Лабораторная работа №10

Дисциплина - операционные системы

Волгин Иван Алексеевич

Содержание

# 1 Цель работы

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

# 2 Задание

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 ар- хиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
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 ко- манды 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 ко- мандной строки.

# 3 Теоретическое введение

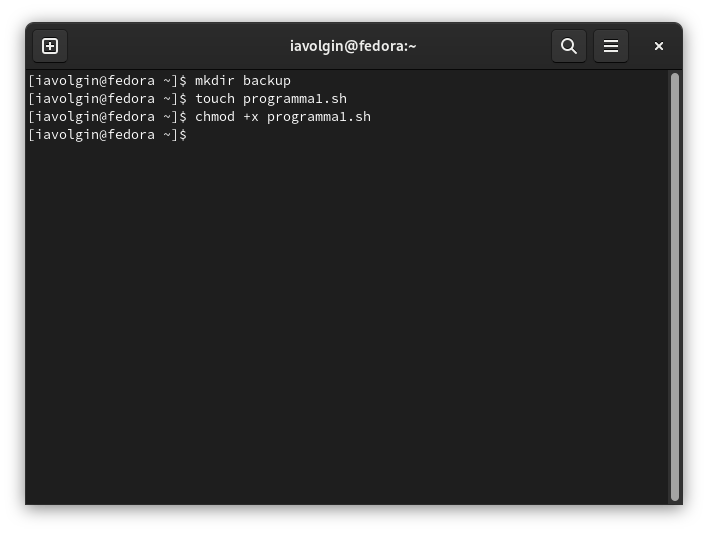
* Командный процессор (командная оболочка, интерпретатор команд shell) — это про- грамма, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления програм- мой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей сов- мещает свойства оболочек С и Корна (разработка компании Free Software Foundation). 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 опера- ционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболоч- ках большинство команд будет совпадать с описанными ниже.
* Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (term), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате типа radix#number, где radix (основание системы счисления) — любое чис- ло не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%). Команда let берет два операнда и присваивает их переменной. Положительным мо- ментом 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 let также расширяет другие выражения let, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда let не ограничена простыми арифметическими выражениями. Табл. 10.1 показывает полный набор let-операций. Подобно С оболочка bash может присваивать переменной любое значение, а произволь- ное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина». Для облегчения программирования можно записывать условия оболочки bash в двойные скобки — (( )).

Арифметические операторы оболочки bash Оператор Синтаксис Результат ! !ехр Если ехр равно 0, то возвращает 1; иначе 0 != ехр1 !=ехр2 Если ехр1 не равно ехр2, то возвращает 1; иначе 0 % ехр1%ехр2 Возвращает остаток от деления ехр1 на ехр2 %= var=%exp Присваивает остаток от деления var на ехр переменной var & ехр1&ехр2 Возвращает побитовое AND выражений ехр1 и ехр2 && ехр1&&ехр2 Если и ехр1 и ехр2 не равны нулю, то возвращает 1; иначе 0 &= var &= ехр Присваивает переменной var побитовое AND var и ехр \* ехр1 \* ехр2 Умножает ехр1 на ехр2 *= var* = ехр Умножает ехр на значение переменной var и присваивает результат переменной var + ехр1 + ехр2 Складывает ехр1 и ехр2 += var += ехр Складывает ехр со значением переменной var и результат присваивает переменной var - -exp Операция отрицания exp (унарный минус) - expl - exp2 Вычитает exp2 из exp1 -= var -= exp Вычитает exp из значения переменной var и присваивает результат переменной var / exp / exp2 Делит exp1 на exp2 /= var /= exp Делит значение переменной var на exp и присваивает ре- зультат переменной var < expl < exp2 Если exp1 меньше, чем exp2, то возвращает 1, иначе возвра- щает 0 << exp1 << exp2 Сдвигает exp1 влево на exp2 бит <<= var <<= exp Побитовый сдвиг влево значения переменной var на exp <= expl <= exp2 Если exp1 меньше или равно exp2, то возвращает 1; иначе возвращает 0 = var = exp Присваивает значение exp переменной var == exp1==exp2 Если exp1 равно exp2, то возвращает 1; иначе возвращает 0 > exp1 > exp2 1, если exp1 больше, чем exp2; иначе 0 >= exp1 >= exp2 1, если exp1 больше или равно exp2; иначе 0 >> exp >> exp2 Сдвигает exp1 вправо на exp2 бит >>= var >>=exp Побитовый сдвиг вправо значения переменной var на exp ^ exp1 ^ exp2 Исключающее OR выражений exp1 и exp2 ^= var ^= exp Присваивает переменной var побитовое XOR var и exp | exp1 | exp2 Побитовое OR выражений exp1 и exp2 |= var |= exp Присваивает переменной var результат операции XOR var и exp || exp1 || exp2 1, если или exp1 или exp2 являются ненулевыми значениями; иначе 0 ~ ~exp Побитовое дополнение до exp

* 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, начинаю- щемуся с любой строчной буквы латинского алфавита. Такие символы, как ’ < > \* ? |  ” &, являются метасимволами и имеют для ко- мандного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с по- мощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавыч- ки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , “. Например, – echo \* выведет на экран символ *, – echo ab’*|*’cd выведет на экран строку ab*|\*cd.

# 4 Выполнение лабораторной работы

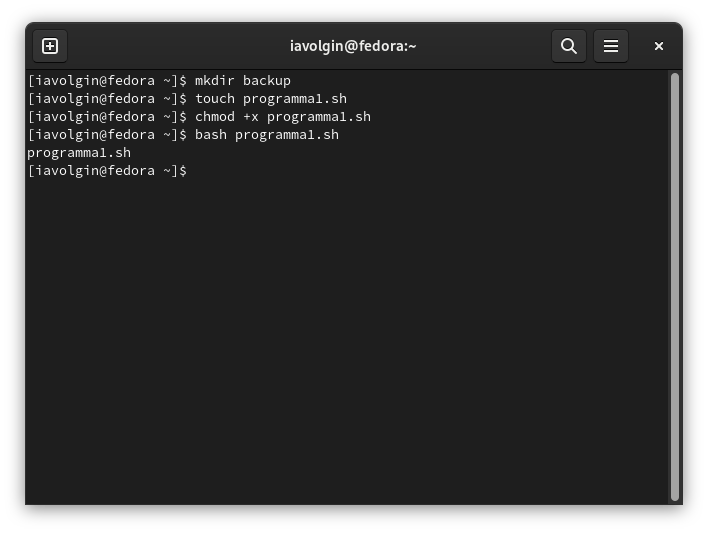
Я приступил к первому заданию и для начала создал файл и изменил для него права доступа (рис. ??). После этого написал код, который выполняет поставленную задачу (рис. ??). Далее запустил программу (рис. ??) и проверил правильность ее выполнения (рис. ??).



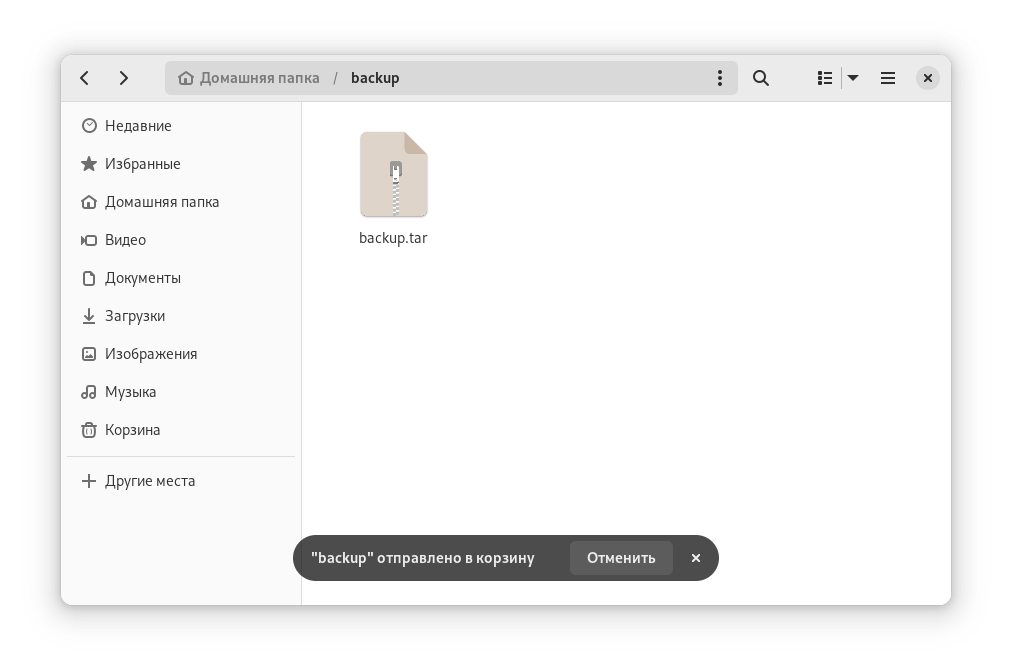
Создание файла



Код

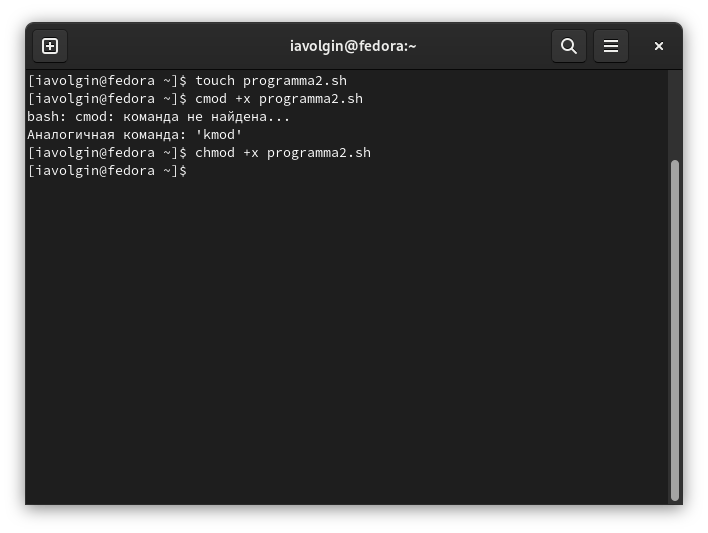


Исполнение программы



Проверка правильности выполнения

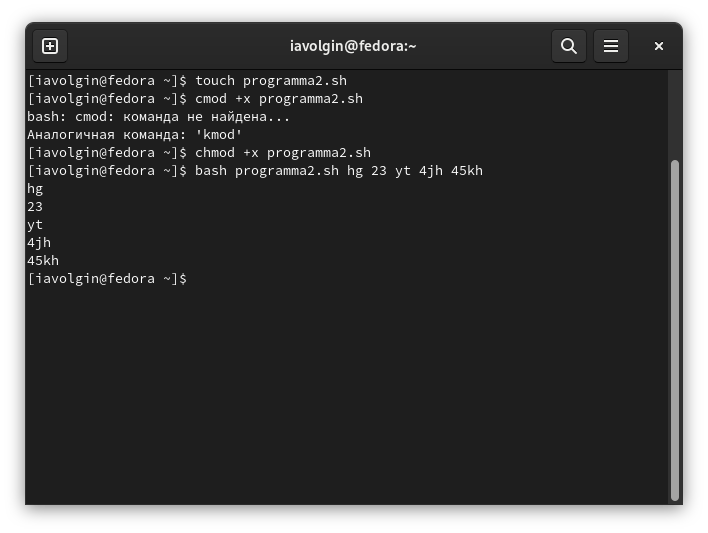
После этого я приступил к выполнению второго задания. Все идентично: создал файл и поменял права доступа (рис. ??), написал код (рис. ??), использую цикл for, и выполнил его (рис. ??).



Создание файла

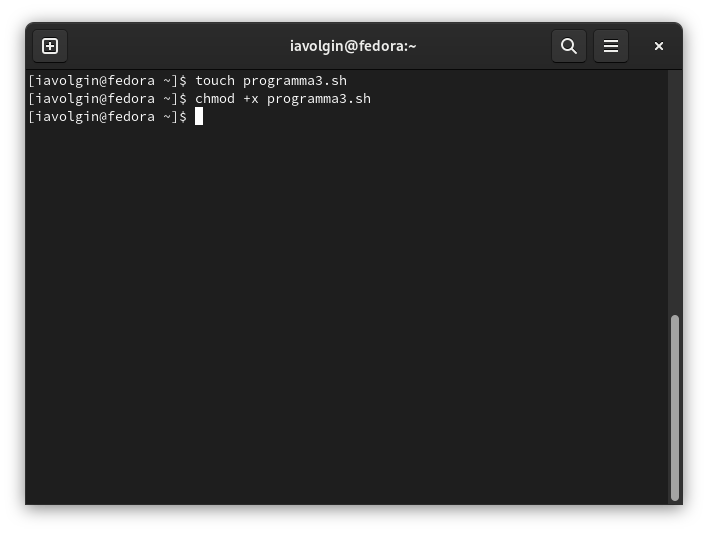


Код

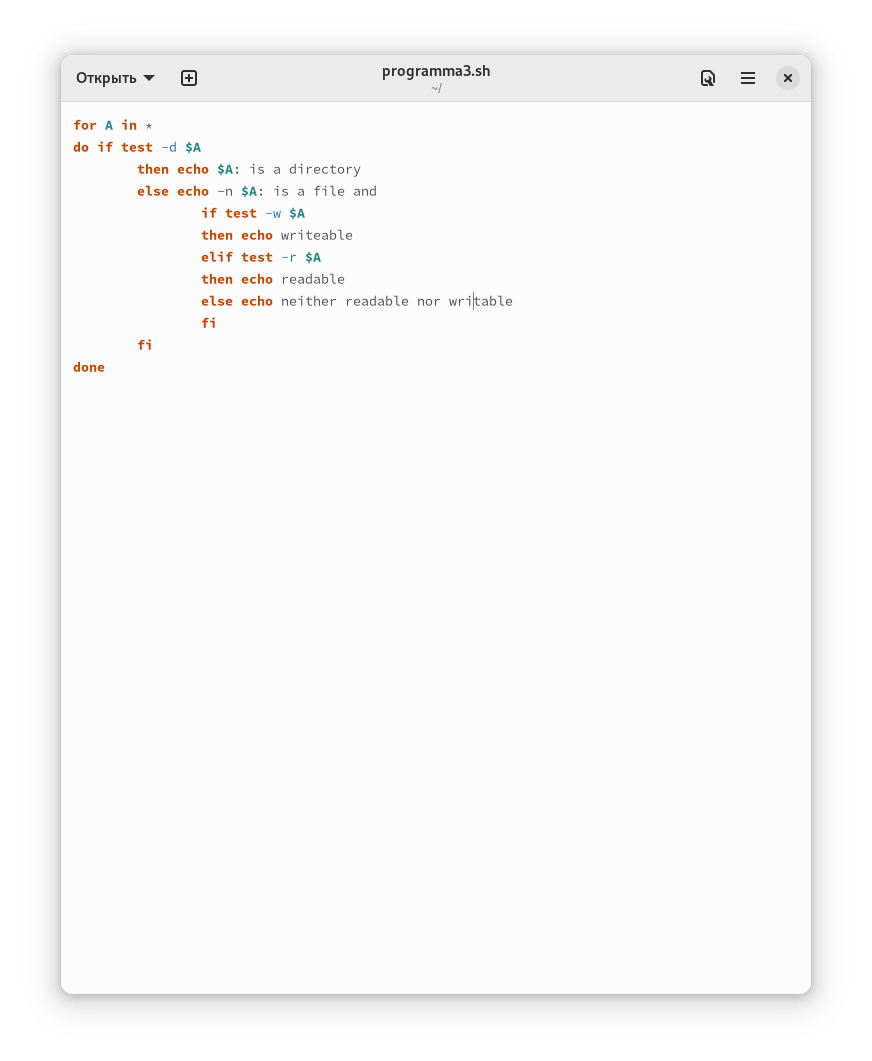


Выполнение программы

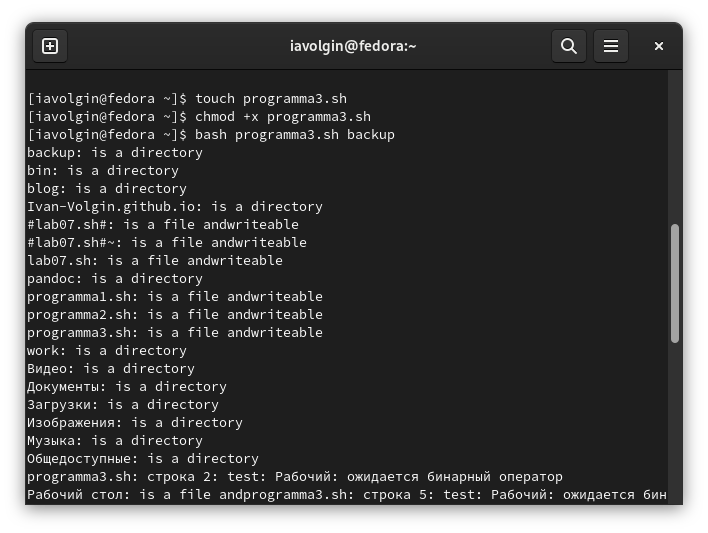
Далее задание номер 3. Все по тому же сценарию: создаю файл (рис. ??), пишу код (рис. ??). В коде так же используется цикл for, но уже с оператором if, с помощью которого мы проверяем, чем является наш элемент (директорией или файлом) и если файлом, то тоже через оператор if проверяем какие у него есть права доступа. Далее выполняю его (рис. ??)



Создание файла

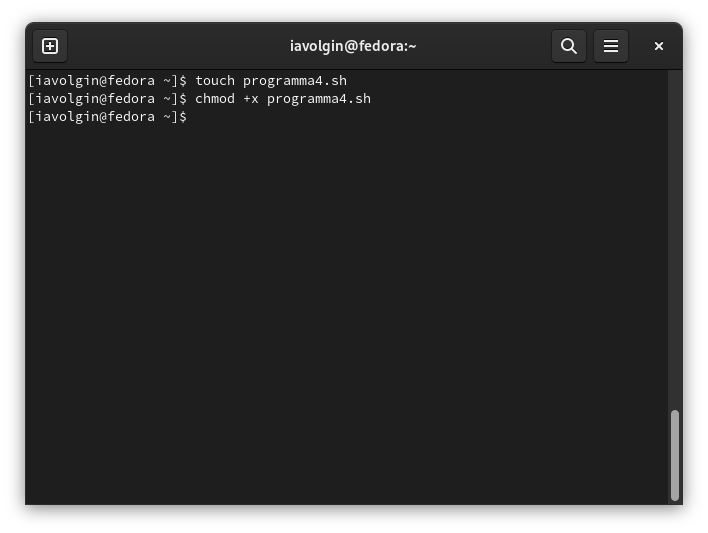


Код

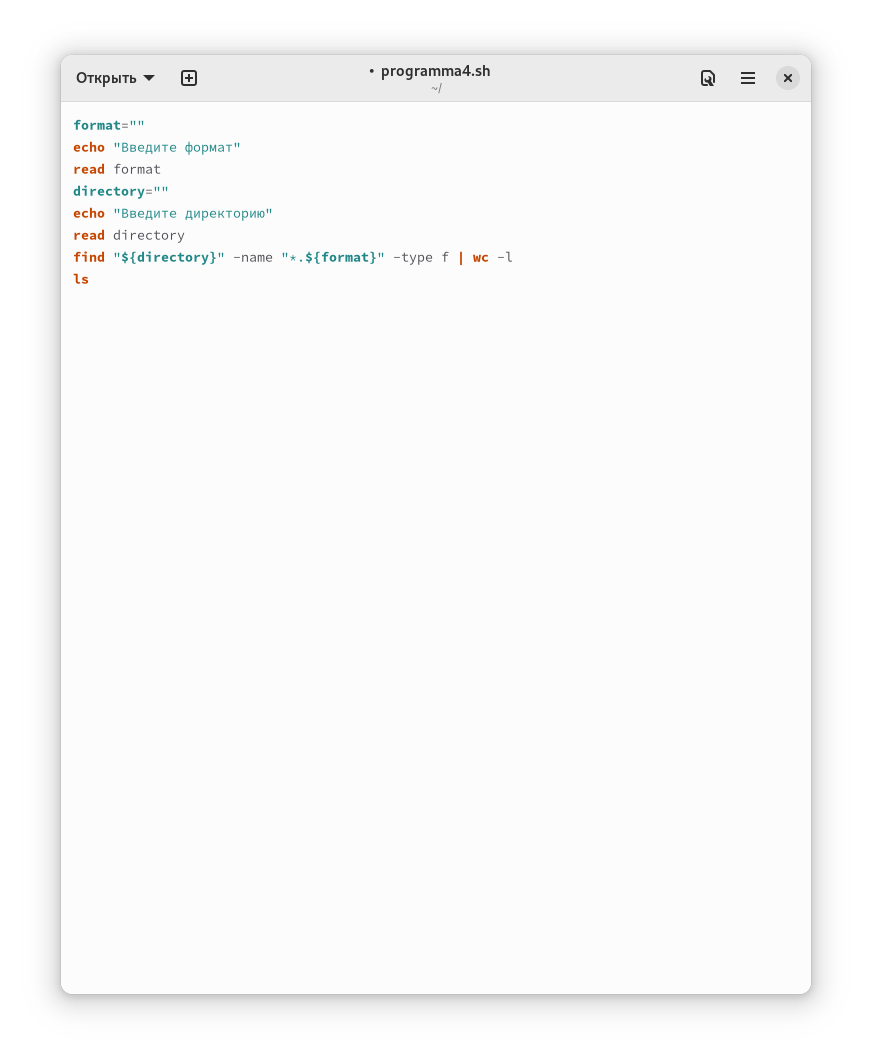


Выполнение программы

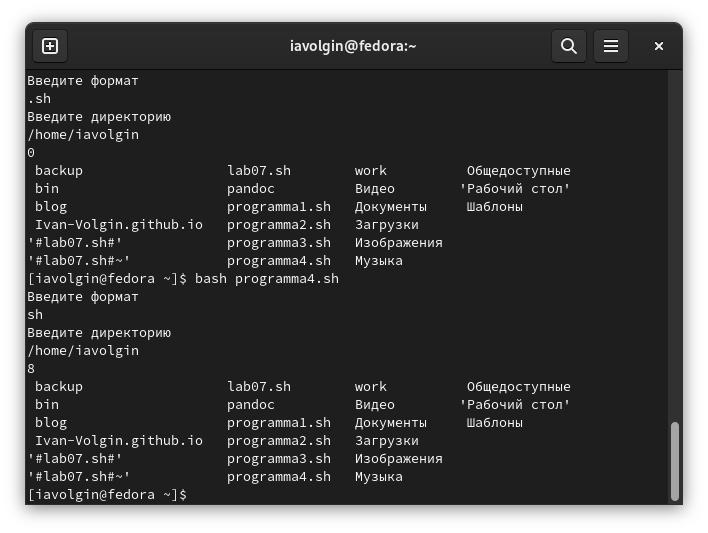
Четвертое задание выполнялось так же. Создал файл и добавил права доступа (рис. ??), написал код (рис. ??), и выполнил его (рис. ??).



Создание файла



Код



Выполнение программы

# 5 Выводы

В ходе выполнения лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.