Отчёта по лабораторной работе №11

Лабораторная работа №11. Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Надежда Александровна Рогожина

Содержание

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# 2 Задание

1. Используя команды getopts grep, написать командный файл, который анализирует командную строку с ключами:

* -iinputfile — прочитать данные из указанного файла;
* -ooutputfile — вывести данные в указанный файл;
* -pшаблон — указать шаблон для поиска;
* -C — различать большие и малые буквы;
* -n — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом -p.

1. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции exit(n), передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды $?, выдать сообщение о том, какое число было введено.
2. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до 𝑁 (например 1.tmp, 2.tmp, 3.tmp,4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).
3. Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).

# 3 Теоретическое введение

## 3.1 Метасимволы

При перечислении имён файлов текущего каталога можно использовать следующие символы:

* + — соответствует произвольной, в том числе и пустой строке;
* ? — соответствует любому одинарному символу;
* [c1-c1] — соответствует любому символу, лексикографически находящемуся между символами c1 и с2.

Например, - echo \* — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды ls; - ls *.c — выведет все файлы с последними двумя символами, совпадающими с .c. - echo prog.? — выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются prog.. - [a-z]* — соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

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

## 3.2 Использование команды getopts

Весьма необходимой при программировании является команда getopts, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий:

getopts option-string variable [arg ... ]

*Флаги* — это опции командной строки, обычно помеченные знаком минус; Например, для команды ls флагом может являться -F. Иногда флаги имеют аргументы, связанные с ними. Программы интерпретируют флаги, соответствующим образом изменяя своё поведение.

Строка опций option-string — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда getopts может распознать аргумент, то она возвращает истину. Принято включать getopts в цикл while и анализировать введённые данные с помощью оператора case.

## 3.3 Управление последовательностью действий в командных файлах

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования bash предоставляет возможность использовать такие управляющие конструкции, как for, case, if и while.

С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования bash. Поэтому при описании языка программирования bash термин оператор будет использоваться наравне с термином команда.

Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда test, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения. Так например, команда

test -f file

возвращает нулевой код завершения (истина), если файл file существует, и ненулевой код завершения (ложь) в противном случае:

* test s –— истина, если аргумент s имеет значение истина;
* test -f file — истина, если файл file существует;
* test -i file — истина, если файл file доступен по чтению;
* test -w file — истина, если файл file доступен по записи;
* test -e file — истина, если файл file — исполняемая программа;
* test -d file — истина, если файл file является каталогом

## 3.4 Оператор выбора case

Оператор выбора case реализует возможность ветвления на произвольное число ветвей. Эта возможность обеспечивается в большинстве современных языков программирования, предполагающих использование структурного подхода.

В обобщённой форме оператор выбора case выглядит следующим образом:

case имя in  
шаблон1) список-команд;;  
шаблон2) список-команд;;  
...  
esac

Выполнение оператора выбора case сводится к тому, что выполняется последовательность команд (операторов), задаваемая списком список-команд, в строке, для которой значение переменной имя совпадает с шаблоном. Поскольку метасимвол \* соответствует произвольной, в том числе и пустой, последовательности символов, то его можно использовать в качестве шаблона в последней строке перед служебным словом esac. В этом случае реализуются все действия, которые необходимо произвести, если значение переменной имя не совпадает ни с одним из шаблонов, заданных в предшествующих cтроках.

# 4 Выполнение лабораторной работы

Для выполнения лабораторной работы будем пользоваться текстовым редактором Emacs.

1. Используя команды getopts, grep, написать командный файл, который анализирует командную строку с ключами:

* -iinputfile — прочитать данные из указанного файла;
* -ooutputfile — вывести данные в указанный файл;
* -pшаблон — указать шаблон для поиска;
* -C — различать большие и малые буквы;
* -n — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом -p (рис. [1](#fig:001),[2](#fig:002),[3](#fig:003)):

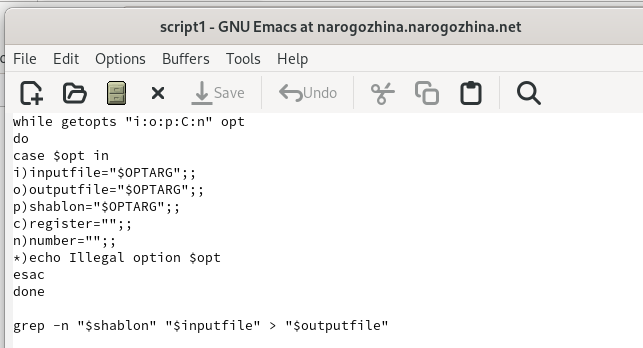


Figure 1: Скрипт №1

После первого запуска, я подправила текст скрипта (готовый вариант прикрепила выше) и запустила снова.

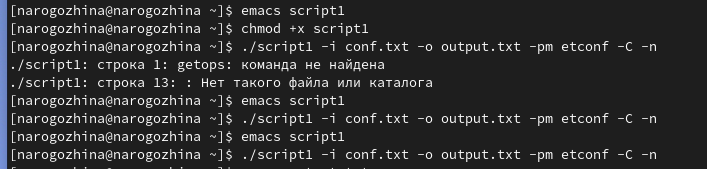


Figure 2: Запуск первого скрипта

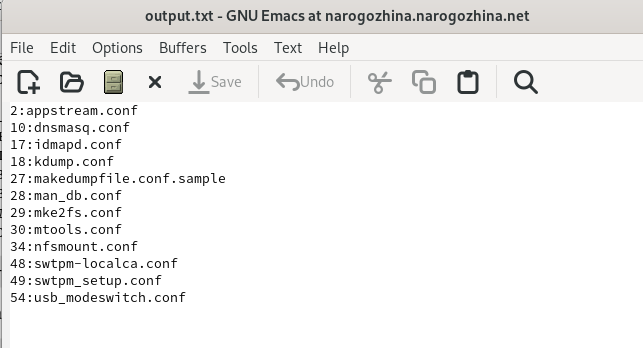


Figure 3: Результат

В данном случае, результат выполнения скрипта сохранился в текстовый файл output.txt

1. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции exit(n), передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды $?, выдать сообщение о том, какое число было введено (рис. [4](#fig:004)):

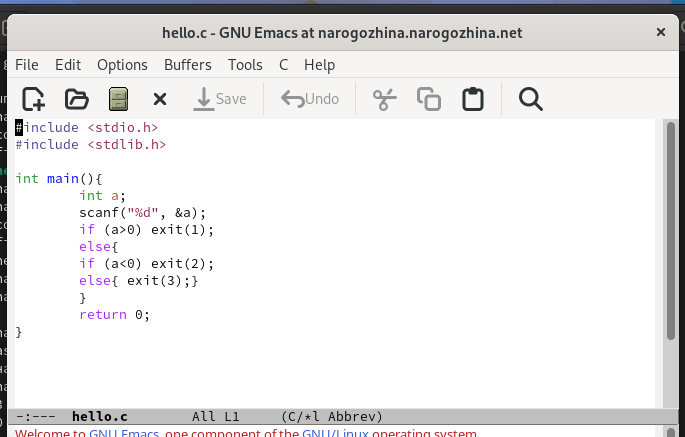


Figure 4: Программа на С

Чтобы программа работала нормально, пришлось доустановить gcc компилятор с++ (рис. [5](#fig:005)):

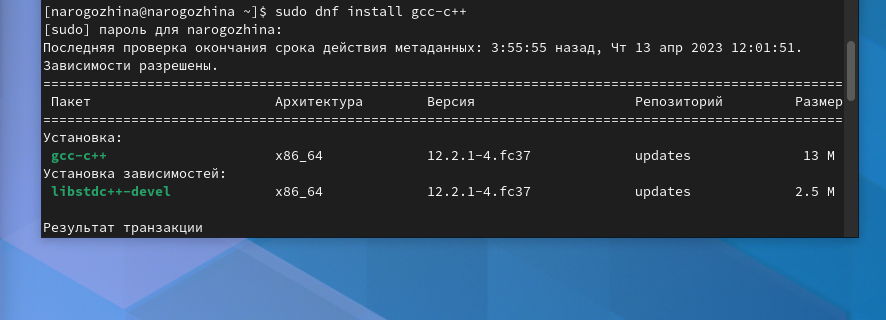


Figure 5: Доустановка

Далее скомпилируем файл (рис. [6](#fig:006)):

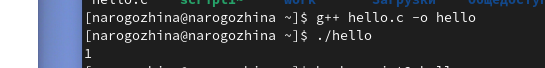


Figure 6: Компиляция

Сам скрипт (рис. [7](#fig:007)):

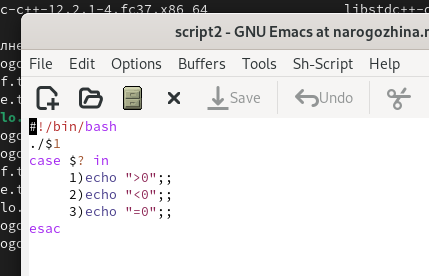


Figure 7: Скрипт №2

Пример выполнения (рис. [8](#fig:008)):

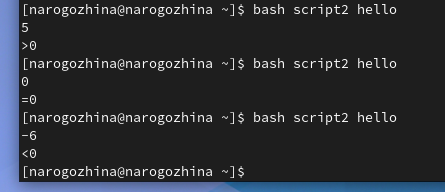


Figure 8: Выполнение скрипта №2

1. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до 𝑁 (например 1.tmp, 2.tmp, 3.tmp,4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют) (рис. [9](#fig:009)):

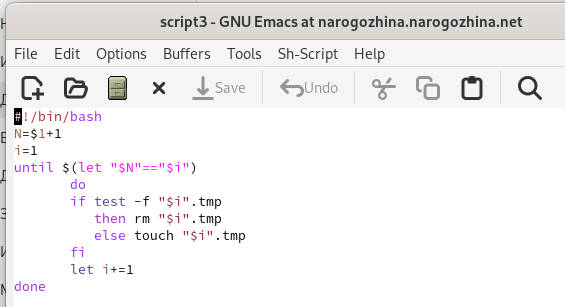


Figure 9: Скрипт №3

Пример выполнения (первый раз) (рис. [10](#fig:010)):

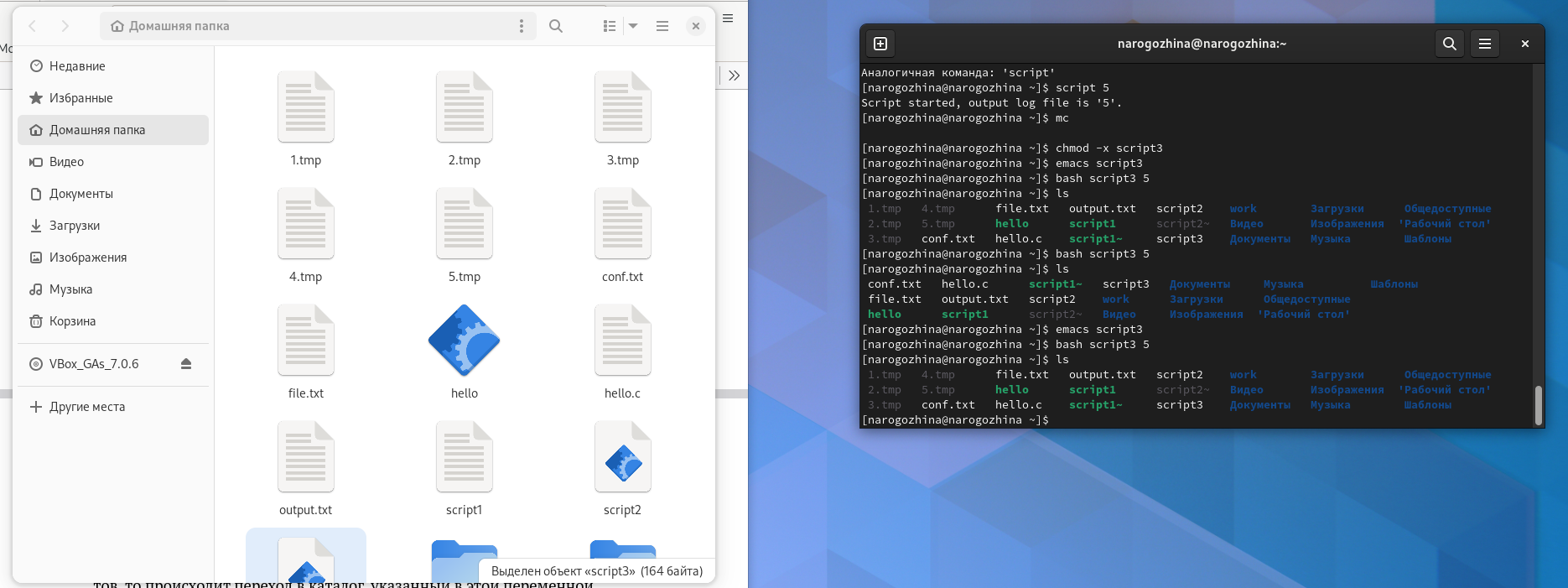


Figure 10: Выполнение скрипта №3

Выполнение повторно (рис. [11](#fig:011)):

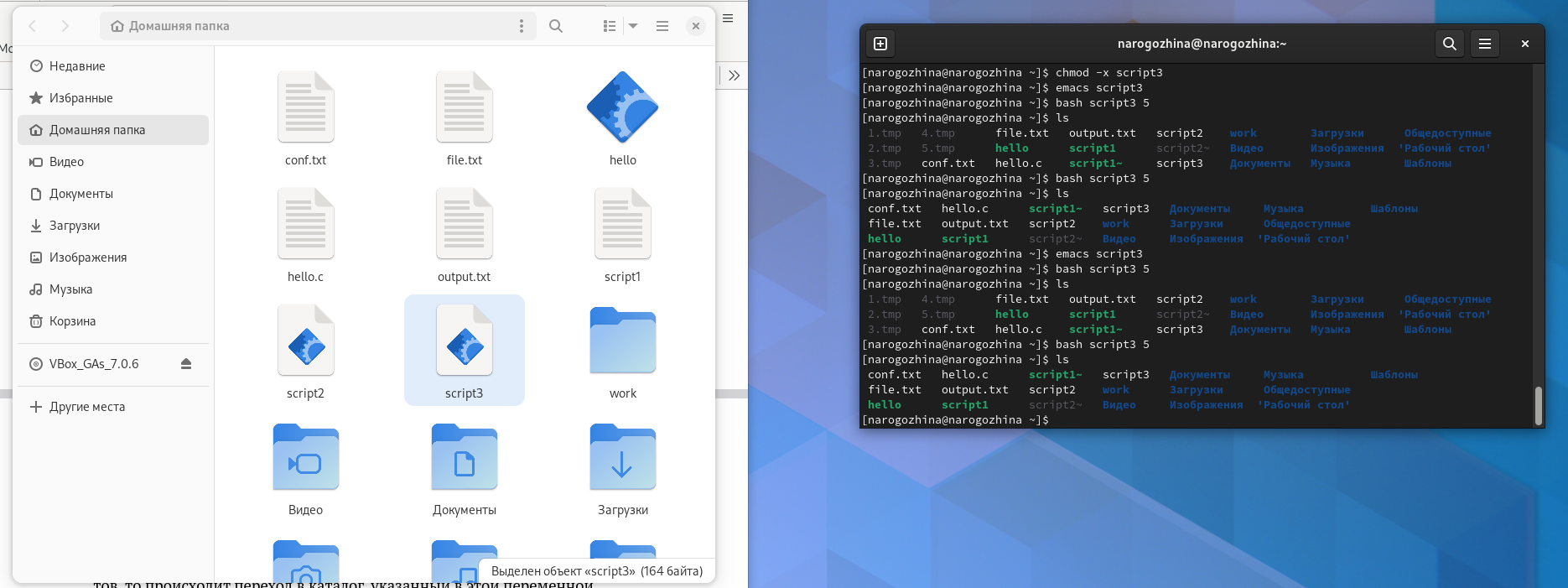


Figure 11: Повторное выполнение скрипта №3

1. Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find) (рис. [12](#fig:012)):

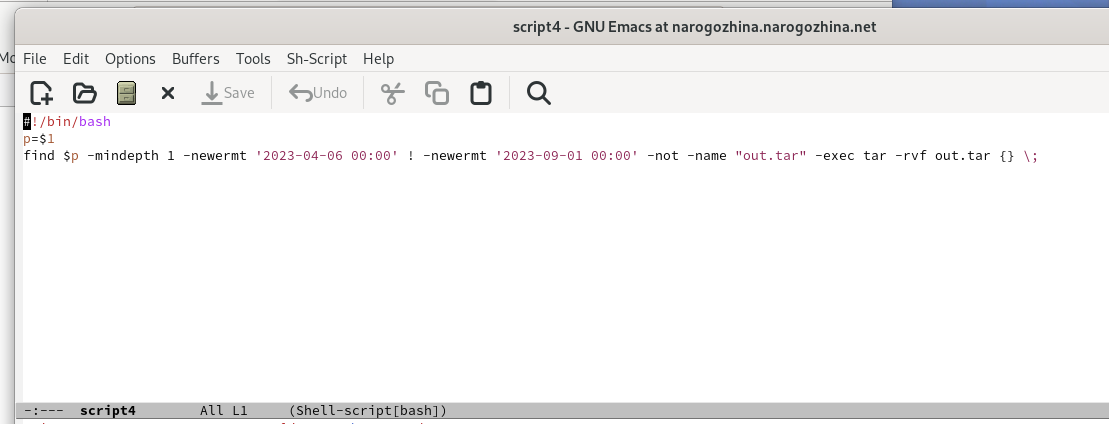


Figure 12: Скрипт №4

Пример выполнения (рис. [13](#fig:013)):

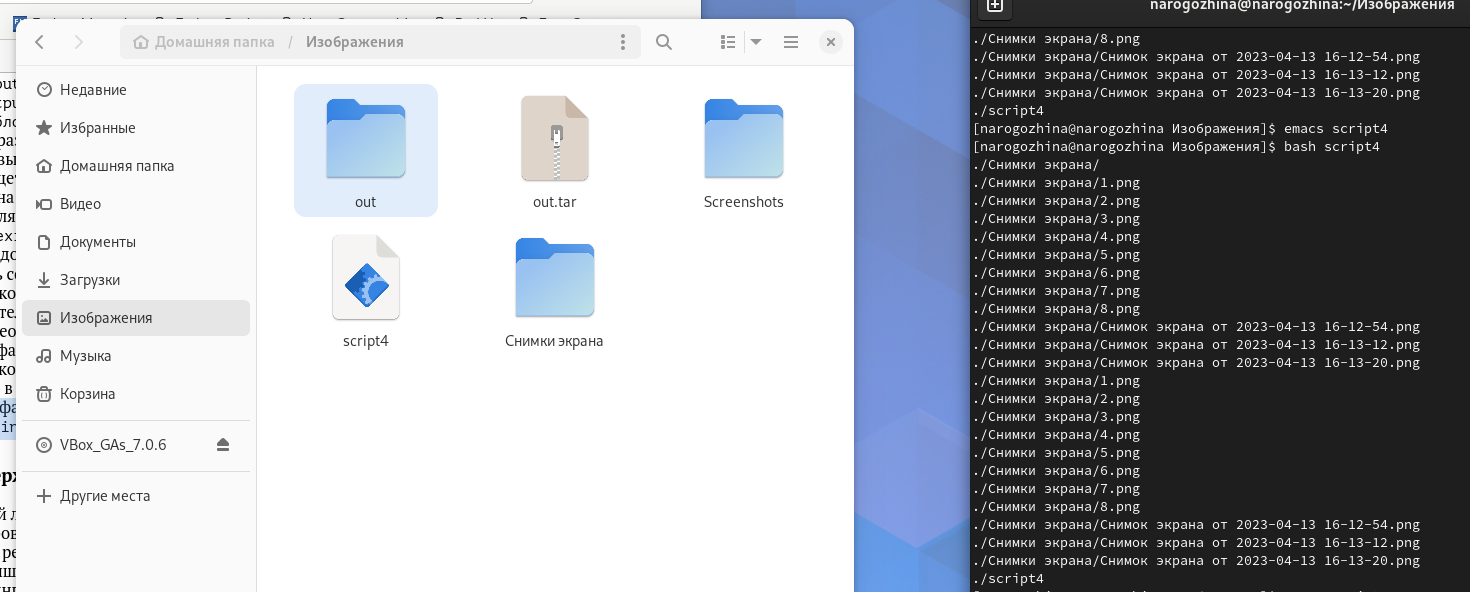


Figure 13: Запуск скрипта

Здесь представлен архив out, который был создан во время работы скрипта, и в нем (рис. [14](#fig:014)):

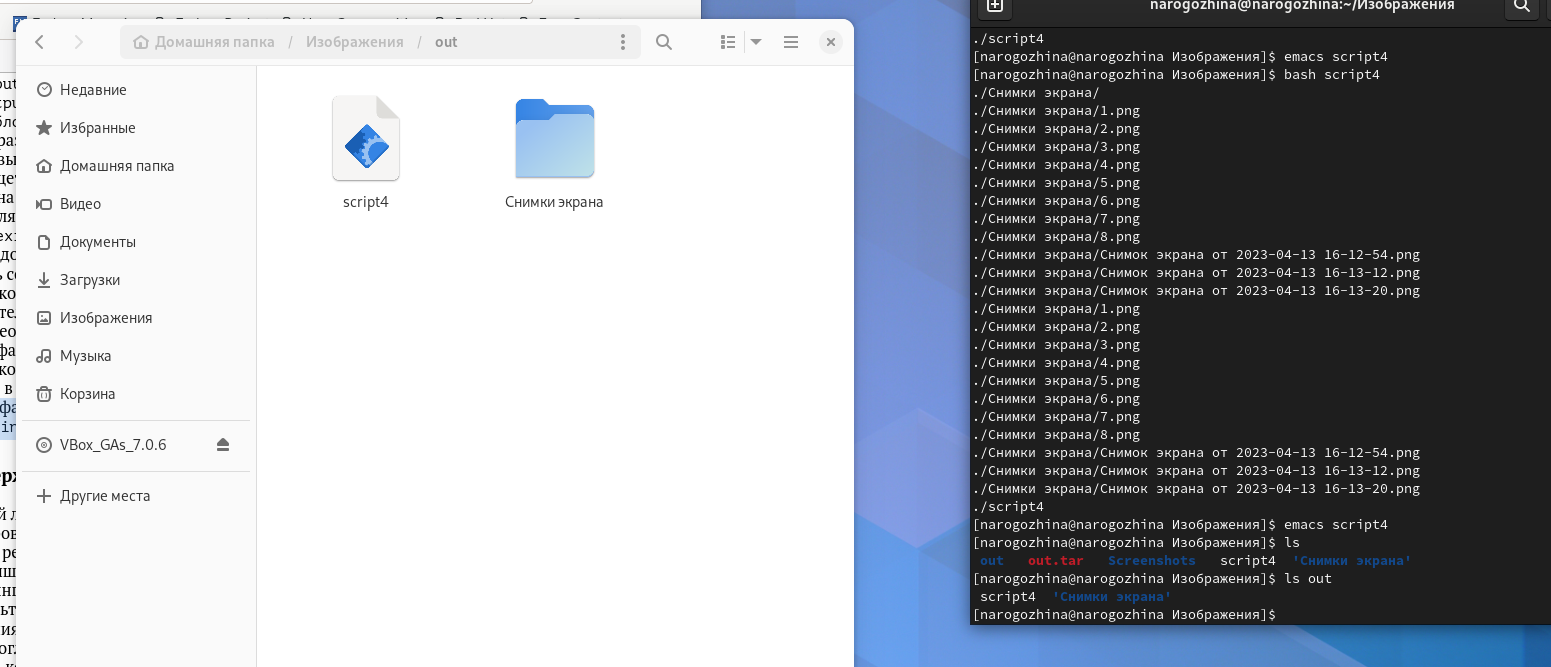


Figure 14: Папка out

Я запускала этот скрипт именно в папке “Изображения”, т.к. в домашнем каталоге слишком много недавно изменённых файлов.

# 5 Выводы

В ходе лабораторной работы мы научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# Список литературы

1. [Руководство по выполнению лабораторной работы №10](https://esystem.rudn.ru/pluginfile.php/1975779/mod_resource/content/4/010-lab_shell_prog_1.pdf)
2. [Руководство по выполнению лабораторной работы №11](https://esystem.rudn.ru/pluginfile.php/1975781/mod_resource/content/5/011-lab_shell_prog_2.pdf)