

Отчёт по лабораторной работе №9

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Тагиев Павел Фаикович

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Задание 1	8
4.2	Задание 2	10
4.3	Задание 3	11
4.4	Задание 4	12
5	Ответы на контрольные вопросы	14
6	Выводы	19
	Список литературы	20

Список иллюстраций

4.1	Создание резервной копии	8
4.2	Аргументы командной строки	10
4.3	Содержимое корневой директории	11
4.4	Количество файлов с заданным расширением	12

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы [1].

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;
- *C-оболочка* (или *csh*) — надстройка на оболочкой 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*).

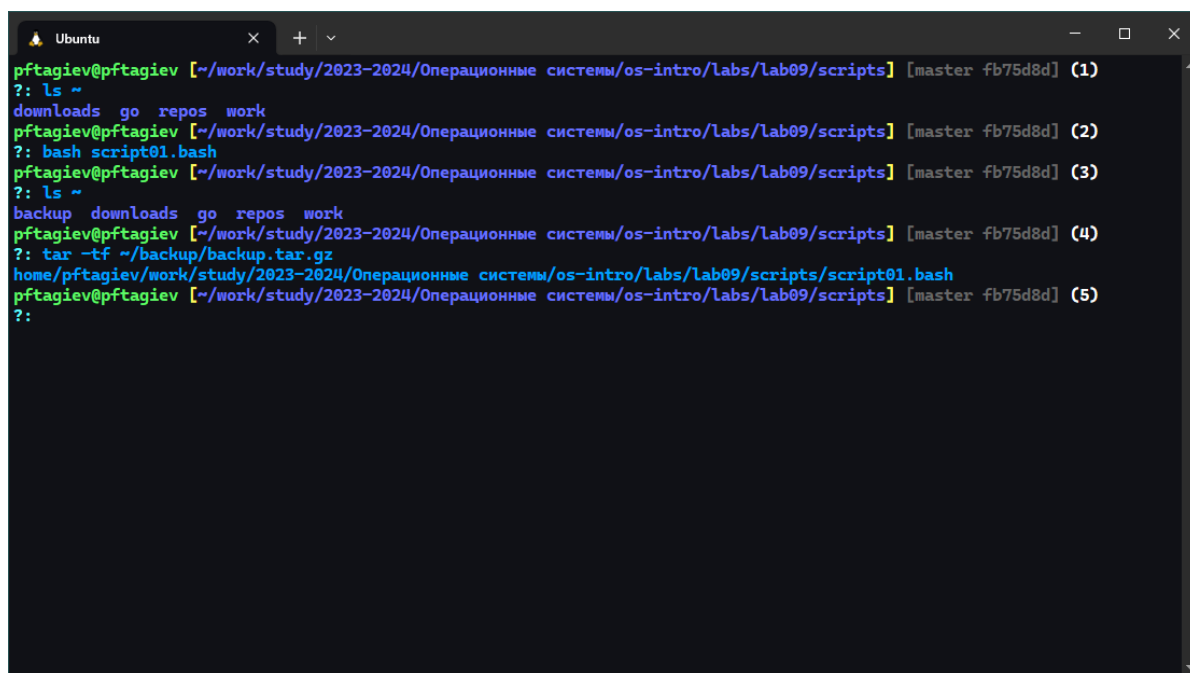
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты *POSIX* разработаны комитетом *IEEE (Institute of Electrical and Electronics Engineers)* для обеспечения совместимости различных *UNIX/Linux-подобных* операционных систем и переносимости прикладных программ на уровне исходного кода. *POSIX-совместимые* оболочки разработаны

на базе оболочки *Korna*. Рассмотрим основные элементы программирования в оболочке *bash*. В других оболочках большинство команд будет совпадать с описанными ниже [1].

4 Выполнение лабораторной работы

Чтобы не выдавать права на исполнение каждому написанному скрипту, я буду исполнять их передавая напрямую интерпретатору *bash*: `bash имя_скрипта`. Если все же нужно выдать права на исполнение это можно сделать командой `chmod +x имя_скрипта`.

4.1 Задание 1



```
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (1)
?: ls ~
downloads go repos work
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (2)
?: bash script01.bash
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (3)
?: ls ~
backup downloads go repos work
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (4)
?: tar -tf ~/backup/backup.tar.gz
home/pftagiev/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts/script01.bash
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (5)
?:
```

Рис. 4.1: Создание резервной копии

На лист. 4.1 можно увидеть код скрипта, который делает резервную копию самого себя. Он создает архив утилитой `tar`, в директории `~/backup/`.

На рис. 4.1 можно увидеть результат работы этого скрипта. В промте (2) вызывается сам скрипт, а в промте (4) выводится содержимое созданного архива.

Листинг 4.1 Скрипт создающий резервную копию себя

```
#!/usr/bin/bash

backup=~/backup

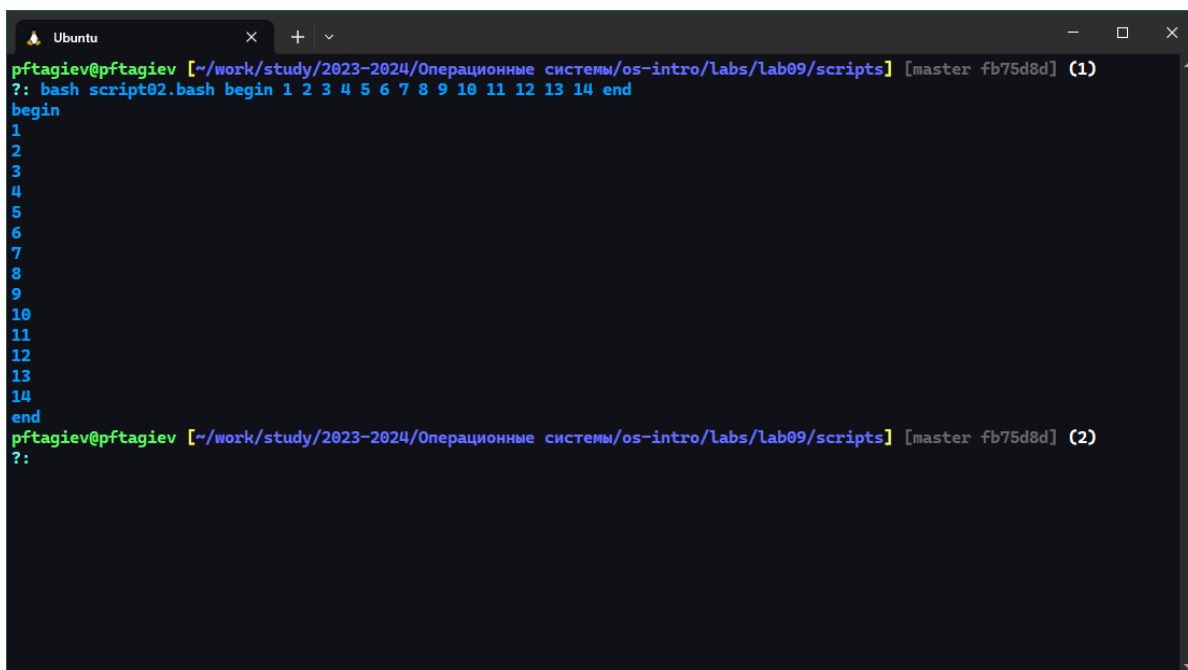
# создаю директорию для архива
mkdir -p $backup

# realpath нужен чтобы скрипт архивировался
# не зависимо от того из какого каталога его вызывают
# путь возвращаемый realpath абсолютный
script=$(realpath $0)

# удаление лидирующего слеша и
# обрамление строки кавычками,
# чтобы можно было работать с путями
# в которых встречаются пробелы и
# русские буквы
script="\${script#/}\'"

# eval - исполнить строку
eval tar -zcf $backup/backup.tar.gz -C / $script
```

4.2 Задание 2

A screenshot of a terminal window titled 'Ubuntu'. The prompt is 'pftagiev@pftagiev' and the current directory is '~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts'. The user has run the command 'bash script02.bash begin 1 2 3 4 5 6 7 8 9 10 11 12 13 14 end'. The script has executed, outputting the word 'begin' followed by the numbers 1 through 14, and finally the word 'end'. The prompt is now 'pftagiev@pftagiev' and the directory remains the same.

```
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (1)
?: bash script02.bash begin 1 2 3 4 5 6 7 8 9 10 11 12 13 14 end
begin
1
2
3
4
5
6
7
8
9
10
11
12
13
14
end
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (2)
?:
```

Рис. 4.2: Аргументы командной строки

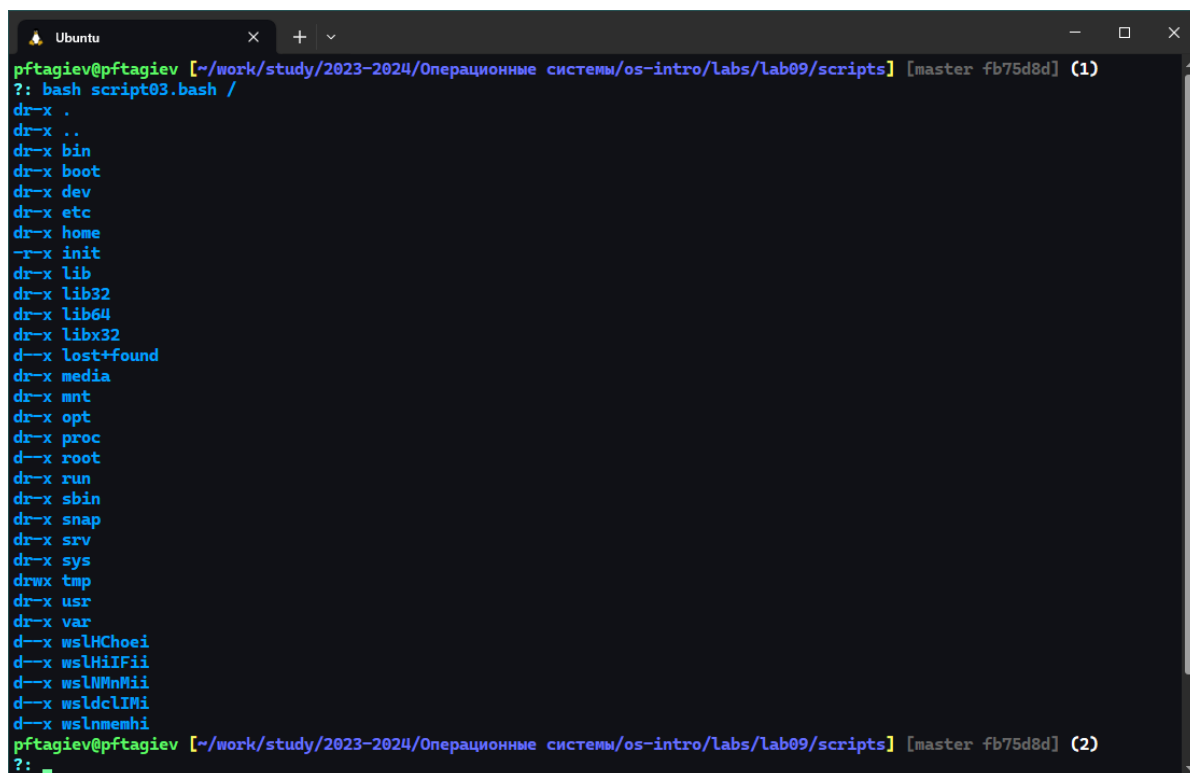
В этом задании нужно написать скрипт который выводит аргументы командной строки на экран. Сам скрипт можно увидеть на лист. 4.2, результат на рис. 4.2

Листинг 4.2 Вывод аргументов командной строки на экран

```
#!/usr/bin/bash

# построчный вывод аргументов командной строки
for i
do echo $i
done
```

4.3 Задание 3



```
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (1)
?: bash script03.bash /
dr-x .
dr-x ..
dr-x bin
dr-x boot
dr-x dev
dr-x etc
dr-x home
-r-x init
dr-x lib
dr-x lib32
dr-x lib64
dr-x libx32
d-x lost+found
dr-x media
dr-x mnt
dr-x opt
dr-x proc
d-x root
dr-x run
dr-x sbin
dr-x snap
dr-x srv
dr-x sys
drwx tmp
dr-x usr
dr-x var
d-x wslHChoei
d-x wslHiIFii
d-x wslNMnMii
d-x wslclIMi
d-x wslmemhi
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (2)
?:
```

Рис. 4.3: Содержимое корневой директории

По заданию требуется написать аналог команды `ls`. Код скрипта находится на лист. 4.3. На рис. 4.3 можно увидеть как написанный скрипт выводит содержимое корневой директории, а так же типы файлов и права доступа к ним, в формате принятом в *Linux*.

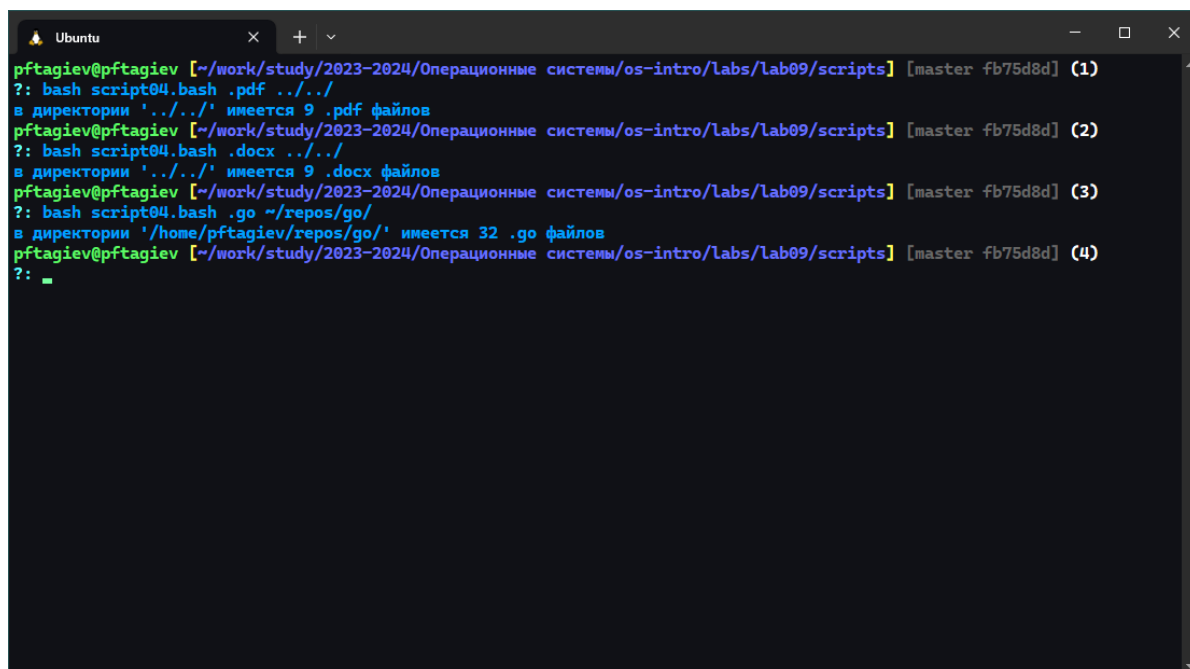
Листинг 4.3 Аналог команды ls

```
#!/usr/bin/bash

# если директория не указана работаем с текущей
cd ${1:-.}

# выводим и скрытые файлы
for file in .* *
do
    # выводит информацию о правах доступа
    # и типе файла в том же формате что и ls
    perm=(- - - -)
    [ -d $file ] && perm[0]=d
    [ -r $file ] && perm[1]=r
    [ -w $file ] && perm[2]=w
    [ -e $file ] && perm[3]=x
    perm=$(IFS= ; echo "${perm[*]}")
    echo $perm $file
done
```

4.4 Задание 4



The screenshot shows a terminal window titled 'Ubuntu' with a dark background. The user is in a directory and runs a script named 'script04.bash'. The script takes an extension as an argument and counts the number of files with that extension in the current directory. The output shows that there are 9 PDF files, 9 DOCX files, and 32 GO files.

```
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (1)
?: bash script04.bash .pdf ../../
в директории '../../' имеется 9 .pdf файлов
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (2)
?: bash script04.bash .docx ../../
в директории '../../' имеется 9 .docx файлов
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (3)
?: bash script04.bash .go ~/repos/go/
в директории '/home/pftagiev/repos/go/' имеется 32 .go файлов
pftagiev@pftagiev [~/work/study/2023-2024/Операционные системы/os-intro/labs/lab09/scripts] [master fb75d8d] (4)
?: _
```

Рис. 4.4: Количество файлов с заданным расширением

На лист. 4.4 можно увидеть код скрипта, который принимает расширение файла в формате: .расширение; и директорию. Затем выводит количество файлов с заданным расширением в указанной директории (рис. 4.4).

Листинг 4.4 Подсчет файлов с заданным расширением

```
#!/usr/bin/bash

# проверяет что расширение соответствует шаблону - '.расширение'
pattern=$1
if ! [[ $pattern =~ ^\..*$ ]]; then
    echo неверно задано расширение файла
    exit 1
fi

# если директория не указана работаем с текущей
dir=${2:-.}

total=$(find $dir -name "$pattern" -type f | wc -l)
echo "в директории '$dir' имеется $total $pattern файлов"
```

5 Ответы на контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

Командная оболочка — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В *Unix/Linux* часто используются следующие реализации командных оболочек:

- *sh* — стандартная командная оболочка *UNIX/Linux*, содержащая базовый, но при этом полный набор функций;
- *csh* — надстройка на оболочкой *Борна*, использующая *C*-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- *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?

POSIX (*Portable Operating System Interface for Computer Environments*) — это 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* операционных систем и переносимости прикладных программ на уровне исходного кода. *POSIX-совместимые* оболочки разработаны на базе оболочки *Корна*.

3. Как определяются переменные и массивы в языке программирования *bash*?

- Переменные: имя_переменной=значение
- Массивы: имя_массива=(элемент1 элемент2 элемент3 ...) или `declare -a` имя_массива

4. Каково назначение операторов `let` и `read`?

- Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (term), обычно целочисленный.
- Команда `read` позволяет читать значения переменных со стандартного ввода. Например: `read a b c`

5. Какие арифметические операции можно применять в языке программирования *bash*?

В языке программирования *bash* можно использовать следующие арифметические операции:

- Сложение: +
- Вычитание: -
- Умножение: *
- Деление: /
- Остаток от деления: %
- Возведение в степень: **

6. Что означает операция `(())`?

Эта конструкция во многом похожа на инструкцию `let`, внутри `(())` вычисляются арифметические выражения и возвращается их результат.

7. Какие стандартные имена переменных Вам известны?

- PATH — значением этой переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если

указанное имя программы или команды не содержит ни одного символа /.

- PS1 и PS2 — предназначены для отображения промптера командного процессора.
- 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.
- 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.`
- TERM — тип используемого терминала.
- LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

8. Что такое метасимволы?

Символы имеющие специальный смысл для командного процессора, например: ' < > * ? | \ " &.

9. Как экранировать метасимволы?

Чтобы экранировать метасимвол (т.е. снять с него специальный смысл) можно использовать `\`. Чтобы экранировать группу символов можно использовать *одинарные кавычки*.

10. Как создавать и запускать командные файлы?

Создать обычный файл, например командой `touch`. Добавить в первой строке шебанг `#!/`, после которого будет следовать путь к интерпретатору. Написать нужные команды. Добавить файлу право исполняться:

`chmod +x имя_файла`. Запустить его `./имя_файла` аргументы.... Чтобы не добавлять шебанг и права на исполнение можно запускать скрипт передавая его напрямую интерпретатору: `bash имя_файла`.

11. Как определяются функции в языке программирования `bash`?

Функцию можно определить так: `function имя() { тело }` или короче `имя() { тело }`.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом? Можно применить к файлу команду `test`, с флагами:

- `test -d` файл — истина, если файл существует и является каталогом.
- `test -f` файл — истина, если файл существует и является обычным файлом.

13. Каково назначение команд `set`, `typeset` и `unset`?

- `set` — изменяет значения внутренних переменных сценария.
- `typeset` — команды `declare` и `typeset` задают и/или накладывают ограничения на переменные.
- `unset` — удаляет переменную, фактически устанавливает ее значение в `null`.

14. Как передаются параметры в командные файлы?

Они присваиваются переменным `$1`, `$2`, ... `$n`, чтобы получить например 12 аргумент нужно использовать синтаксис `${12}`.

15. Назовите специальные переменные языка `bash` и их назначение.

- `$$` — **PID** текущего процесса.
- `$0` — **Имя скрипта**.
- `$1` до `$9` - **Позиционные параметры**, где `$1` это первый аргумент, и так далее.
- `@` — **Все позиционные параметры** как отдельные слова.
- `*` — **Все позиционные параметры** как одна строка.
- `#` — **Количество позиционных параметров**.

- `$?` — **Статус выхода** последней выполненной команды.
- `$-` — **Текущие опции** для оболочки.
- `$_` — **Последний аргумент** предыдущей команды.

6 Выводы

В этой лабораторной работе мы научились работать с *bash-скриптами*.
Познакомились с основными конструкциями языка *bash*.

Список литературы

1. Кулябов. Операционные системы. Москва: РУДН, 2016. 118 с.