Отчёт по лабораторной работе №10

Программирование в командном процессоре ОС UNIX. Командные файлы

Надежда Александровна Рогожина

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 архиваторов 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 команды 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 командной строки.

# 3 Теоретическое введение

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:

* 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 программой совместимы с операторами оболочки Борна;
* BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).

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 число 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 (%).

Например, в табл. [1](#tbl:std-dir) приведено краткое описание некоторых арифметических операторов оболочки bash.

Table 1: Описание некоторых арифметических операторов оболочки bash

| Оператор | Синтаксис | Результат |
| --- | --- | --- |
| ! | !ехр | Если ехр равно 0, то возвращает 1; иначе 0 |
| % | ехр1%ехр2 | Возвращает остаток от деления ехр1 на ехр2 |
| & | ехр1&ехр2 | Возвращает побитовое AND выражений ехр1 и ехр2 |
| \* | ехр1 \* ехр2 | Умножает ехр1 на ехр2 |
| | | exp1 | exp2 | Побитовое OR выражений exp1 и exp2 |
| ~ | ~exp | Побитовое дополнение до exp |
| ^ | exp1 ^ exp2 | Исключающее OR выражений exp1 и exp2 |
| >> | exp >> exp2 | Сдвигает exp1 вправо на exp2 бит |

# 4 Выполнение лабораторной работы

1. Для начала создадим файл script1 с помощью emacs (рис. [1](#fig:001)):

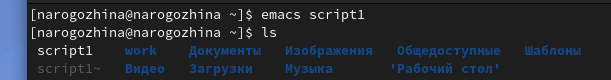


Figure 1: Создание файла

1. Создадим нужную нам директорию (рис. [2](#fig:002)):

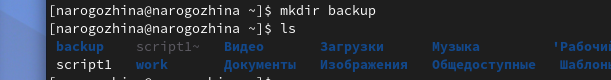


Figure 2: Директория backup

1. Скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в вашем домашнем каталоге (рис. [3](#fig:003)):

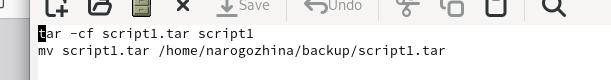


Figure 3: Скрипт №1

1. Чтобы каждый раз не прописывать *“bash script1”* сразу присвоим ему режим выполнения (+х) и также выполним скрипт и проверим (рис. [4](#fig:004)):

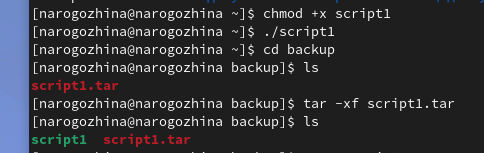


Figure 4: Команды

1. Командный файл, обрабатывающий любое произвольное число аргументов командной строки, в том числе превышающее десять, последовательно распечатывающий значения всех переданных аргументов (рис. [5](#fig:005)):

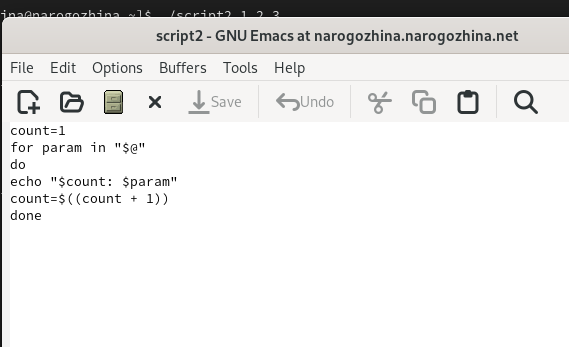


Figure 5: Скрипт №2

Также присвоим режим выполнения и проверим выполнение самого скрипта (рис. [6](#fig:006)):

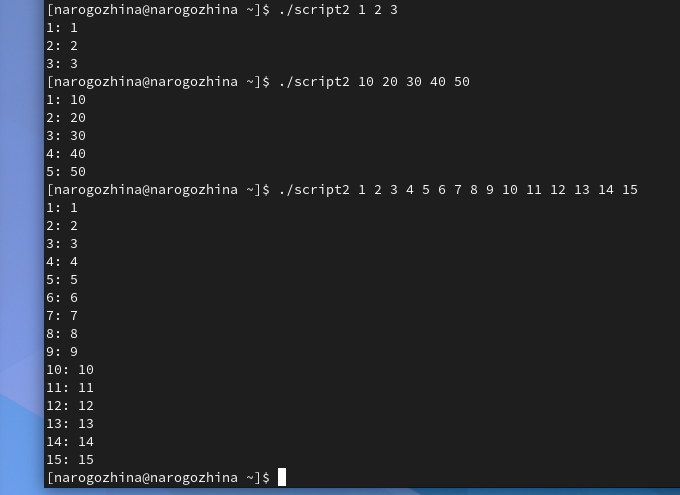


Figure 6: Скрипт №2

1. 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 (рис. [7](#fig:007)):

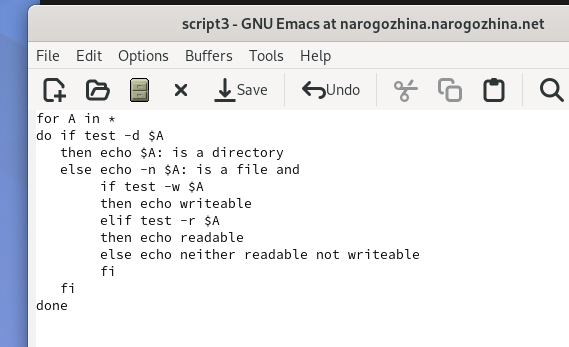


Figure 7: Скрипт №3

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

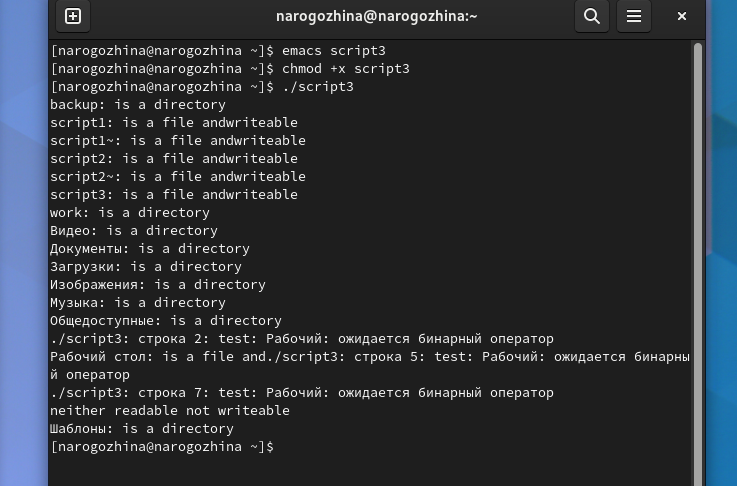


Figure 8: Скрипт №3

1. 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 (рис. [9](#fig:009)):

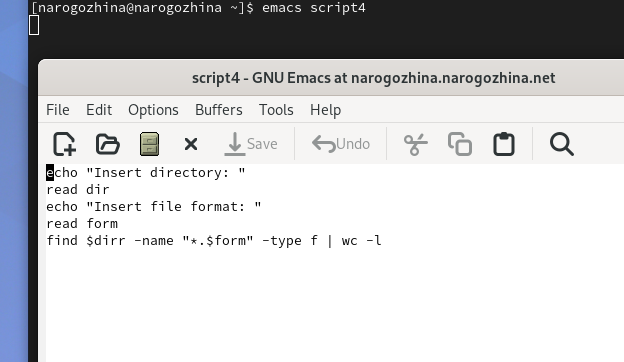


Figure 9: Скрипт №4

Пример вывода (рис. [10](#fig:010)):

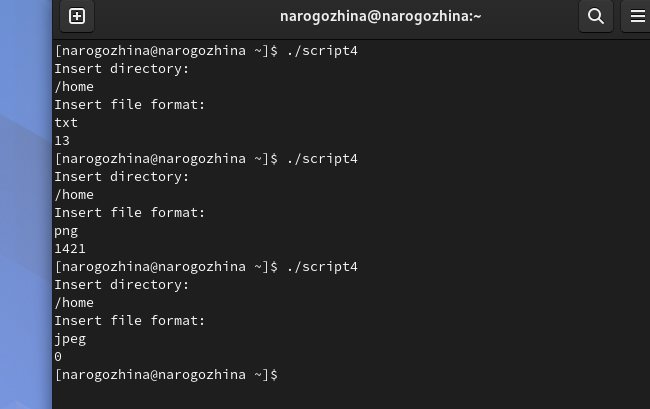


Figure 10: Скрипт №4

# 5 Выводы

В ходе лабораторной работы мы познакомились с командными файлами Linux, а также научились писать небольшие командные файлы сами.

# 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. [Справочник по архивации в linux](https://losst.pro/arhivatsiya-v-linux)