

# **Отчет по лабораторной работе №10**

**Дисциплина Операционные системы**

Колобова Елизавета, гр. НММбд-01-22

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	9
4	Выводы	14
	Список литературы	15

## Список иллюстраций

2.1	Рис. 1. Скрипт резервного копирования . . . . .	6
2.2	Рис. 2. Скрипт с неограниченным числом аргументов . . . . .	7
2.3	Рис. 3. Скрипт ls . . . . .	7
2.4	Рис. 4. Скрипт подсчета количества файлов . . . . .	8
2.5	Рис. 5. Выполнение скриптов . . . . .	8

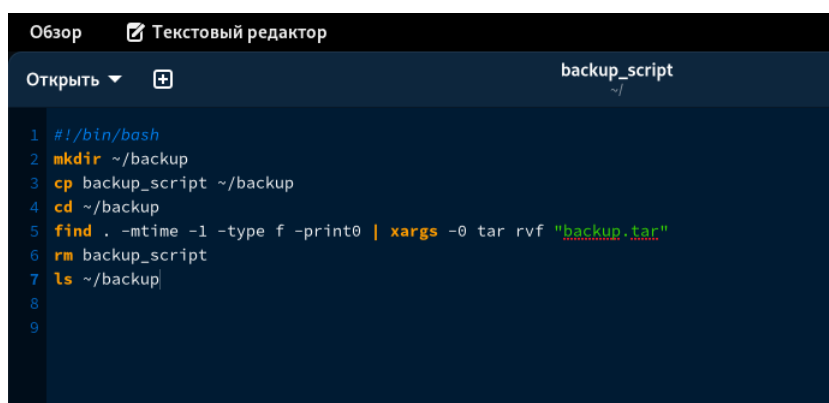
## **Список таблиц**

# 1 Цель работы

Цель работы - изучить основы программирования в оболочке ОС UNIX/Linux.  
Научиться писать небольшие командные файлы.

## 2 Выполнение лабораторной работы

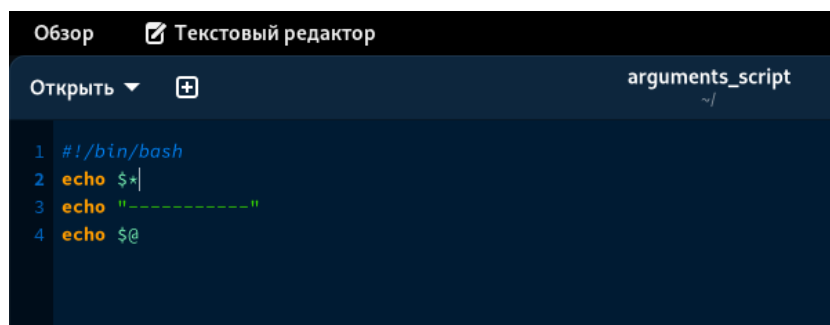
1. Напишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. (рис. [2.1], [2.5])

A screenshot of a terminal window with a dark background. The title bar at the top says "Обзор" and "Текстовый редактор". Below the title bar, there's a tab labeled "backup\_script" with a "~/" icon. The terminal content shows a script with line numbers 1 through 9. The script is as follows:

```
1 #!/bin/bash
2 mkdir ~/backup
3 cp backup_script ~/backup
4 cd ~/backup
5 find . -mtime -1 -type f -print0 | xargs -0 tar rvf "backup.tar"
6 rm backup_script
7 ls ~/backup
8
9
```

Рис. 2.1: Рис. 1. Скрипт резервного копирования

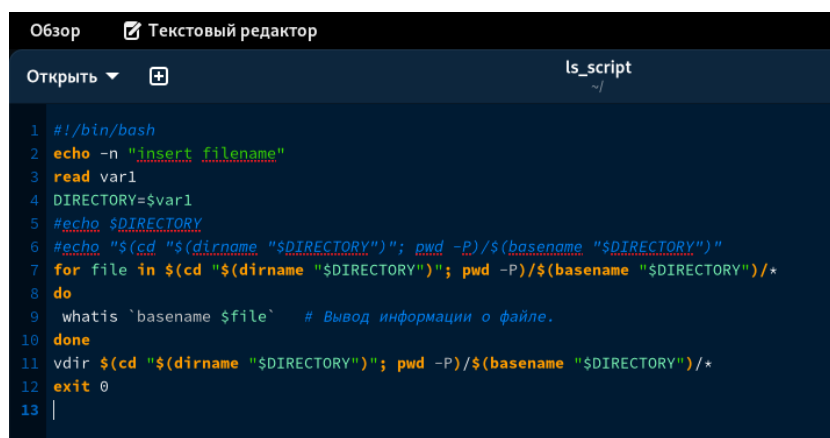
2. Напишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Скрипт может последовательно распечатывать значения всех переданных аргументов. (рис. [2.2], [2.5])



```
Обзор  Текстовый редактор
Открыть  + arguments_script
~/
1 #!/bin/bash
2 echo $*
3 echo "-----"
4 echo $@
```

Рис. 2.2: Рис. 2. Скрипт с неограниченным числом аргументов

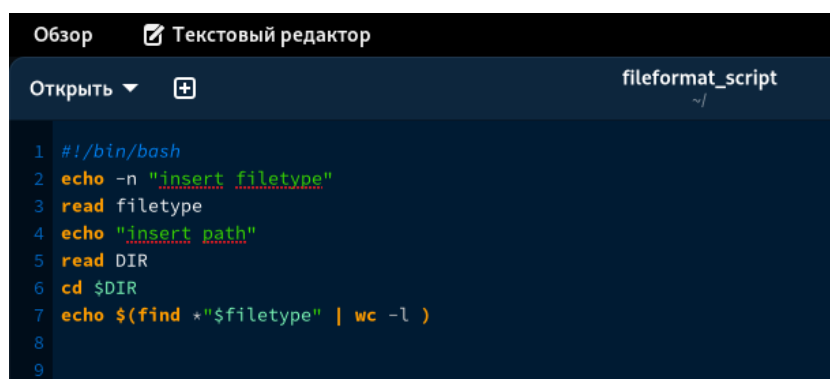
3. Напишем командный файл — аналог команды ls (без использования самой этой ко- манды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога. (рис. [2.3], [2.5])



```
Обзор  Текстовый редактор
Открыть  + ls_script
~/
1 #!/bin/bash
2 echo -n "insert filename"
3 read var1
4 DIRECTORY=$var1
5 #echo $DIRECTORY
6 #echo "$(cd "$(dirname "$DIRECTORY")"; pwd -P)/$(basename "$DIRECTORY")"
7 for file in $(cd "$(dirname "$DIRECTORY")"; pwd -P)/$(basename "$DIRECTORY")/*
8 do
9   whatis `basename $file` # Вывод информации о файле.
10 done
11 vdir $(cd "$(dirname "$DIRECTORY")"; pwd -P)/$(basename "$DIRECTORY")/*
12 exit 0
13 |
```

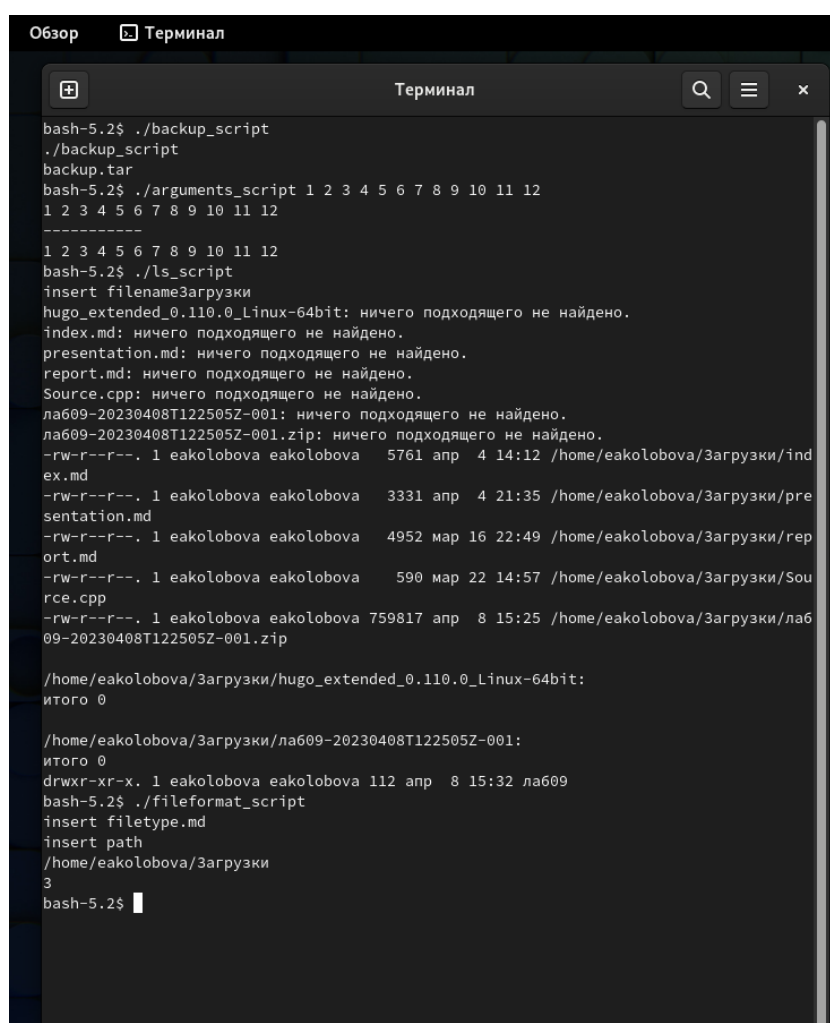
Рис. 2.3: Рис. 3. Скрипт ls

4. Напишем командный файл, который получает в качестве аргумента команд-ной 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.. (рис. [2.4], [2.5])



```
1 #!/bin/bash
2 echo -n "insert filetype"
3 read filetype
4 echo "insert path"
5 read DIR
6 cd $DIR
7 echo $(find *"$filetype" | wc -l )
8
9
```

Рис. 2.4: Рис. 4. Скрипт подсчета количества файлов



```
bash-5.2$ ./backup_script
./backup_script
backup.tar
bash-5.2$ ./arguments_script 1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
-----
1 2 3 4 5 6 7 8 9 10 11 12
bash-5.2$ ./ls_script
insert filenameЗагрузки
hugo_extended_0.110.0_Linux-64bit: ничего подходящего не найдено.
index.md: ничего подходящего не найдено.
presentation.md: ничего подходящего не найдено.
report.md: ничего подходящего не найдено.
Source.cpp: ничего подходящего не найдено.
ла609-20230408T122505Z-001: ничего подходящего не найдено.
ла609-20230408T122505Z-001.zip: ничего подходящего не найдено.
-rw-r--r--. 1 eakolobova eakolobova 5761 апр 4 14:12 /home/eakolobova/Загрузки/index.md
-rw-r--r--. 1 eakolobova eakolobova 3331 апр 4 21:35 /home/eakolobova/Загрузки/presentation.md
-rw-r--r--. 1 eakolobova eakolobova 4952 мар 16 22:49 /home/eakolobova/Загрузки/report.md
-rw-r--r--. 1 eakolobova eakolobova 590 мар 22 14:57 /home/eakolobova/Загрузки/Source.cpp
-rw-r--r--. 1 eakolobova eakolobova 759817 апр 8 15:25 /home/eakolobova/Загрузки/ла609-20230408T122505Z-001.zip

/home/eakolobova/Загрузки/hugo_extended_0.110.0_Linux-64bit:
итого 0

/home/eakolobova/Загрузки/ла609-20230408T122505Z-001:
итого 0
drwxr-xr-x. 1 eakolobova eakolobova 112 апр 8 15:32 ла609
bash-5.2$ ./fileformat_script
insert filetype.md
insert path
/home/eakolobova/Загрузки
3
bash-5.2$
```

Рис. 2.5: Рис. 5. Выполнение скриптов



### 3 Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются? 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; – C-оболочка (или csh) — настройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation)
2. Что такое POSIX? OSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
3. Как определяются переменные и массивы в языке программирования bash? Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Значение, присвоенное некоторой

переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол `$`. Оболочка `bash` позволяет работать с массивами. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами.

4. Каково назначение операторов `let` и `read`? Команда `let` берет два операнда и присваивает их переменной. Положительным моментом 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 `read` позволяет читать значения переменных со стандартного ввода.
5. Какие арифметические операции можно применять в языке программирования `bash`? 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 (%).
6. Что означает операция `(( ))`? Для облегчения программирования можно записывать условия оболочки `bash` в двойные скобки — `(( ))`.
7. Какие стандартные имена переменных Вам известны? Переменные `PS1` и `PS2` предназначены для отображения промптера командного процессора. `PS1` — это промптер командного процессора, по умолчанию его значение равно символу `$` или `#`. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер `PS2`.

Он по умолчанию имеет значение символа `>`. Другие стандартные переменные: `HOME` — имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной. `IFS` — последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (`new line`). `MAIL` — командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение `You have mail` (у Вас есть почта). `TERM` — тип используемого терминала. `LOGNAME` — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

8. Что такое метасимволы? Такие символы, как `' < > * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл.
9. Как экранировать метасимволы? Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$, ', , "`
10. Как создавать и запускать командные файлы? Создать обычный пустой файл, записать в него текст программы. Можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как-будто он является выполняемой программой.
11. Как определяются функции в языке программирования `bash`? Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды

unset с флагом -f.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом? С помощью проверки информации о файле с ключом -d и циклом, устанавливающим булев флаг
13. Каково назначение команд set, typeset и unset? 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 хранятся в командных файлах, а при их вызове оболочка просматривает переменную FPATH, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции. Значение всех переменных можно просмотреть с помощью команды set. Изъять переменную из программы можно с помощью команды unset.
14. Как передаются параметры в командные файлы? При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными.
15. Назовите специальные переменные языка bash и их назначение. – \$\* — 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, раз-

делённые 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`.

## 4 Выводы

Результатом проделанной работы является изучение основ программирования в оболочке ОС UNIX/Linux.

## **Список литературы**