Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №5

по Операционной системе Linux

Программирование на SHELL. Использование командных файлов

Студент Лобов М.Ю.

Группа АИ-18

Руководитель Кургасов В.В.

Доцент, к.п.н.

Липецк 2020 г.

Оглавление

Цель работы

Изучение основных возможностей языка программирования Shell с целью автоматизации процесса администрирования системы за счёт написания и использования командных файлов.

Задание кафедры

1. Используя команды ECHO, PRINTF вывести информационные сообщения на экран.

2. Присвоить переменной A целочисленное значение. Просмотреть значение переменной A.

3. Присвоить переменной B значение переменной A. Просмотреть значение переменной B.

4. Присвоить переменной C значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной.

5. Присвоить переменной D значение “имя команды”, а именно, команды DATE. Выполнить эту команду, используя значение переменной.

6. Присвоить переменной E значение “имя команды”, а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной.

7. Присвоить переменной F значение “имя команды”, а именно сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.

Написать скрипты, при запуске которых выполняются следующие действия:

8. Программа запрашивает значение переменной, а затем выводит значение этой переменной.

9. Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной.

10. Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды а) EXPR; б) BC).

11. Вычислить объем цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран.

12. Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки.

13. Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.

14. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно.

15. Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения.

16. Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран.

17. Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются.

18. В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc.

19. Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение.

20. Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог, просматривается содержимое файла.

21. Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются соответствующие сообщения (использовать имена файлов и позиционные параметры).

22. Если файл запуска программы найден, программа запускается (по выбору).

23. В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля, содержимое файла сортируется по первому столбцу по возрастанию, отсортированная информация помещается в другой файл, содержимое которого затем отображается на экране.

24. Командой TAR осуществляется сборка всех текстовых файлов текущего каталога в один архивный файл my.tar, после паузы просматривается содержимое файла my.tar, затем командой GZIP архивный файл my.tar сжимается.

25. Написать скрипт с использованием функции, например, функции, суммирующей значения двух переменных.

Ход работы

Создадим файл с помощью команды **nano script**, в этом файле и будем записывать сценарии для выполнения заданий.

1. Используя команды ECHO, PRINTF вывести информационные сообщения на экран.

Напишем следующий скрипт:

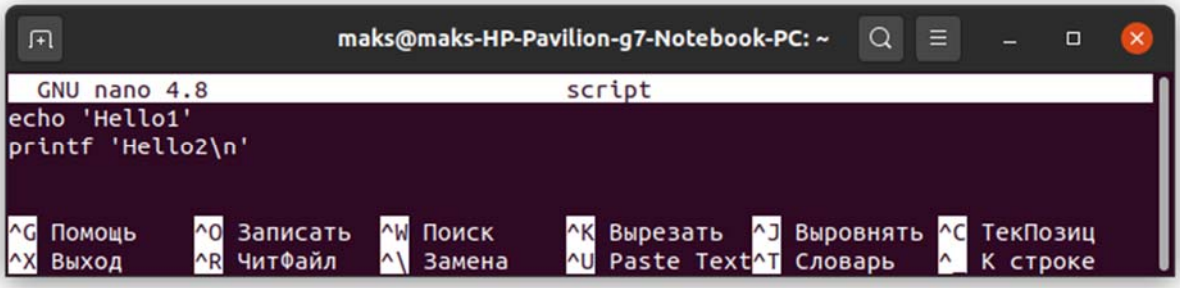

Рисунок 1.1 – Скрипт для задания 1

После запуска данного сценария увидим:

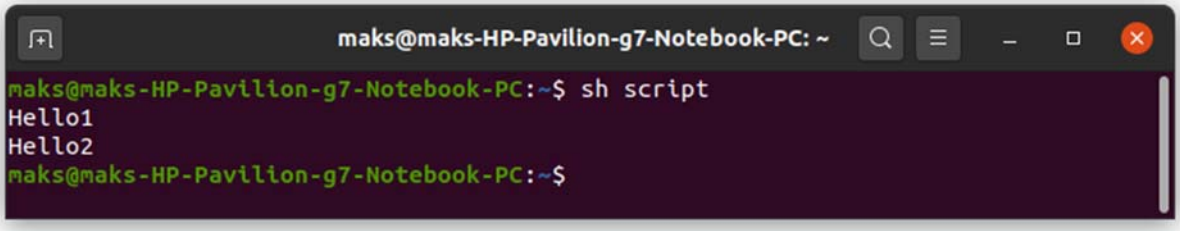

Рисунок 1.2 – Результат выполнения сценария

2. Присвоить переменной A целочисленное значение. Просмотреть значение переменной A.

Напишем сценарий:

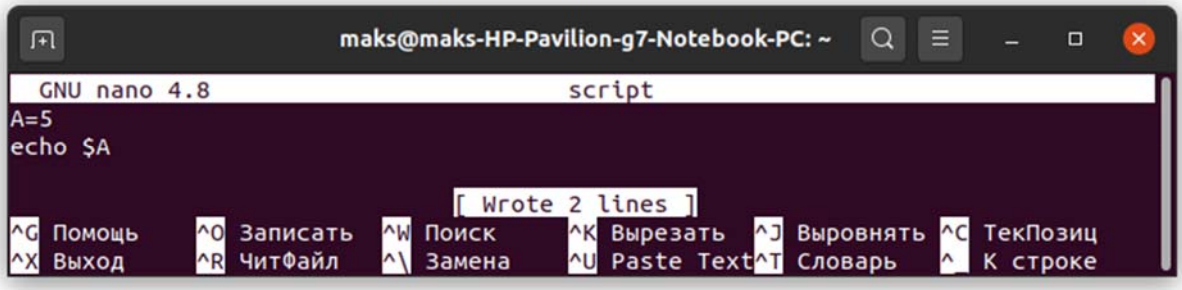

Рисунок 2.1 – Скрипт для задания 2

Запустим сценарий:

Рисунок 2.2 – Результат выполнения сценария

3. Присвоить переменной B значение переменной A. Просмотреть значение переменной B.

Напишем сценарий:

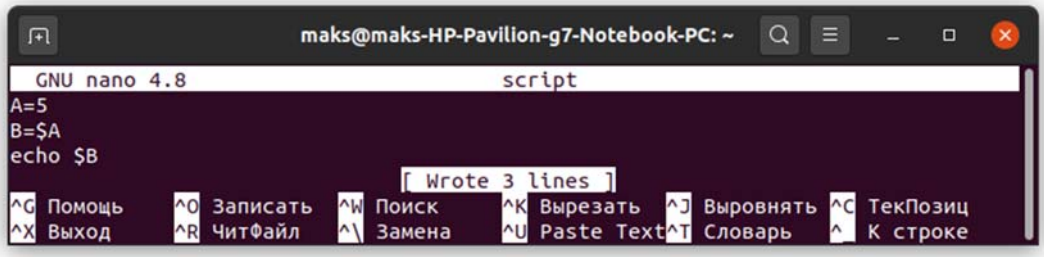

Рисунок 3.1 – Скрипт для задания 3

Выполним сценарий:

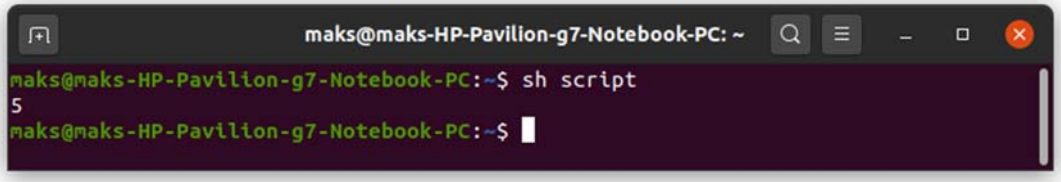

Рисунок 3.2 – Результат выполнения сценария

4. Присвоить переменной C значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной.

Сделаем это с использованием встроенной переменной **HOME**:

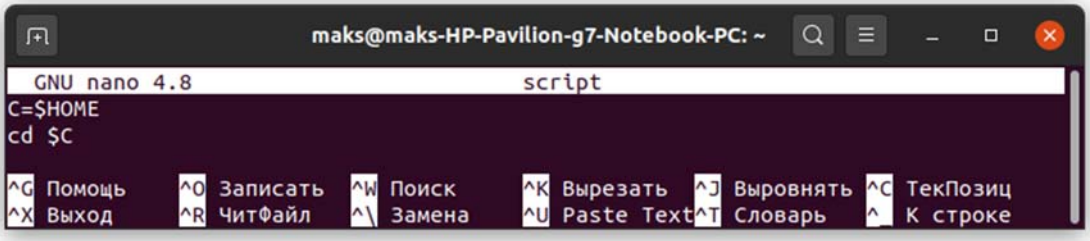

Рисунок 4.1 – Скрипт для задания 4

Проверим работу сценария:

Рисунок 4.2 – Результат выполнения сценария

5. Присвоить переменной D значение “имя команды”, а именно, команды DATE. Выполнить эту команду, используя значение переменной.

Выполним задание с использованием встроенной переменной **DATE**:

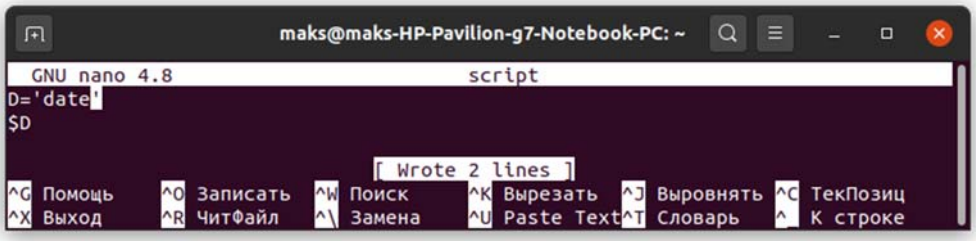

Рисунок 5.1 – Скрипт для задания 5

Исполним сценарий:

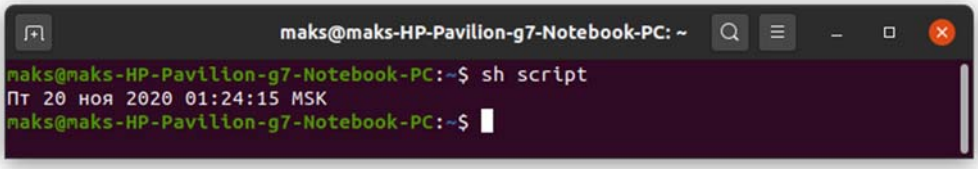

Рисунок 5.2 – Результат выполнения сценария

6. Присвоить переменной E значение “имя команды”, а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной.

Присвоим переменной значение команды **cat**:

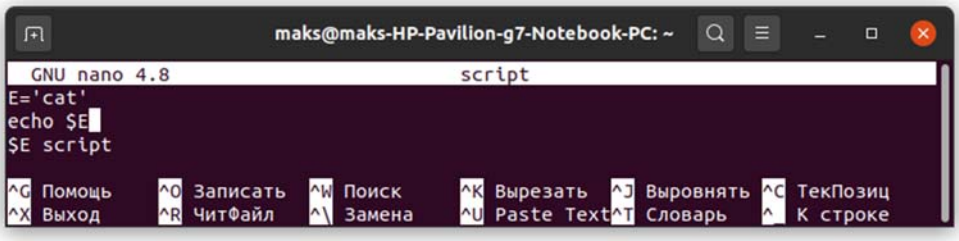

Рисунок 6.1 – Скрипт для задания 6

Выполним сценарий:

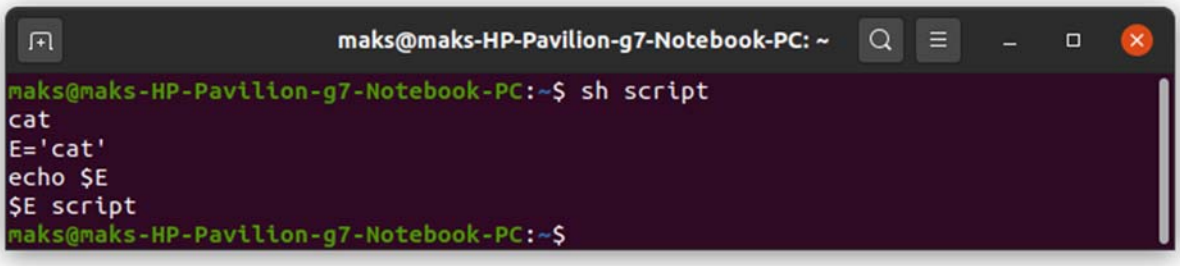

Рисунок 6.2 – Результат выполнения сценария

7. Присвоить переменной F значение “имя команды”, а именно сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.

Присвоим переменной F значение **sort**:

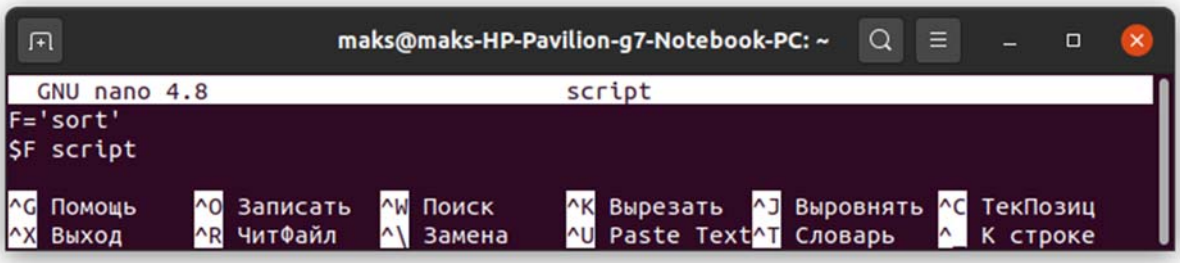

Рисунок 7.1 – Скрипт для задания 7

Исполним сценарий:

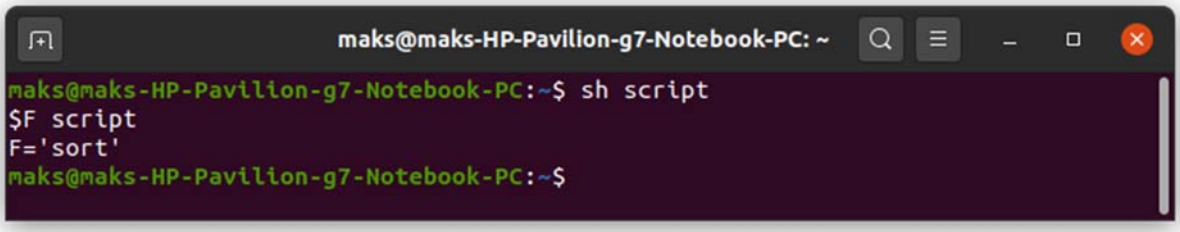

Рисунок 7.2 – Результат выполнения сценария

8. Программа запрашивает значение переменной, а затем выводит значение этой переменной.

Запросить у пользователя значение переменной позволяет команда **read**. Используем эту команду в написании скрипта:

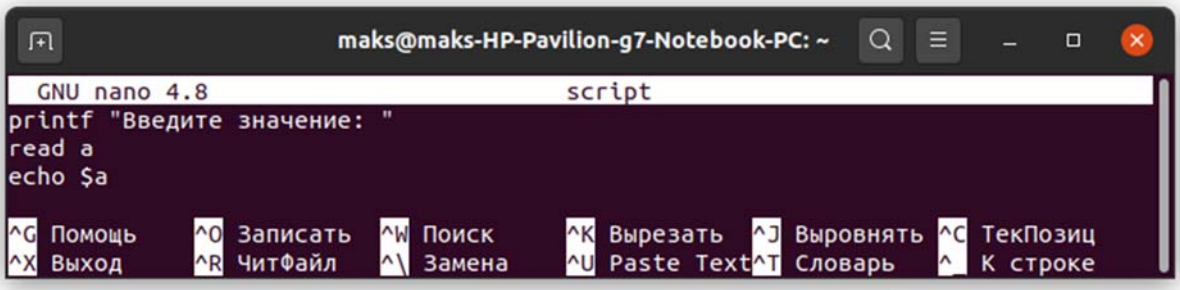

Рисунок 8.1 – Скрипт для задания 8

Исполним сценарий:

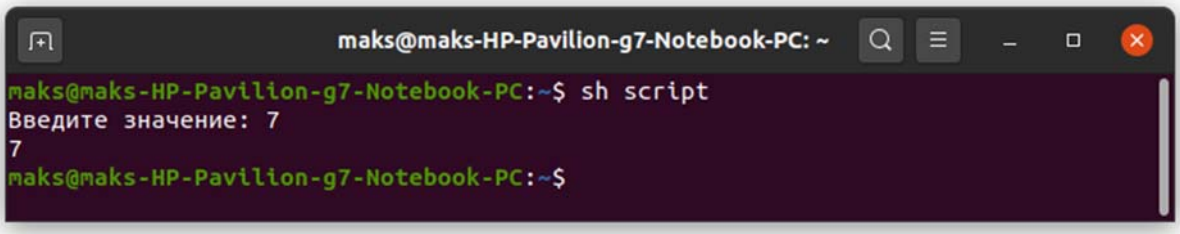

Рисунок 8.2 – Результат выполнения сценария

9. Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной.

Сделаем по примеру предыдущего задания:

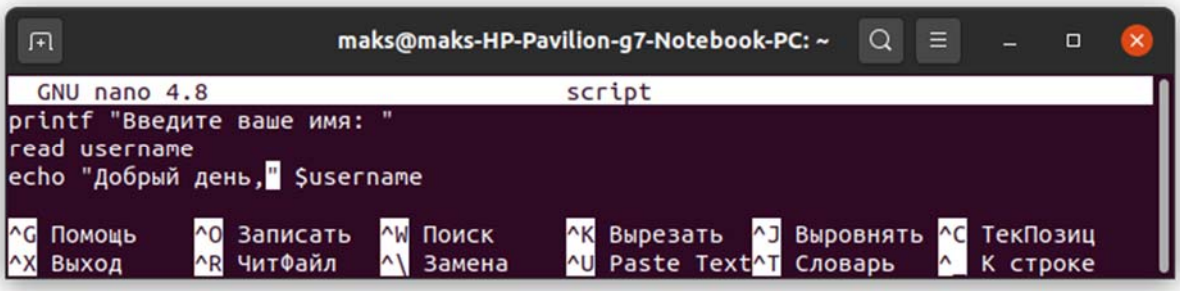

Рисунок 9.1 – Скрипт для задания 9

Запустим файл сценария:

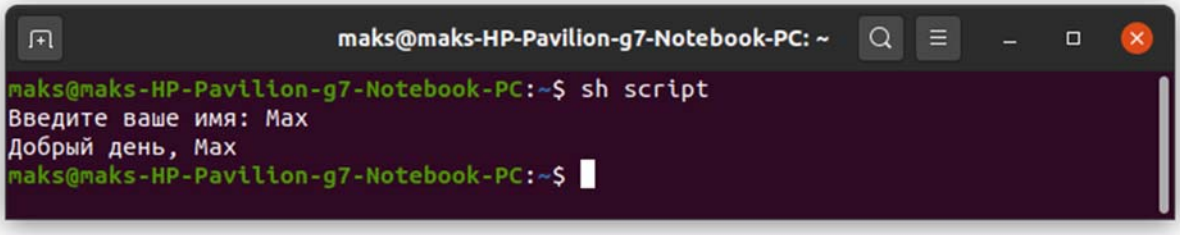

Рисунок 9.2 – Результат выполнения сценария

10. Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды а) EXPR; б) BC).

Исполним оба варианта выполнения задания в одном файле:

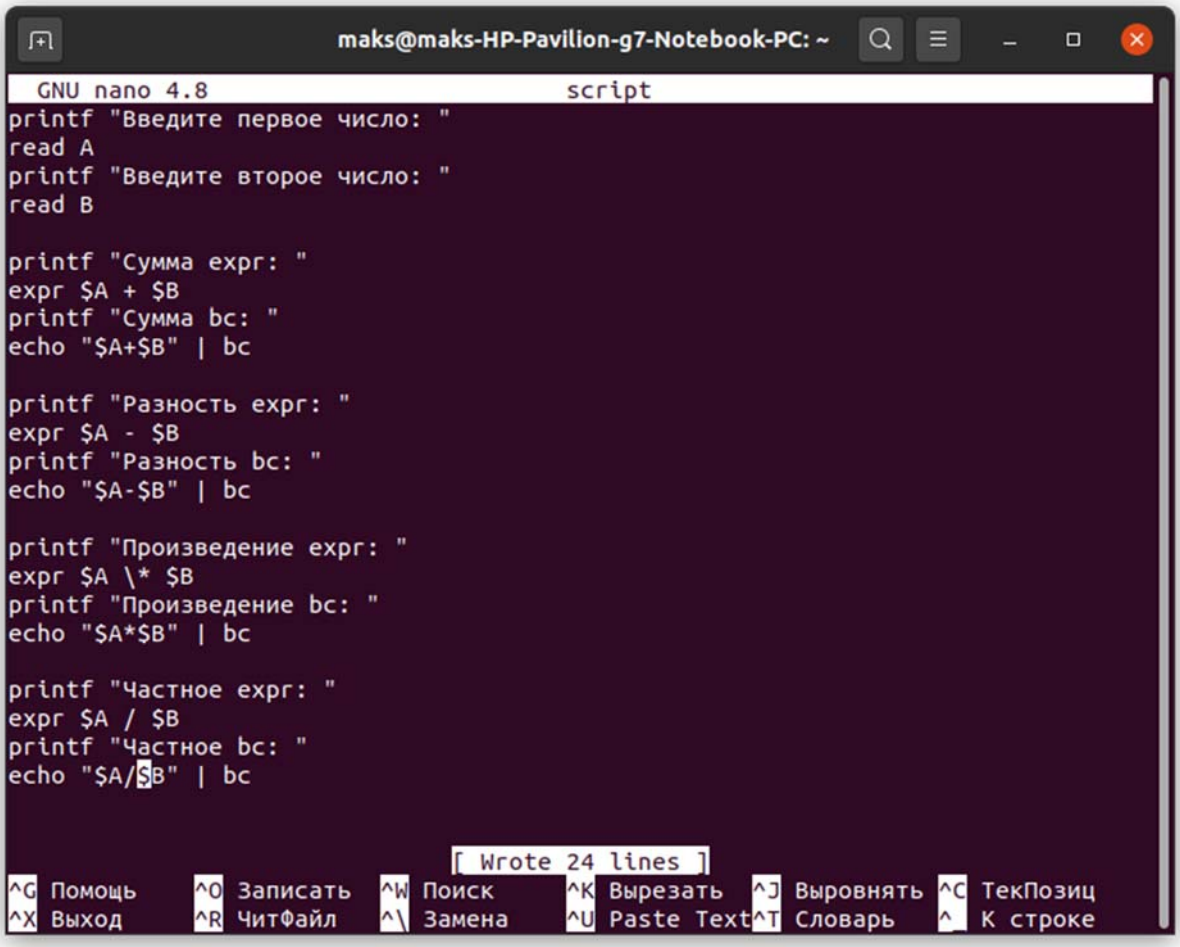

Рисунок 10.1 – Скрипт для задания 10

Исполним скрипт:

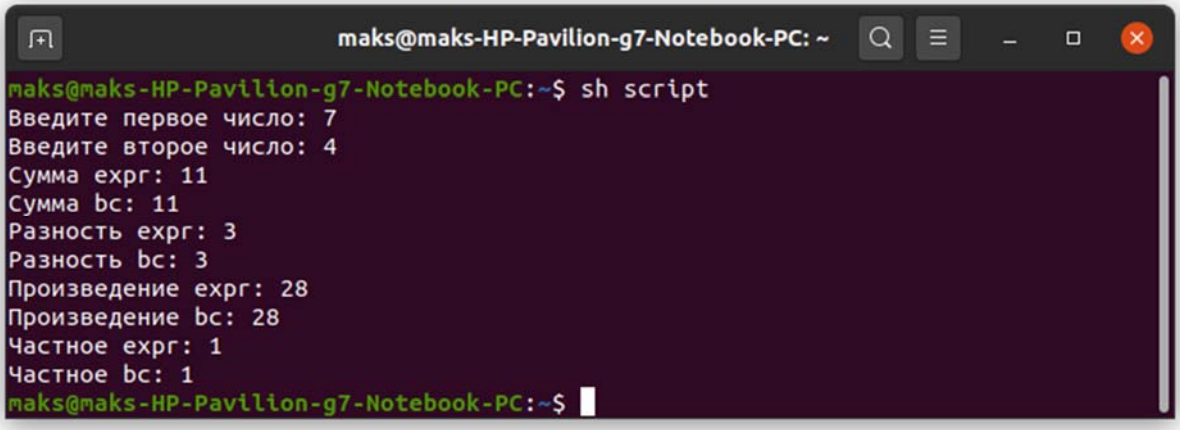

Рисунок 10.2 – Результат выполнения сценария

11. Вычислить объем цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран.

Формула объёма цилиндра иммет вид $V = \pi R^2 h$. Реализуем её:

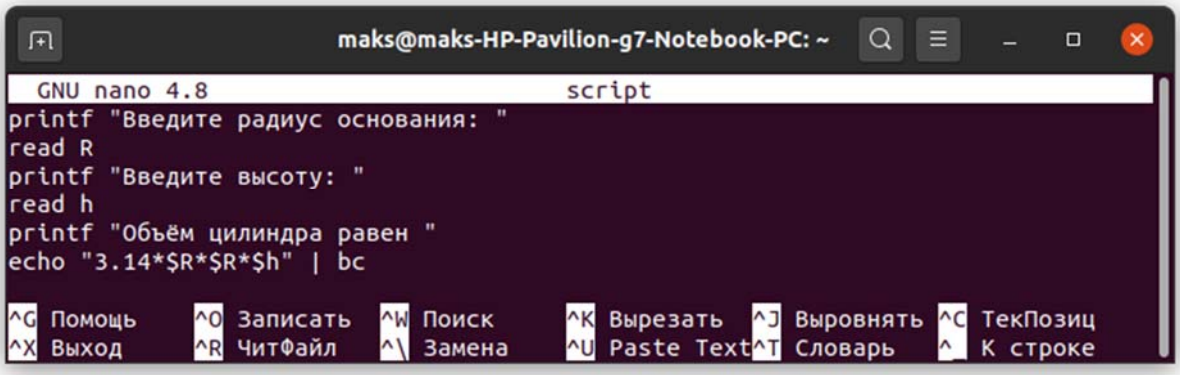

Рисунок 11.1 – Скрипт для выполнения задания 11

Запустим сценарий:

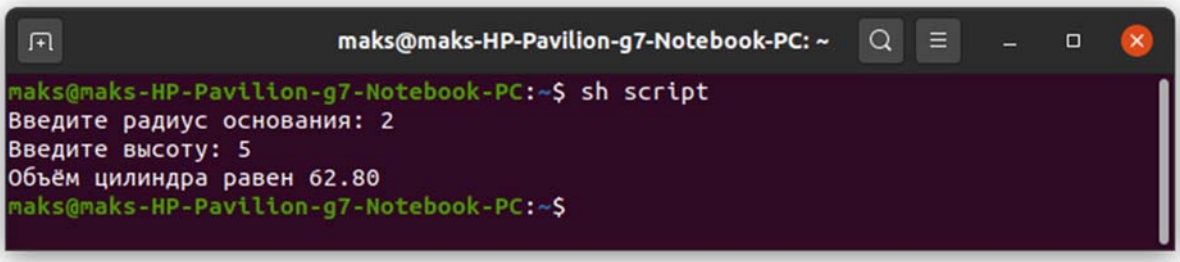

Рисунок 11.2 – Результат выполнения сценария

12. Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки.

Для обозначения количества аргументов командной строки используем переменную **$#**. А для обращения к аргументу командной строки будем использовать **$<num>**, где **num** – номер аргумента. Напишем скрипт:

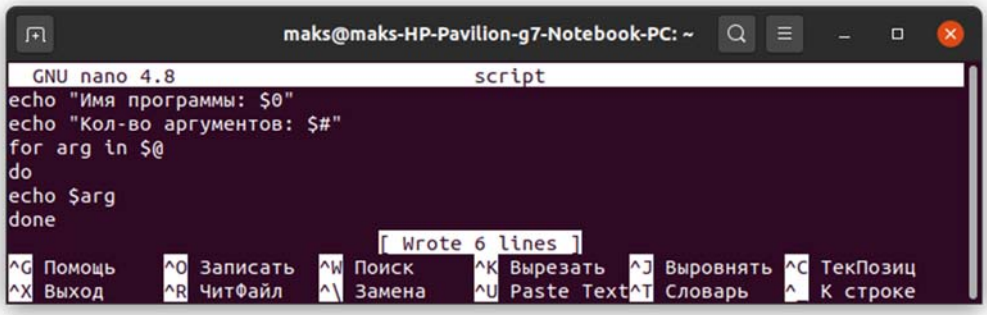

Рисунок 12.1 – Скрипт для задания 12

Исполним сценарий:

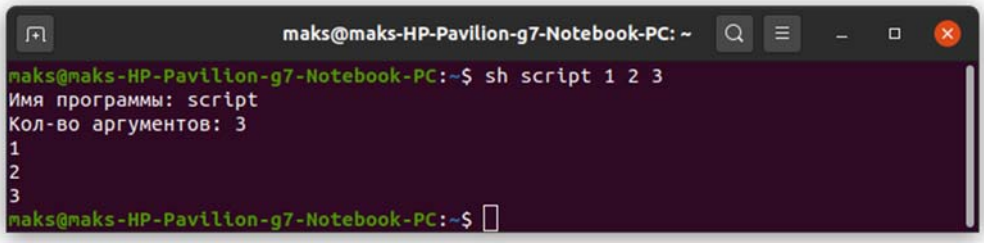

Рисунок 12.2 – Результат выполнения сценария

13. Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.

Напишем сценарий:

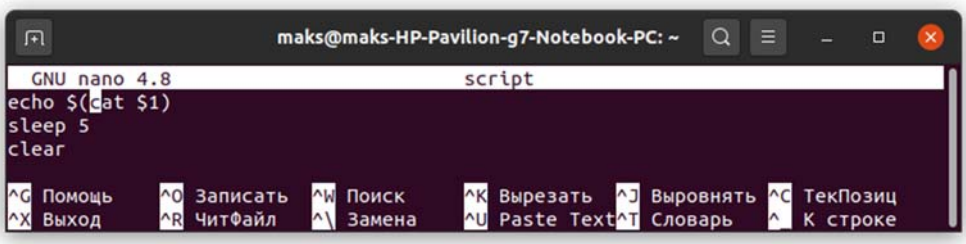

Рисунок 13.1 – Скрипт для задания 13

Запустим сценарий на выполнение:

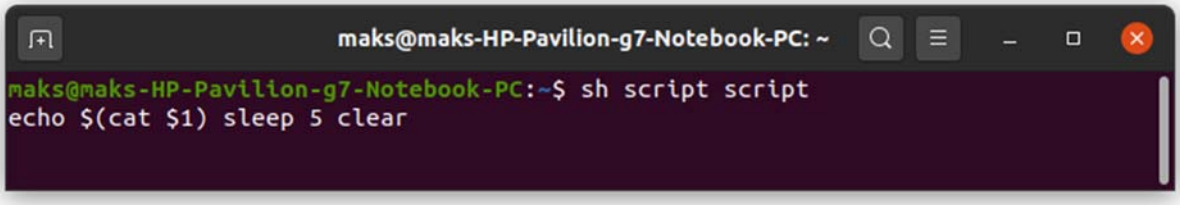

Рисунок 13.2 – Результат выполнения сценария

14. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно.

Для поиска в данной директории используем конструкцию **./***. Затем проверим, не является ли выбранный файл директорией, чтобы наш скрипт сработал для файлов:

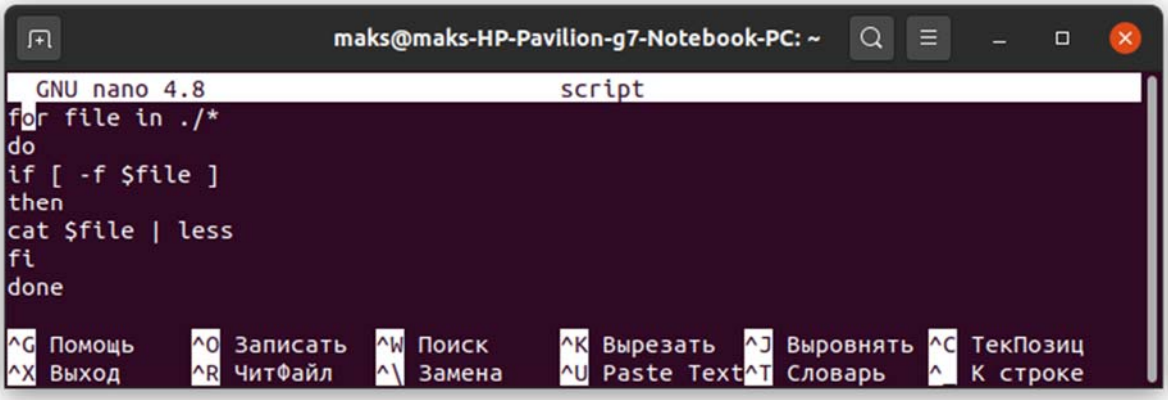

Рисунок 14.1 – Скрипт для задания 14

Запустим скрипт:

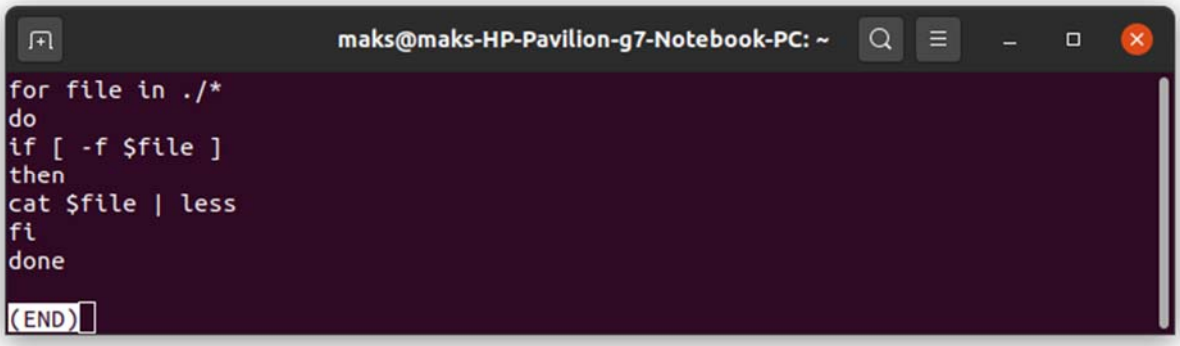

Рисунок 14.2 – Результат выполнения сценария

15. Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения.

Для сравнения чисел в оболочке **bash** используется конструкция **–eq** (от англ. *equal – равно*). Используем это в написании сценария:

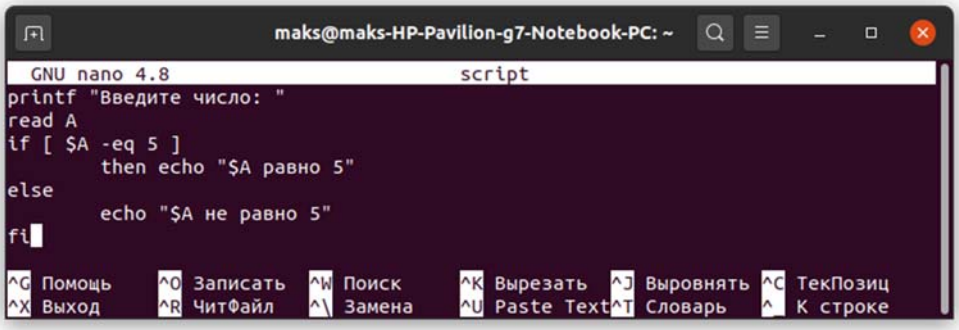

Рисунок 15.1 – Скрипт для задания 15

Выполним скрипт:

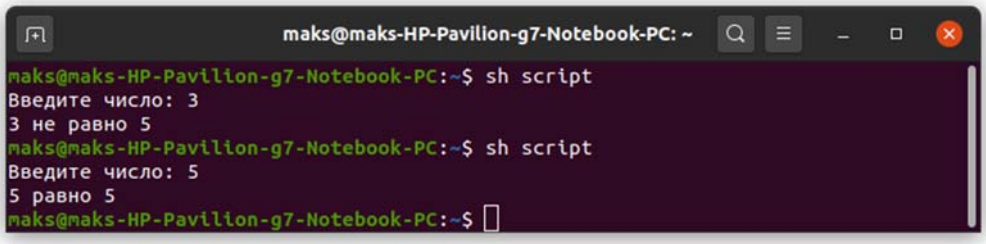

Рисунок 15.2 – Результат выполнения сценария

16. Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран.

Для того, чтобы год был високосным, он должен делиться без остатка на 4. Реализуем это:

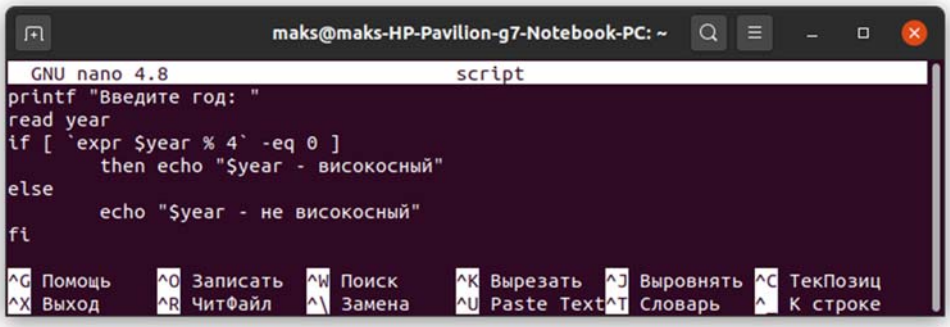

Рисунок 15.1 – Скрипт для задания 16

Исполним данный сценарий:

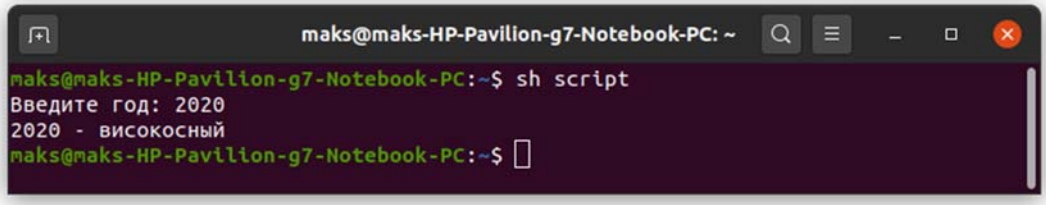

Рисунок 16.2 – Результат выполнения сценария

17. Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются.

Напишем сценарий:

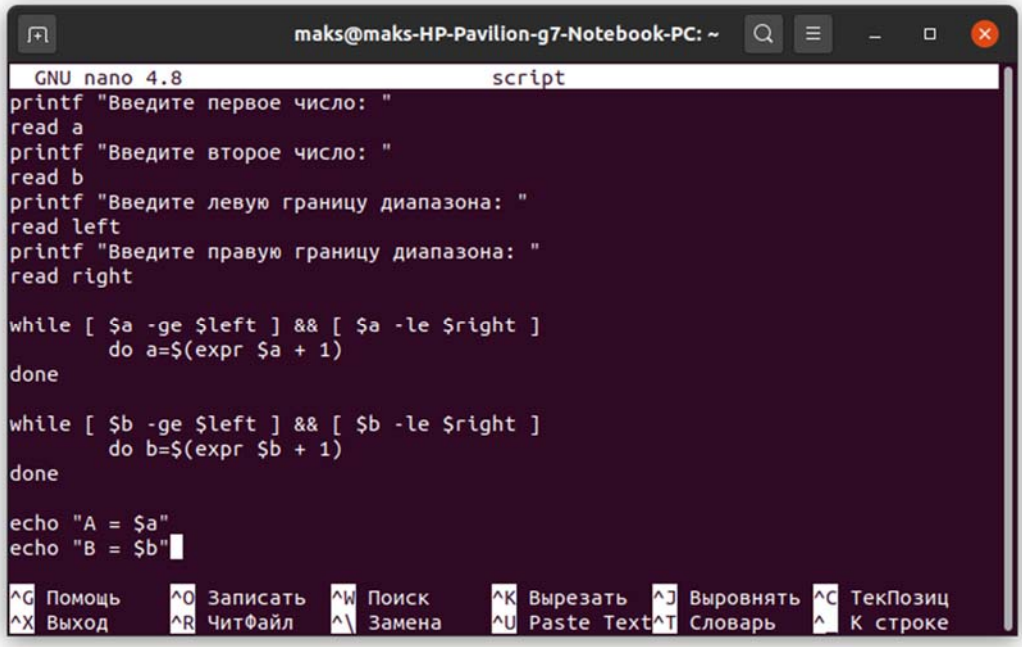

Рисунок 17.1 – Скрипт для задания 17

Запустим скрипт на выполнение:

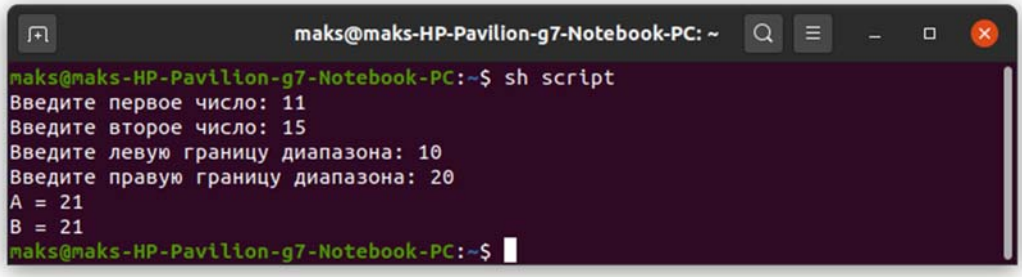

Рисунок 17.2 – Результат выполнения сценария

18. В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc.

Для сравнивания двух строк используется оператор =. Напишем скрипт:

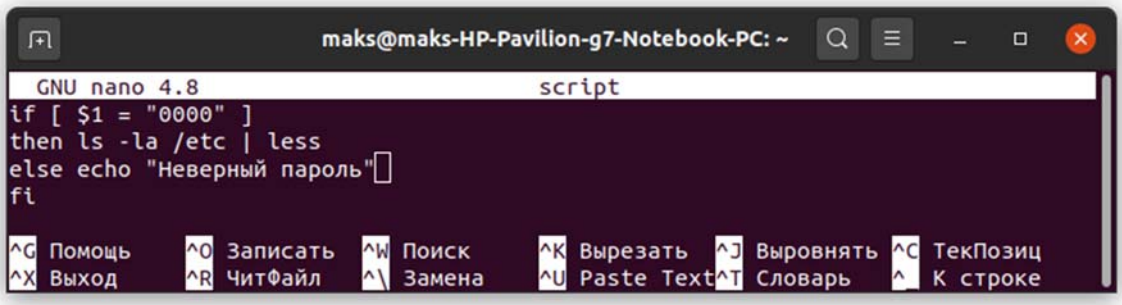

Рисунок 18.1 – Скрипт для задания 18

Исполним сценарий:

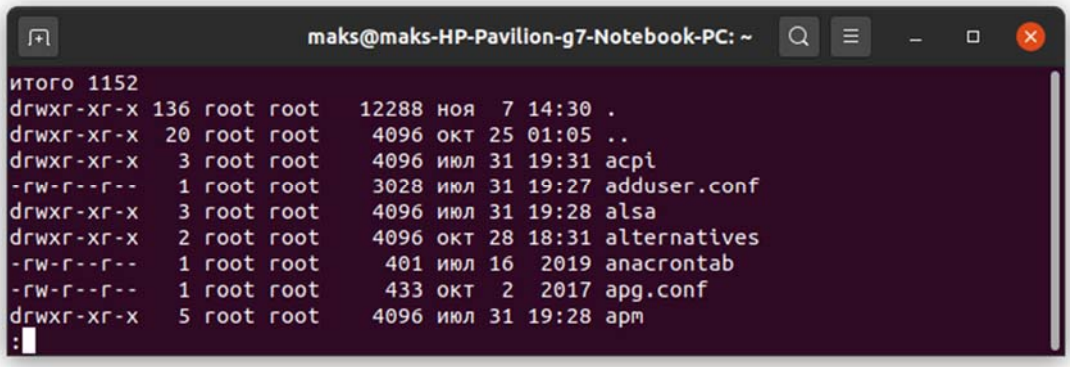

Рисунок 18.2 – Результат выполнения сценария

19. Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение.

Чтобы проверить файл на существование, используем конструкцию **–e**:

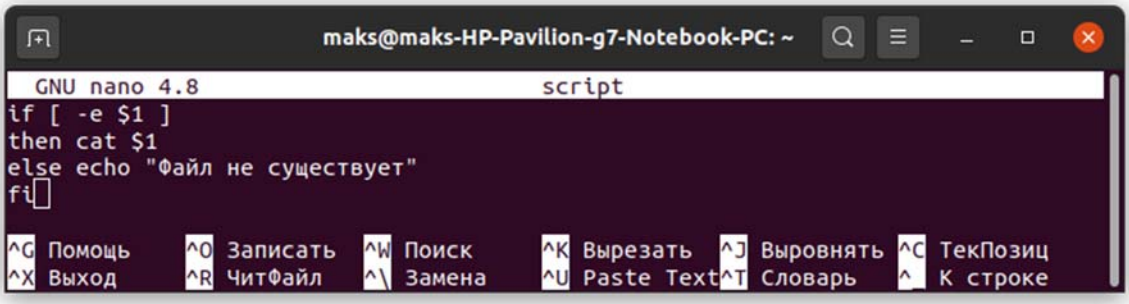

Рисунок 19.1 – Скрипт для задания 19

Запустим сценарий:

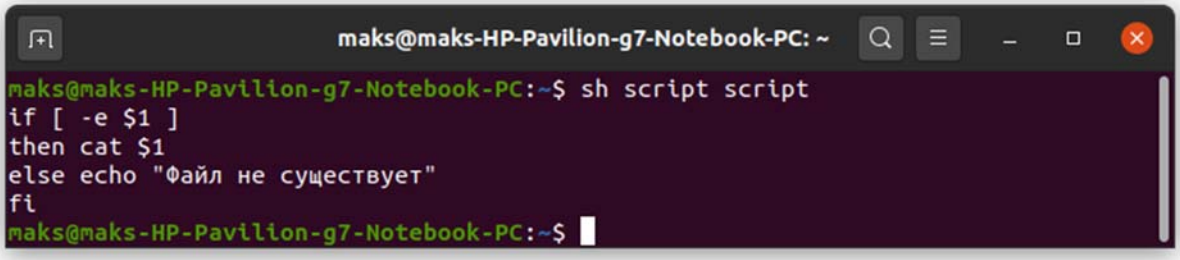

Рисунок 19.2 – Результат выполнения сценария

20. Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог, просматривается содержимое файла.

Чтобы проверить, доступен ли файл или каталог для чтения, используется опция **–r**. Напишем скрипт с использованием данной конструкции:

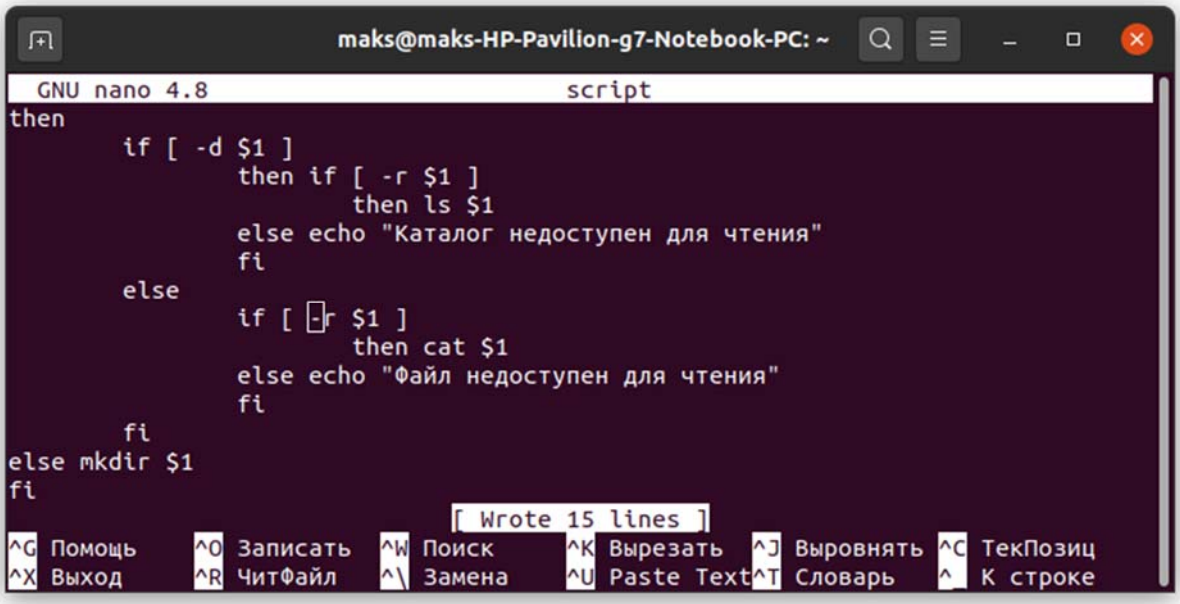

Рисунок 20.1 – Скрипт для задания 20

Теперь проверим наш сценарий:

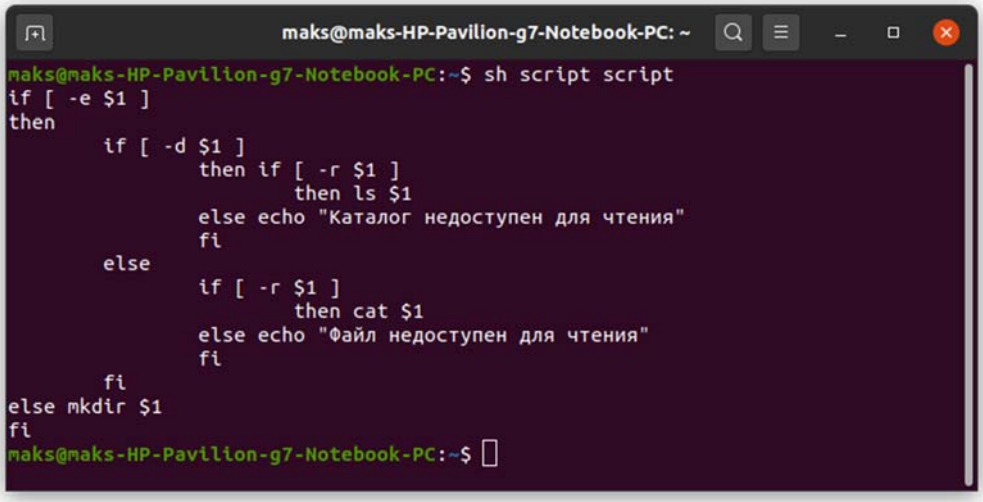

Рисунок 20.2 – Результат выполнения сценария

21. Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются соответствующие сообщения (использовать имена файлов и позиционные параметры).

Для проверки доступности файла для записи используется конструкция **–r**. Сначала реализуем задание, спросив названия файлов у пользователя:

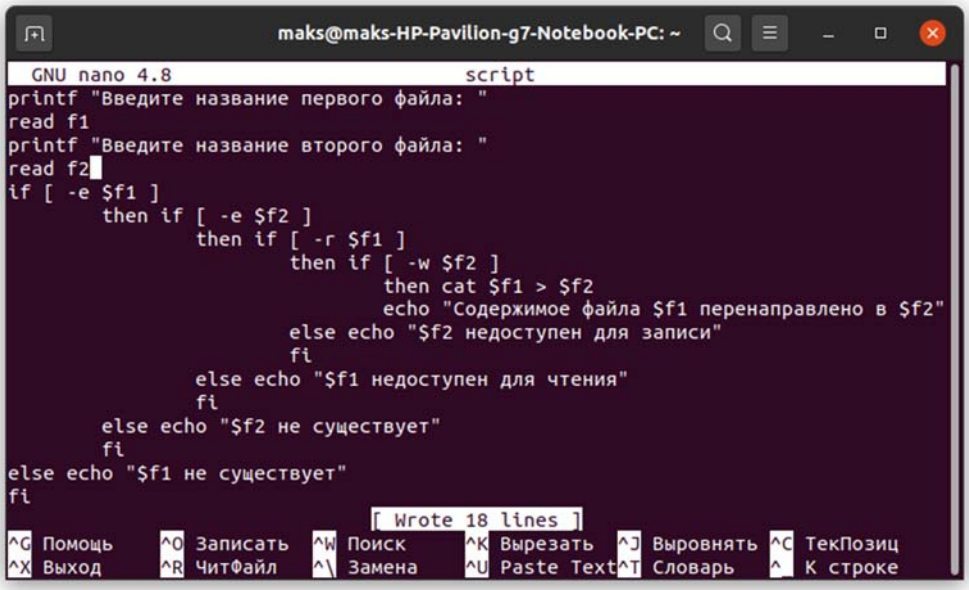

Рисунок 21.1 – Скрипт для задания 21 (через имена файлов)

Выполним скрипт:

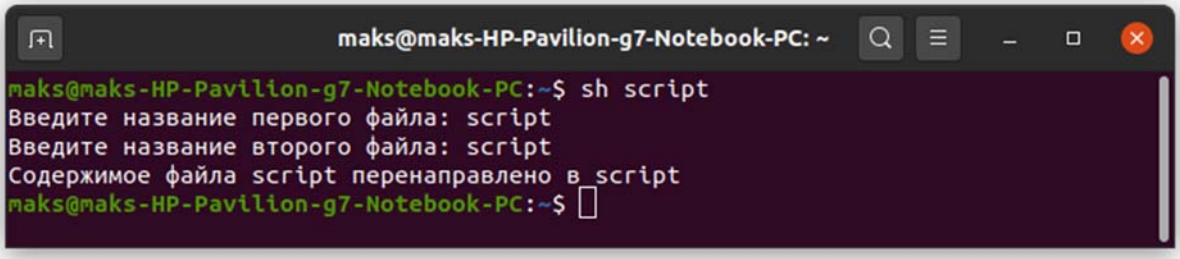

Рисунок 21.2 – Результат выполнения сценария (через имена файлов)

Теперь передадим имена файлов в качестве аргументов командной строки и немного модифицируем наш сценарий:

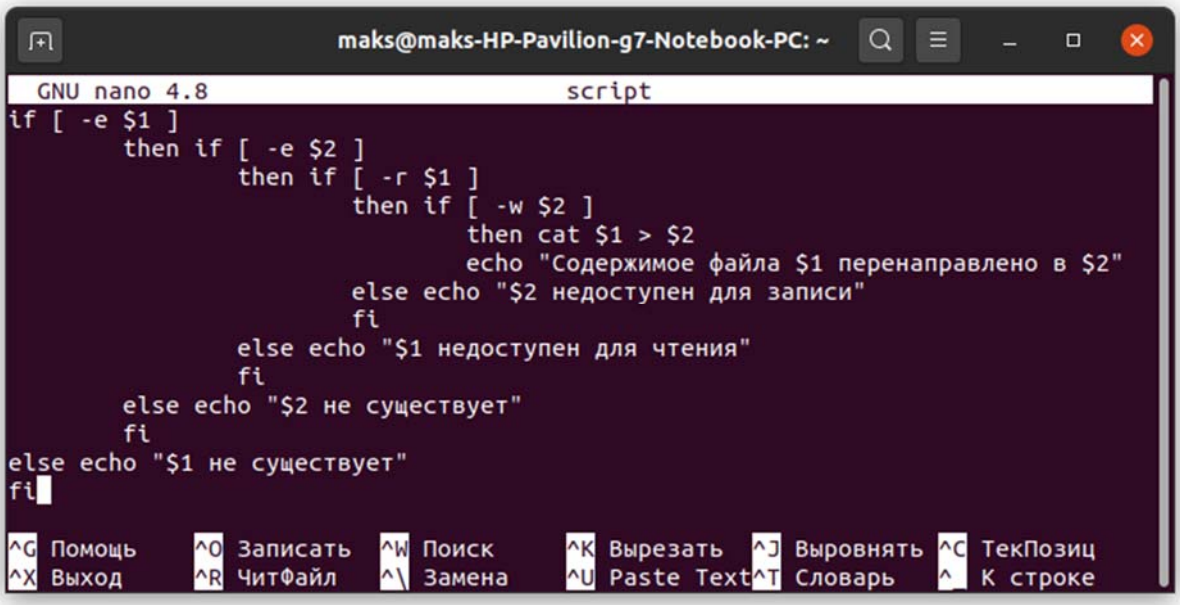

Рисунок 21.3 – Скрипт для задания 21 (через позиционные параметры)

Запустим скрипт:

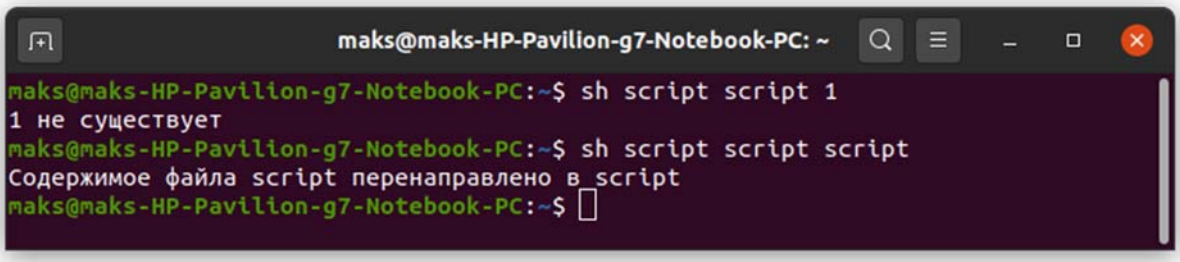

Рисунок 21.4 – Результат выполнения сценария (через позиционные параметры)

22. Если файл запуска программы найден, программа запускается (по выбору).

Проверим файл, переданный в качестве аргумента командной строки, на существование и доступность для исполнения (опция **–x**):

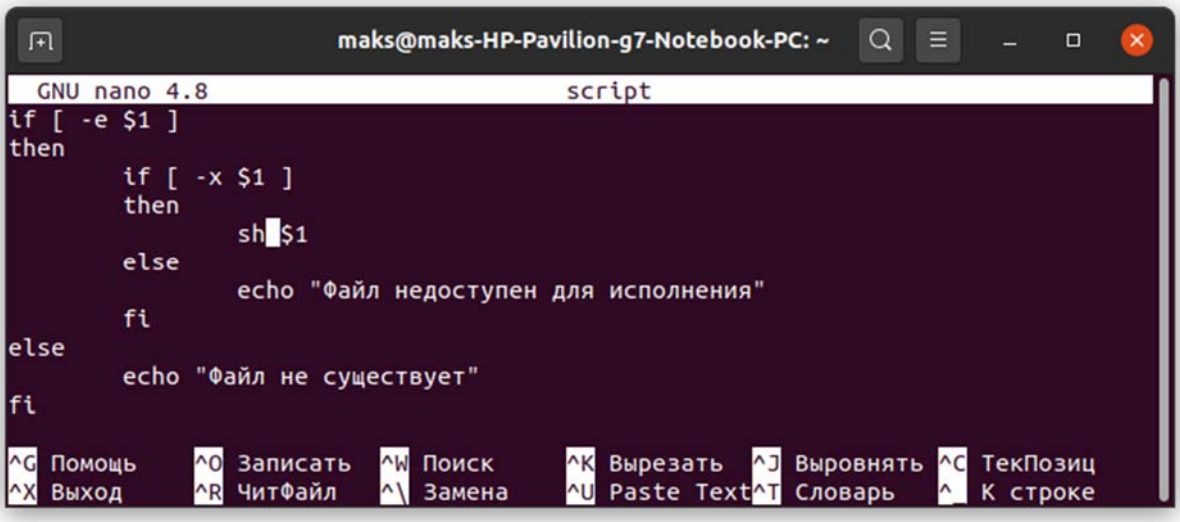

Рисунок 22.1 – Скрипт для задания 22

Затем создадим новый файл сценария, который нам придётся запустить:

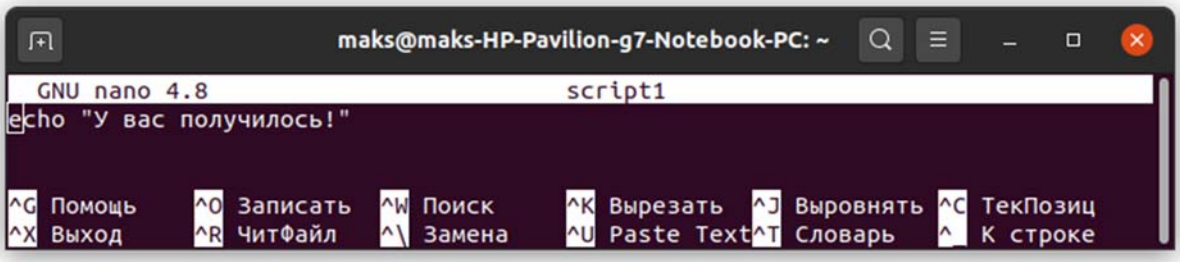

Рисунок 22.2 – Содержание файла script1

Ну и, наконец, запустим наш основной сценарий:

Рисунок 22.3 – Результат выполнения сценария

23. В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля, содержимое файла сортируется по первому столбцу по возрастанию, отсортированная информация помещается в другой файл, содержимое которого затем отображается на экране.

Для проверки файла на содержание в нём хотя бы одного символа используется конструкция **–s**. Чтобы отсортировать данные в файле по определённому столбцу, используем опцию **–k<num>** (где **num** – номер столбца) команды **sort**:

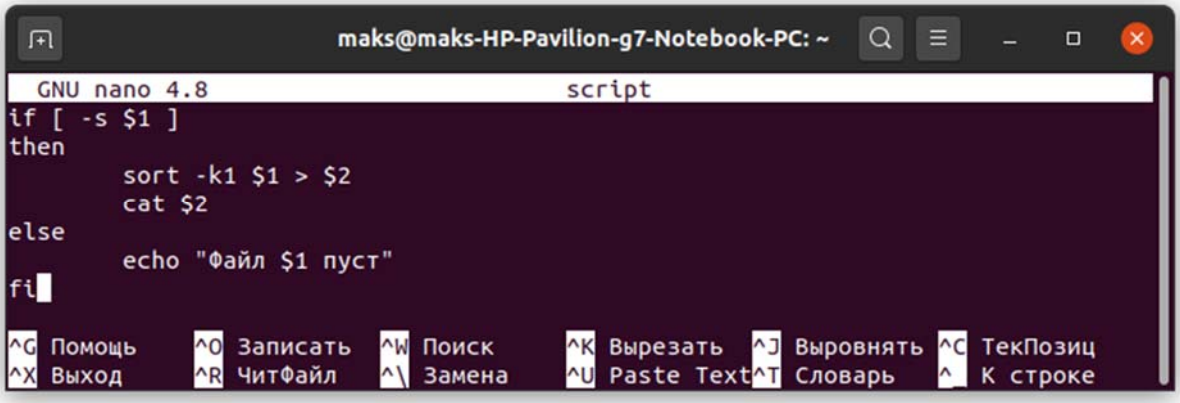

Рисунок 23.1 – Скрипт для задания 23

Исполним скрипт:

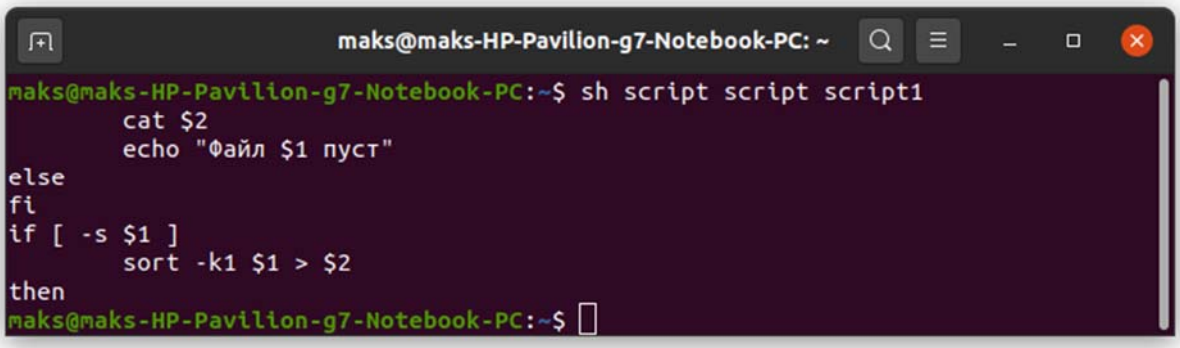

Рисунок 23.2 – Результат выполнения сценария

24. Командой TAR осуществляется сборка всех текстовых файлов текущего каталога в один архивный файл my.tar, после паузы просматривается содержимое файла my.tar, затем командой GZIP архивный файл my.tar сжимается.

Сначала найдём в данном каталоге с помощью команды **find** нужные файлы (параметр **–maxdepth 1** говорит о том, что в подкаталогах искать не нужно, а параметр **–type f** – о том, что ищем только файлы, а не каталоги). Затем с помощью команды **tar –cf** собираем найденные файлы в один архив. Ну и выводим содержимое с помощью команды **tar –tf**:

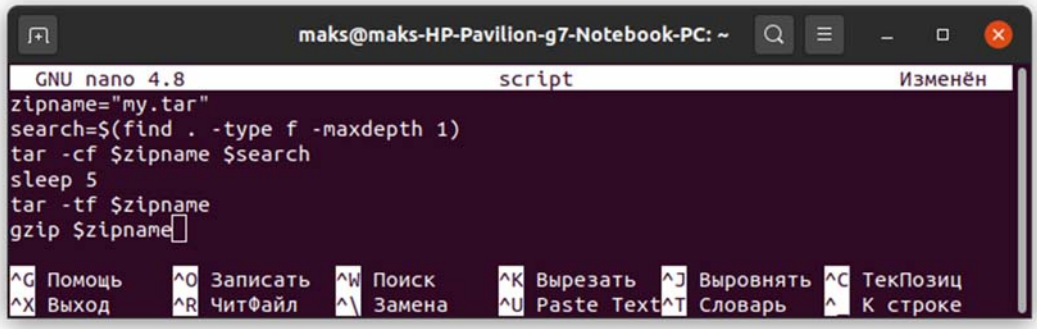

Рисунок 24.1 – Скрипт для задания 24

Выполним скрипт:

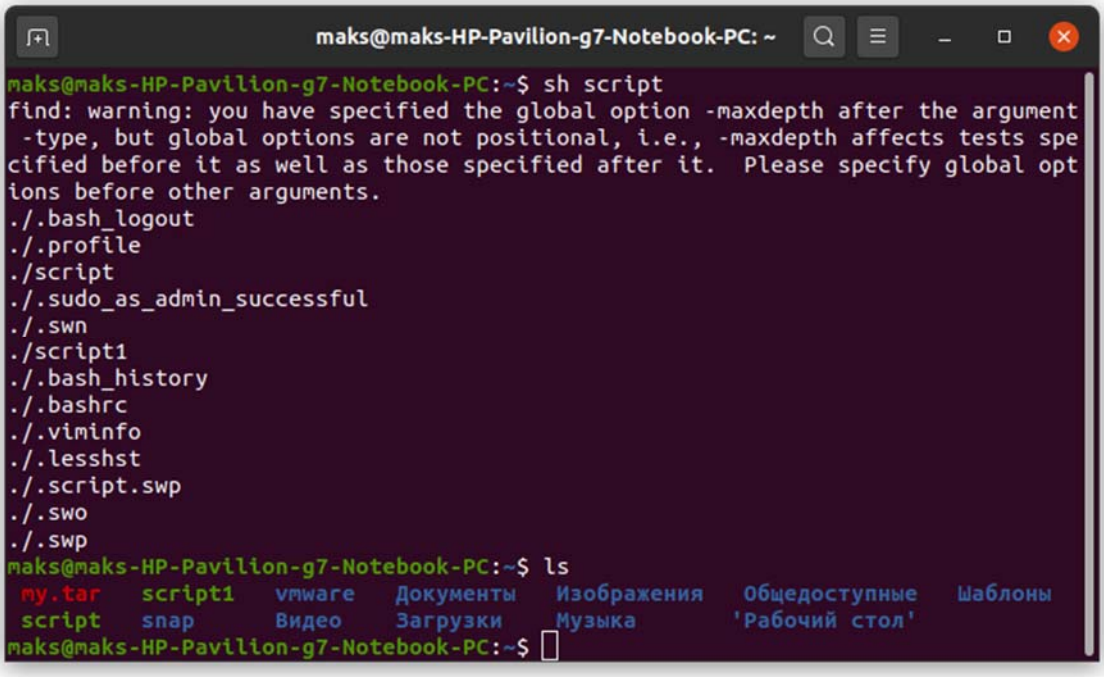

Рисунок 24.2 – Результат выполнения сценария

25. Написать скрипт с использованием функции, например, функции, суммирующей значения двух переменных.

Напишем сценарий:

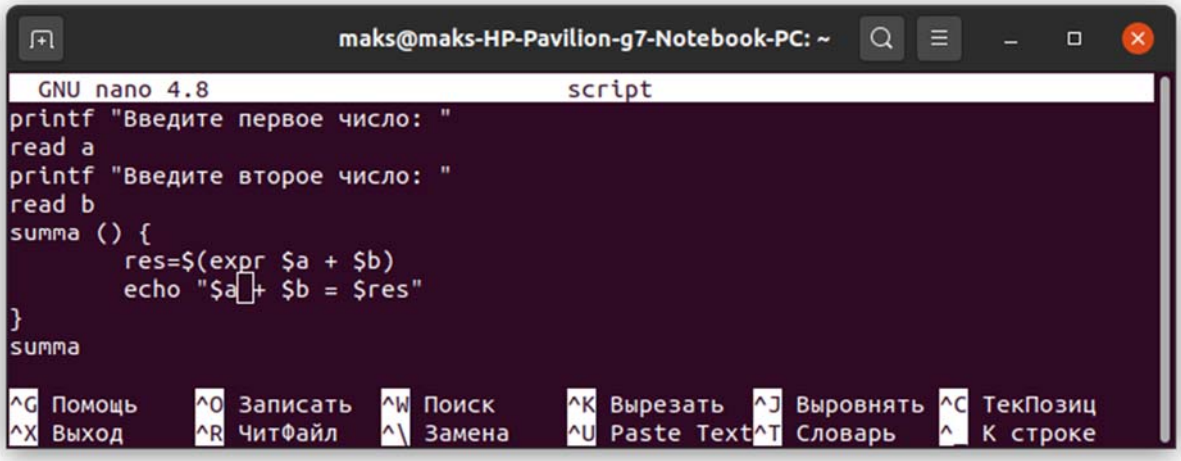

Рисунок 25.1 – Скрипт для задания 25

Выполним скрипт:

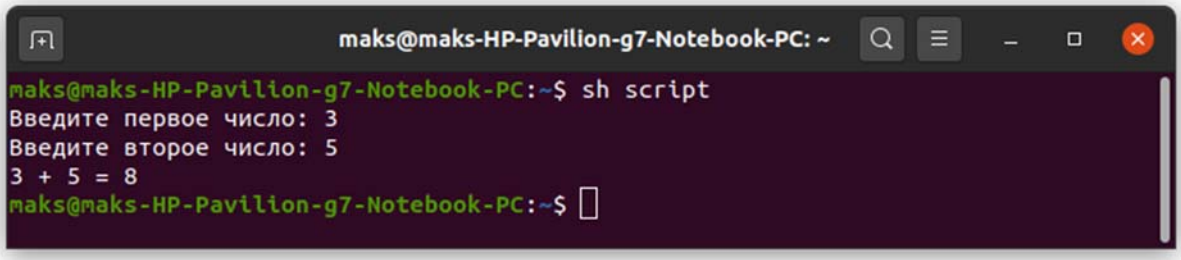

Рисунок 25.2 – Результат выполнения сценария

Вывод

В ходе выполнения лабораторной работы были изучены основные возможности языка программирования Shell с целью автоматизации процесса администрирования системы за счёт написания и использования командных файлов.