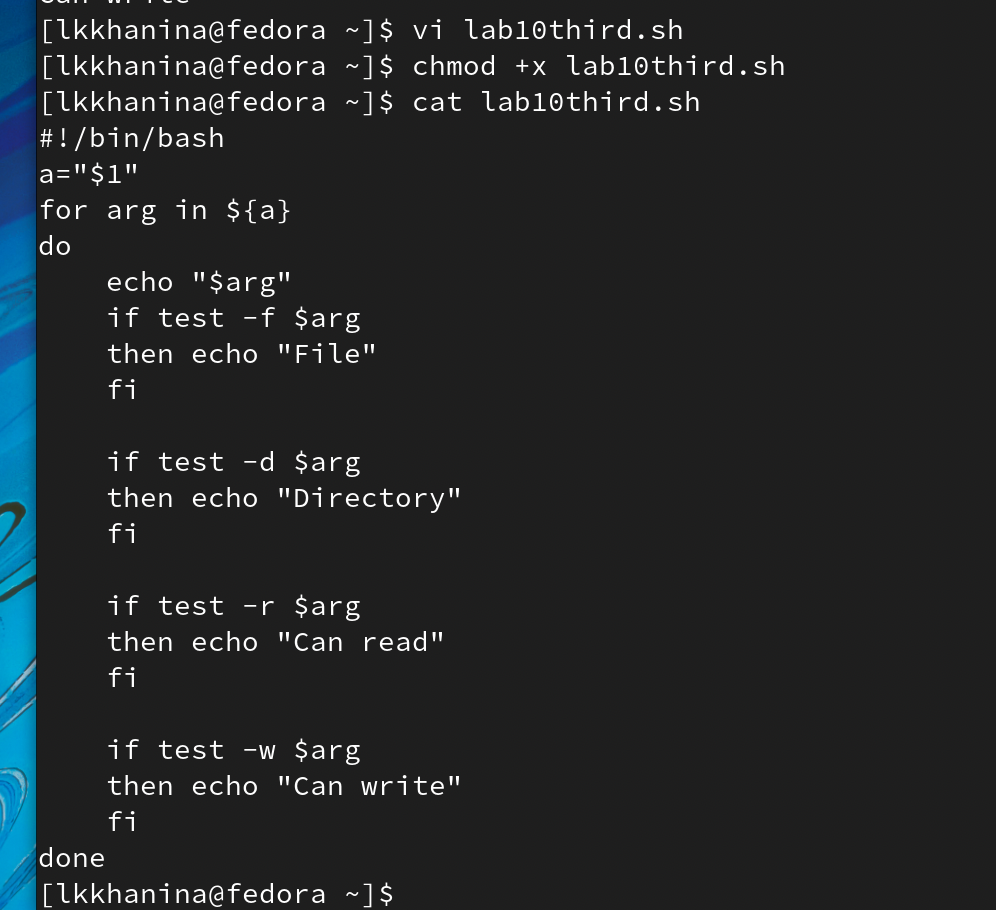
Второй скрипт

1. Запустим скрипт и убедимся, что он исправно работает.



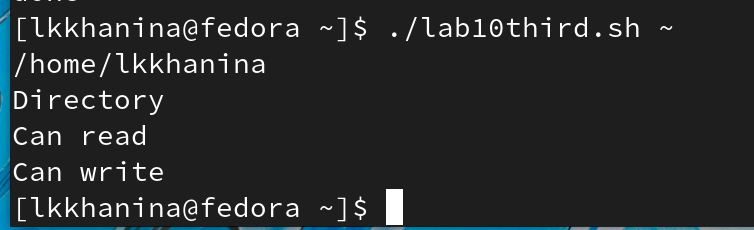
Запускаем второй скрипт

1. Далее создаем файл lab10third, в котором будет третий скрипт, 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. Также изменим права доступа, чтобы суметь запустить скрипт.



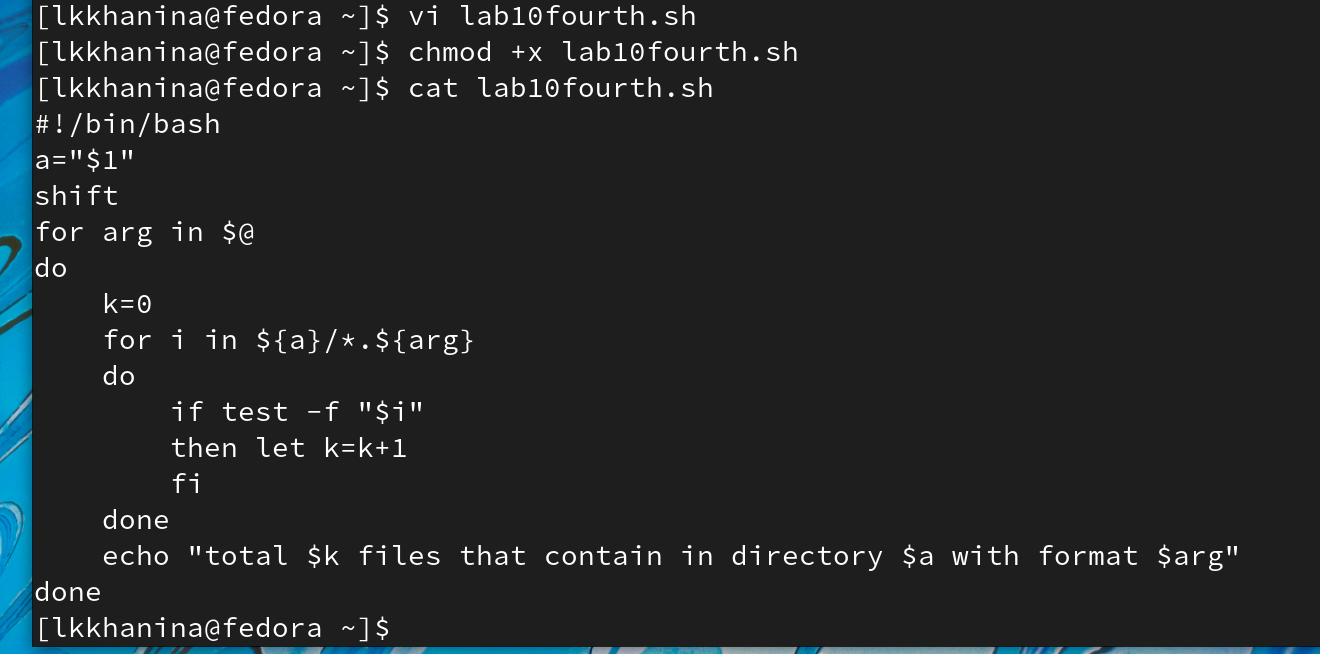
Третий скрипт

1. Запустим скрипт и убедимся, что он исправно работает.



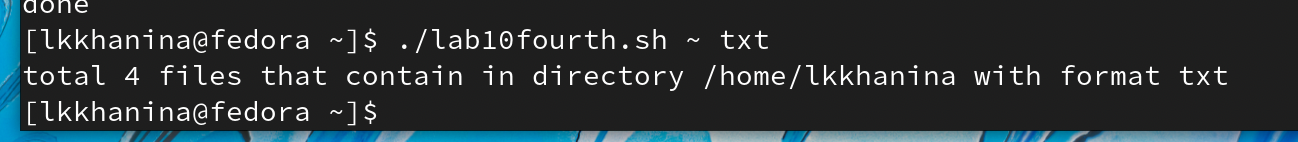
Запускаем третий скрипт

1. Далее создаем файл lab10fourth, в котором будет четвертый скрипт, 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. Также изменим права доступа, чтобы суметь запустить скрипт.



Четвертый скрипт

1. Запустим скрипт и убедимся, что он исправно работает.



Запускаем четвертый скрипт

# Контрольные вопросы

1. Командный процессор (командная оболочка, интерпретатор команд shell) − это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В Linux доступны следующие командные оболочки:

* Bash — самая распространённая оболочка под Linux. Она ведёт историю команд и предоставляет возможность их редактирования;
* pdksh — клон korn shell, хорошо известной оболочки в системах UNIX;
* tcsh — улучшенная версия >C shell;
* zsh — новейшая из перечисленных здесь оболочек; реализует улучшенное дополнение и другие удобные функции.

1. POSIX (англ. Portable Operating System Interface — переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем и переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-Unix систем.
2. 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. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами.
3. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Команда read позволяет читать значения переменных со стандартного ввода.
4. В bash можно применять арифметические операции: сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%).
5. В (( )) можно записывать условия оболочки bash.
6. Стандартные имена переменныx: PATH, HOME, IFS, MAIL, TERM, LOGNAME.
7. Метасимволы – это специальные символы, имеющие особое значение, такие как подстановочный символ, символ повторения, символ несовпадения или диапазон символов.
8. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , ”.
9. 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

1. Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключённых в фигурные скобки.
2. Чтобы выяснить, является ли файл каталогом или обычным файлом, можно воспользоваться командой

test -f [путь до файла]

для проверки, является ли обычным файлом и

test -d [путь до файла]

для проверки, является ли он каталогом.

* + Команду set используют для вывода списка переменных окружения.
* Команду unset используют для удаления переменной из вашего окружения командной оболочки.
* Команда typeset является встроенной и предназначена для наложения ограничений на переменные.

1. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. 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 языка bash и их назначение:

* $0 — в этом переменной лежит путь и имя скрипта, который запустил пользователь;
* $# — количество параметров, переданных скрипту из командной строки;
* $? — код возврата (exit code, result code), с которым завершилась предыдущая команда;
* $! — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
* $$ - номер процесса, в котором выполняется данный скрипт.

# Выводы

Я научилась писать небольшие командный файлы для решения различных вопрсов.