

# **Отчёт по лабораторной работе**

**Лабораторная работа № 10**

Мурзаев Замир Зейнадинович

# Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	8
5	Ответы на вопросы	9
	Список литературы	11

# Список иллюстраций

3.1	Код программы	. . . . .	7
3.2	Код программы	. . . . .	7
3.3	Код программы	. . . . .	7
3.4	Код программы	. . . . .	7

## Список таблиц

# 1 Цель работы

Цель - 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 Выполнение лабораторной работы

1)Пишем скрипт, который при запуске будет делать резервную копию самого себя. (рис. 3.1).

Код программы

Рис. 3.1: Код программы

2)Пишем командный файл, обрабатывающий любое произвольное число аргументов командной строки, в том числе превышающее десять. (рис. 3.2).

Код программы

Рис. 3.2: Код программы

3)Пишем командный файл - аналог команды ls. (рис. 3.3).

Код программы

Рис. 3.3: Код программы

4)Пишем командный файл, который находит файлы определенного формата (рис. 3.4).

Код программы

Рис. 3.4: Код программы

## 4 Выводы

Освоены основы программирования в оболочке ОС UNIX?LINUX. Научились писать небольшие командные файлы.

## 5 Ответы на вопросы

- 1) Командная оболочка позволяет исполнять команды.
- 2) Это набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
- 3) через равно
- 4) let позволяет выполнять арифметические операции при задании переменных, read считывает стандартный поток вывода.
- 5) стандартные
- 6) вычисляет логические выражения
- 7) PATH, ENV, TERM
- 8) специальные символы
- 9) как угодно, но можно через .
- 10) Для создания файла применять команды touch, chmod. Для запуска ввести ./
- 11) При помощи ключевого слова function
- 12) ls -l выведет дополнительную информацию
- 13) Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Удалить функцию можно с помощью команды unset с флагом -f. Команда typeset имеет четыре опции для работы с функциями: – -f — перечисляет определённые на текущий момент функции; – -ft — при последующем вызове функции инициализирует её трассировку; – -fx — экспортирует все перечисленные функции в любые дочерние программы оболочек; – -fu — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранят-

ся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции.

14) Через пробел при запуске программы.

15) – `$*` — отображается вся командная строка или параметры оболочки; – `$?` — код завершения последней выполненной команды; – `$$` — уникальный идентификатор процесса, в рамках которого выполняется командный процессор; – `$!` — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; – `$-` — значение флагов командного процессора; – `${#}` — возвращает целое число — количество слов, которые были результатом `$`; – `${#name}` — возвращает целое значение длины строки в переменной `name`; – `${name[n]}` — обращение к `n`-му элементу массива; – `${name[]}` — перечисляет все элементы массива, разделённые пробелом; – `${name[@]}` — то же самое, но позволяет учитывать символы пробелы в самих переменных; – `${name:-value}` — если значение переменной `name` не определено, то оно будет заменено на указанное `value`; – `${name:value}` — проверяется факт существования переменной; – `${name=value}` — если `name` не определено, то ему присваивается значение `value`; – `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке; – `${name+value}` — это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`; – `${name#pattern}` — представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`); – `${#name[]}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.

## **Список литературы**