

Лабораторная работа №13

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Полякова Юлия Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	14
5	Вывод	18

Список иллюстраций

3.1	Листинг script1.sh	7
3.2	Запуск script1.sh	8
3.3	example.txt	8
3.4	result1-1.txt	8
3.5	result1-2.txt	8
3.6	Листинг check_number.c	9
3.7	Листинг script2.sh	10
3.8	Запуск script2.sh	10
3.9	Листинг script3.sh - часть 1 (проверка аргум. и сущ-я файлов) . . .	11
3.10	Листинг script3.sh - часть 2 (основная часть с режимами)	11
3.11	Запуск script3.sh	12
3.12	Листинг script4.sh - часть 1	12
3.13	Листинг script4.sh - часть 2	13
3.14	Запуск script4.sh	13
3.15	Архив	13

Список таблиц

1 Цель работы

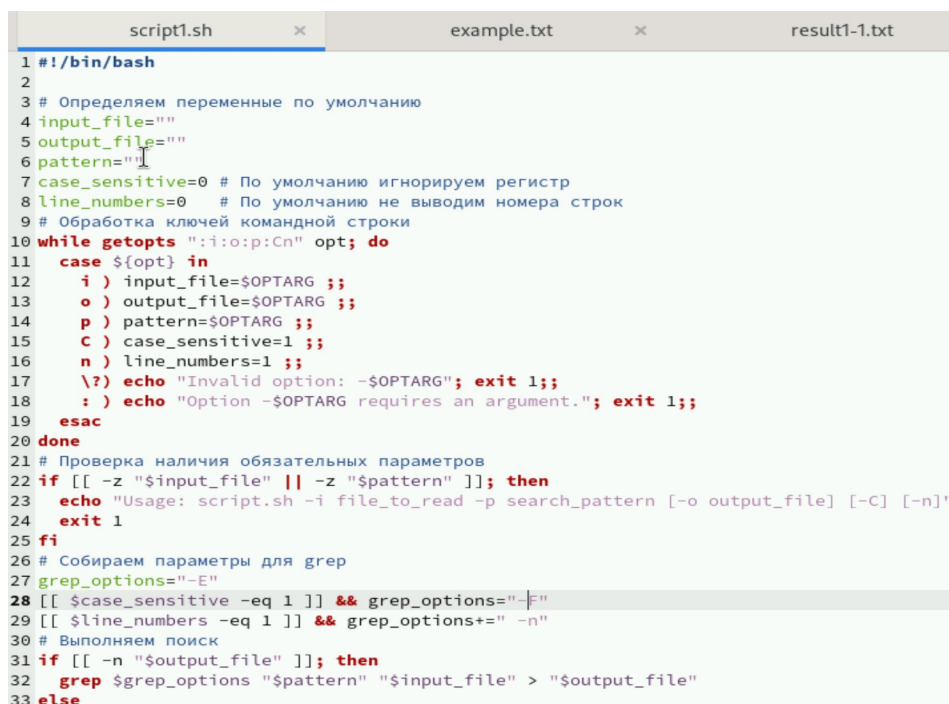
Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Ознакомиться с теоретическим материалом.
2. Написать программы.
3. Ответить на контрольные вопросы.

3 Выполнение лабораторной работы

1. Рассмотрим первый скрипт по заданию: Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами: `-i`inputfile — прочитать данные из указанного файла; `-o`outputfile — вывести данные в указанный файл; `-r`шаблон — указать шаблон для поиска; `-C` — различать большие и малые буквы; `-n` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-r`. (рис. 3.1).



```
1 #!/bin/bash
2
3 # Определяем переменные по умолчанию
4 input_file=""
5 output_file=""
6 pattern=""
7 case_sensitive=0 # По умолчанию игнорируем регистр
8 line_numbers=0 # По умолчанию не выводим номера строк
9 # Обработка ключей командной строки
10 while getopts ":i:o:p:Cn" opt; do
11     case ${opt} in
12         i ) input_file=$OPTARG ;;
13         o ) output_file=$OPTARG ;;
14         p ) pattern=$OPTARG ;;
15         C ) case_sensitive=1 ;;
16         n ) line_numbers=1 ;;
17         \?) echo "Invalid option: -$OPTARG"; exit 1;;
18         : ) echo "Option -$OPTARG requires an argument."; exit 1;;
19     esac
20 done
21 # Проверка наличия обязательных параметров
22 if [[ -z "$input_file" || -z "$pattern" ]]; then
23     echo "Usage: script.sh -i file_to_read -p search_pattern [-o output_file] [-C] [-n]"
24     exit 1
25 fi
26 # Собираем параметры для grep
27 grep_options=""
28 [[ $case_sensitive -eq 1 ]] && grep_options+="-C"
29 [[ $line_numbers -eq 1 ]] && grep_options+="-n"
30 # Выполняем поиск
31 if [[ -n "$output_file" ]]; then
32     grep $grep_options "$pattern" "$input_file" > "$output_file"
33 else
```

Рис. 3.1: Листинг script1.sh

2. Открываем доступ к файлу и запускаем. В начале мы запустили поиск “hello”

в файле “example.txt” без учета регистра и номеров строк в файл result1-1.txt
А затем проверили учет больших букв и номера строк и вывели результат
сначала в терминал, а потом в result1-2.txt (рис. 3.2)

```
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ touch script1.sh script2.sh script3.sh script4.sh
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ chmod +x script1.sh
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p hello -o result1-1.txt
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p Hello -o result1-2.txt
grep: conflicting matchers specified
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p Hello -Cn
grep: conflicting matchers specified
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p HELLO -Cn
grep: conflicting matchers specified
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p hello -o result1-1.txt
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p HELLO -Cn
4:HELLO
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p HELLO -Cn -o result1-2.txt
4:HELLO
yaryakova1@yaryakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$
```

Рис. 3.2: Запуск script1.sh

3. Файл example.txt (рис. 3.3)

script1.sh	example.txt
1 hello 2 world 3 hello 4 HELLO	

Рис. 3.3: example.txt

4. Файл result1-1.txt (рис. 3.4)

script1.sh	example.txt	result1-1.txt
1 hello 2 hello		

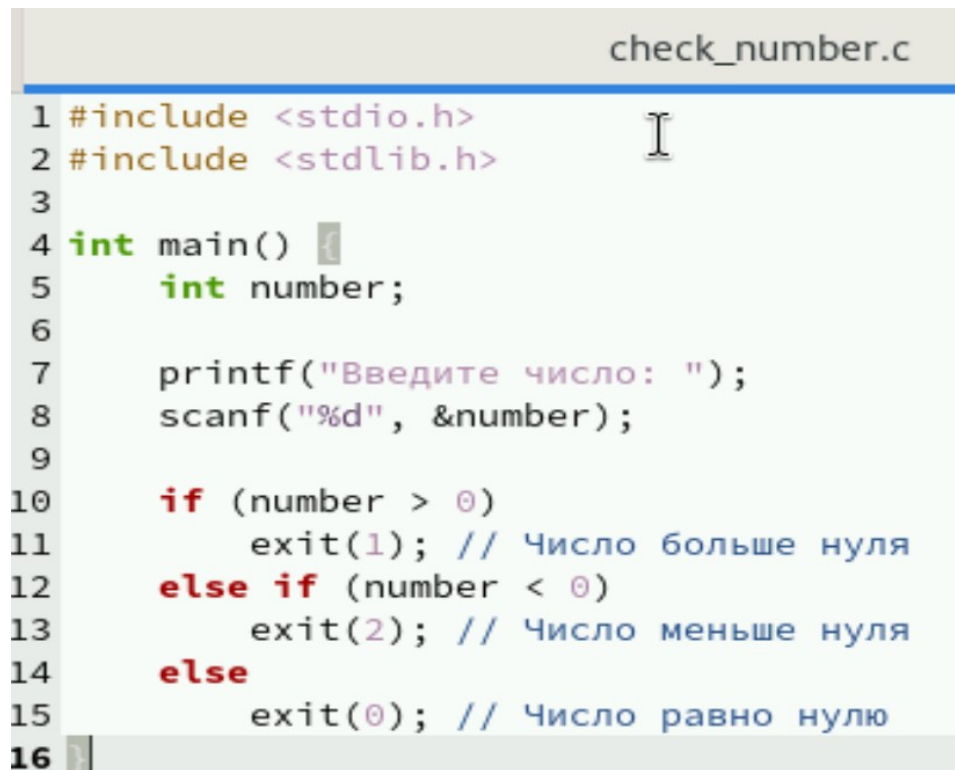
Рис. 3.4: result1-1.txt

5. Файл result1-2.txt (рис. 3.5)

script1.sh	example.txt	result1-1.txt	result1-2.txt
1 4:HELLO			

Рис. 3.5: result1-2.txt

6. Рассмотрим второе задание. Сначала надо было “Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку.” На фото листинг этой программы (рис. 3.6)



```
check_number.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5     int number;
6
7     printf("Введите число: ");
8     scanf("%d", &number);
9
10    if (number > 0)
11        exit(1); // Число больше нуля
12    else if (number < 0)
13        exit(2); // Число меньше нуля
14    else
15        exit(0); // Число равно нулю
16
```

Рис. 3.6: Листинг `check_number.c`

7. Теперь рассмотрим сам командный файл. Задание: Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено. (рис. 3.7)

```

script2.sh
1 #!/bin/bash
2
3 ./check_number
4
5 # Анализируем статус выхода
6 case $? in
7     1)
8         echo "Вы ввели положительное число."
9         ;;
10    2)
11        echo "Вы ввели отрицательное число."
12        ;;
13    0)
14        echo "Вы ввели ноль."
15        ;;
16    *)
17        echo "Ошибка!"
18        ;;
19 esac

```

Рис. 3.7: Листинг script2.sh

8. Компилируем С-файл с помощью gcc, открываем доступ к командному файлу и запускаем. Тестируем с разными числами (рис. 3.8)

```

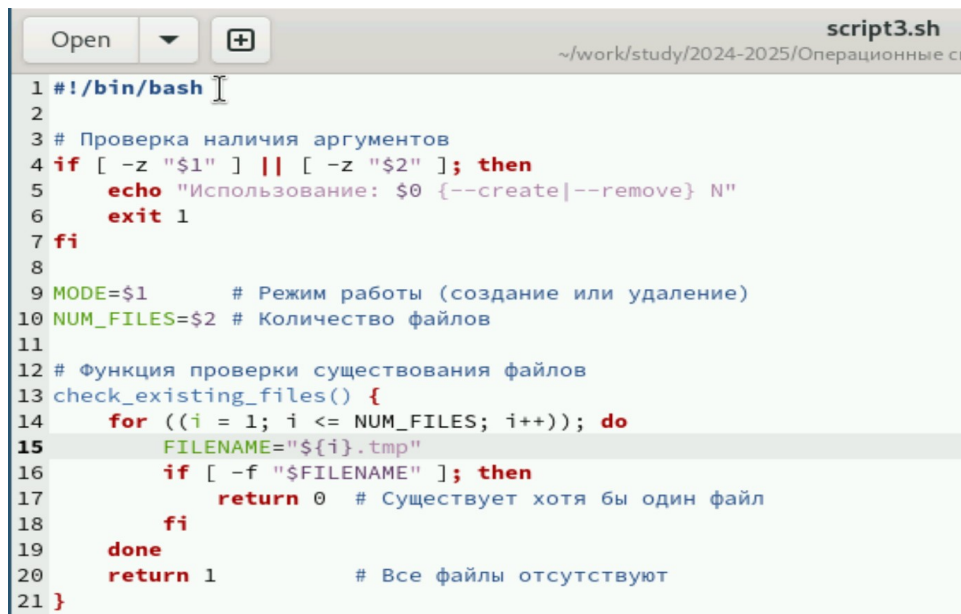
yarpolyakova1@yarpolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab1$ gcc check_number.c -o check_number
yarpolyakova1@yarpolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab1$ ./script2.sh
Введите число: -5
Вы ввели отрицательное число.
yarpolyakova1@yarpolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab1$ ./script2.sh
Введите число: 12
Вы ввели положительное число.
yarpolyakova1@yarpolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab1$ ./script2.sh
Введите число: 0
Вы ввели ноль.
yarpolyakova1@yarpolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab1$

```

chmod +x

Рис. 3.8: Запуск script2.sh

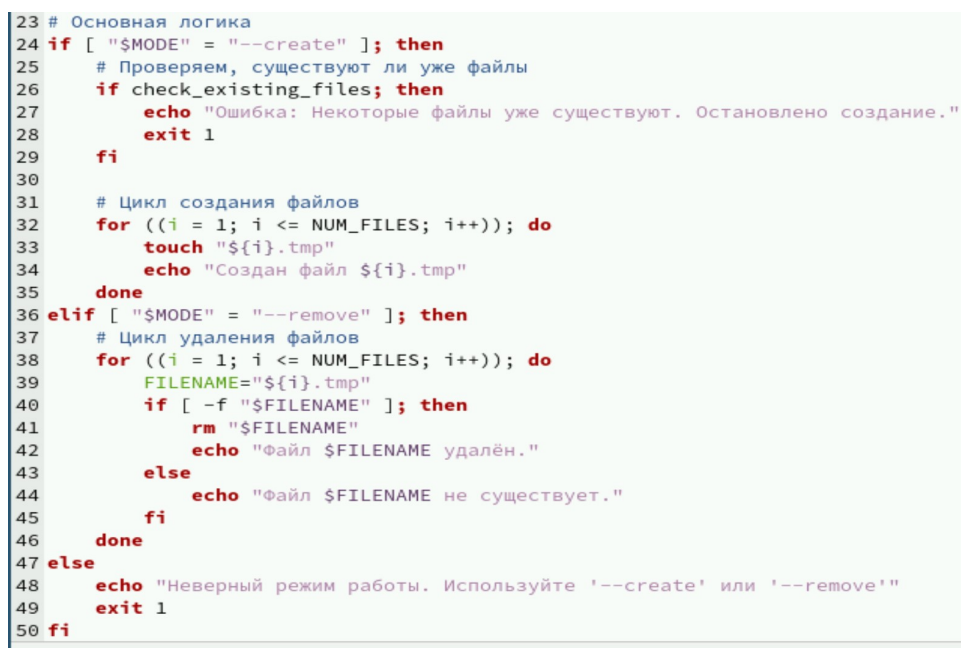
9. Рассмотрим третий скрипт. Задание: Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). Я реализовала 2 режима - создание и удаление (рис. 3.9)



```
1 #!/bin/bash
2
3 # Проверка наличия аргументов
4 if [ -z "$1" ] || [ -z "$2" ]; then
5     echo "Использование: $0 [--create|--remove] N"
6     exit 1
7 fi
8
9 MODE=$1      # Режим работы (создание или удаление)
10 NUM_FILES=$2 # Количество файлов
11
12 # Функция проверки существования файлов
13 check_existing_files() {
14     for ((i = 1; i <= NUM_FILES; i++)); do
15         FILENAME="${i}.tmp"
16         if [ -f "$FILENAME" ]; then
17             return 0 # Существует хотя бы один файл
18         fi
19     done
20     return 1        # Все файлы отсутствуют
21 }
```

Рис. 3.9: Листинг script3.sh - часть 1 (проверка аргум. и сущ-я файлов)

Основная часть кода script3.sh, работа с режимами (рис. 3.10)



```
23 # Основная логика
24 if [ "$MODE" = "--create" ]; then
25     # Проверяем, существуют ли уже файлы
26     if check_existing_files; then
27         echo "Ошибка: Некоторые файлы уже существуют. Остановлено создание."
28         exit 1
29     fi
30
31     # Цикл создания файлов
32     for ((i = 1; i <= NUM_FILES; i++)); do
33         touch "${i}.tmp"
34         echo "Создан файл ${i}.tmp"
35     done
36 elif [ "$MODE" = "--remove" ]; then
37     # Цикл удаления файлов
38     for ((i = 1; i <= NUM_FILES; i++)); do
39         FILENAME="${i}.tmp"
40         if [ -f "$FILENAME" ]; then
41             rm "$FILENAME"
42             echo "Файл $FILENAME удалён."
43         else
44             echo "Файл $FILENAME не существует."
45         fi
46     done
47 else
48     echo "Неверный режим работы. Используйте '--create' или '--remove'"
49     exit 1
50 fi
```

Рис. 3.10: Листинг script3.sh - часть 2 (основная часть с режимами)

10. Открываем доступ к исполнению, тестируем на создании и удалении 3-х файлов (рис. 3.11)

```

yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ chmod +x script3.sh
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script3.sh --create 3
Создан файл 1.tmp
Создан файл 2.tmp
Создан файл 3.tmp
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script3.sh --remove 3
Файл 1.tmp удалён.
Файл 2.tmp удалён.
Файл 3.tmp удалён.
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$

```

Рис. 3.11: Запуск script3.sh

11. Рассмотрим 4-ый командный файл. Задание: Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). (рис. 3.12)

```

Open script4.sh
~/work/study/2024-2025/Операционные системы/os-intro
1 #!/bin/bash
2
3 SOURCE_DIRECTORY=""      # Исходный каталог
4 OUTPUT_DIRECTORY=""     # Целевой каталог
5 ARCHIVE_PREFIX="_recently_modified" # Префикс названия архива
6 ARCHIVE_EXTENSION=".tar.gz" # Расширение архива
7
8 # Разбор аргументов командной строки
9 while getopts s:o:h FLAG; do
10     case $FLAG in
11         s) SOURCE_DIRECTORY=$OPTARG ;;
12         o) OUTPUT_DIRECTORY=$OPTARG ;;
13         h) echo "Использование: $0 -s source_dir -o output_dir"; exit 0 ;;
14         *) echo "Неправильный флаг. Попробуйте '-h' для справки."; exit 1 ;;
15     esac
16 done
17
18 # Проверка наличия обязательных параметров
19 if [ -z "$SOURCE_DIRECTORY" ] || [ -z "$OUTPUT_DIRECTORY" ]; then
20     echo "Необходимо задать источник и целевой каталог."
21     exit 1
22 fi
23
24 # Извлекаем базовое имя каталога (без пути)
25 BASE_DIRNAME=$(basename "$SOURCE_DIRECTORY")
26
27 # Формируем полное имя архива с именем каталога
28 ARCHIVE_NAME="${BASE_DIRNAME}${ARCHIVE_PREFIX}${ARCHIVE_EXTENSION}"
29
30 # Переход в рабочий каталог
31 cd "$SOURCE_DIRECTORY" || exit 1
32
33 # Поиск файлов, изменённых менее недели назад
34 FILES_TO_ARCHIVE=$(find . -type f -mtime -7)

```

Рис. 3.12: Листинг script4.sh - часть 1

Конец скрипта с архивацией (рис. 3.13)

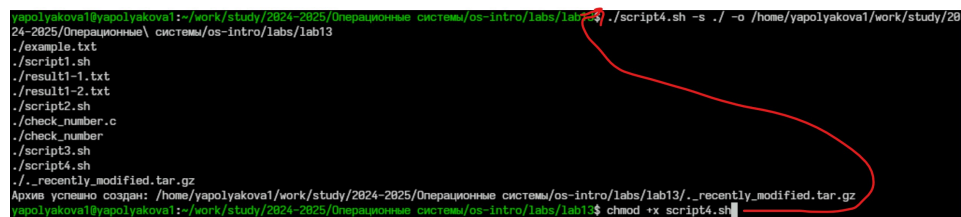
```

35
36 # Проверка наличия свежих файлов
37 if [ -z "$FILES_TO_ARCHIVE" ]; then
38     echo "Нет файлов, изменённых менее недели назад."
39     exit 0
40 fi
41
42 # Создание архива
43 tar czvf "$OUTPUT_DIRECTORY/$ARCHIVE_NAME" $FILES_TO_ARCHIVE
44
45 echo "Архив успешно создан: $OUTPUT_DIRECTORY/$ARCHIVE_NAME"

```

Рис. 3.13: Листинг script4.sh - часть 2

12. Открываем доступ к исполнению, запускаем, архивируем файлы этой лабораторной работы (рис. 3.14)



```

yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script4.sh -s ./ -o /home/yapolyakova1/work/study/2024-2025/Операционные системы/os-intro/labs/lab13
./example.txt
./script1.sh
./result1-1.txt
./result1-2.txt
./script2.sh
./check_number.c
./check_number
./script3.sh
./script4.sh
./_recently_modified.tar.gz
Архив успешно создан: /home/yapolyakova1/work/study/2024-2025/Операционные системы/os-intro/labs/lab13/_recently_modified.tar.gz
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ chmod +x script4.sh

```

Рис. 3.14: Запуск script4.sh

13. Архив действительно создался (рис. 3.15)

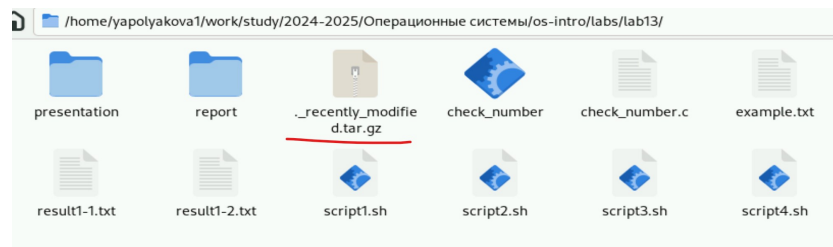


Рис. 3.15: Архив

4 Контрольные вопросы

1. Каково предназначение команды getopt?

Команда `getopts` предназначена для удобной обработки аргументов командной строки в сценариях оболочки (shell scripts). Она помогает разбивать переданные аргументы на отдельные элементы и проверять правильность передачи флагов и значений. Обычно применяется совместно с циклом `while`, что позволяет автоматизировать обработку сложных комбинаций параметров, обеспечивая безопасность и ясность синтаксиса.

Пример использования:

```
while getopt ab:c OPT; do
    case $OPT in
        a) echo "Опция A была указана" ;;
        b) echo "Опция B имеет значение: $OPTARG" ;;
        c) echo "Опция C указана" ;;
        ?) echo "Неправильно использован флаг" ;;
    esac
done
```

2. Какое отношение метасимволы имеют к генерации имён файлов?

Метасимволы (такие как звездочка `*`, знак вопроса `?`, квадратные скобки `[]`) играют важную роль в построении масок для поиска и подбора файлов в операционной системе. Они позволяют удобно находить группы файлов по общим признакам.

Примеры:

- Маска *.txt выберет все файлы с расширением .txt.
- Маска file?.dat подберёт файлы с названием длиной в четыре символа, начинающиеся с file и оканчивающиеся на .dat.

Эти маски часто применяются в командах типа ls, rm, cp, что значительно ускоряет выбор файлов.

3. Какие операторы управления действиями вы знаете?

Операторы управления помогают контролировать поток выполнения инструкций в сценарии оболочки. Основные операторы включают:

- **if** — оператор условного ветвления, позволяющий исполнять блок кода только при выполнении какого-либо условия.
- **case** — оператор множественного выбора, похожий на switch-case в языках программирования.
- **for, while, until** — операторы организации циклов.
- **break, continue** — операторы изменения хода выполнения цикла.
- **&&** (логическое AND) и **||** (логическое OR) — используются для объединения нескольких условий или команд.

4. Какие операторы используются для прерывания цикла?

Для остановки цикла используются два оператора:

- **break** — немедленно выходит из ближайшего охватывающего цикла (for, while, until).
- **continue** — пропускает оставшуюся часть итерации и переходит сразу к следующей итерации цикла.

Пример использования операторов:

```
for i in $(seq 1 10); do
    if [ $i -gt 5 ]; then break; fi
    echo $i
done
```

5. Для чего нужны команды `false` и `true`?

Командами `false` и `true` являются стандартные команды оболочки, используемые для простых тестов условий. Их назначение следующее:

- **false** — всегда возвращает ненулевой код завершения (обычно 1), обозначая ошибку или ложное условие.
- **true** — всегда возвращает нулевой код завершения (0), обозначая успешное завершение или истинное условие.

Они полезны в случаях, когда необходимо организовать простую проверку или задать заведомо известное условие.

Пример использования:

```
if true; then
    echo "Это всегда правда"
fi
```

6. Что означает строка `if test -f man/i.$s`, встречающаяся в командном файле?

Эта строка представляет собой проверку существования файла с определенным форматом. Рассмотрим подробнее:

- `test -f filename` — проверяет, существует ли файл и является ли он обычным файлом (не каталогом, символьной ссылкой и т.п.).
- `man$s/i.$s` — выражение, генерирующее динамическое имя файла. Здесь:
 - `$s` — вероятно, какая-то переменная, заменяющая расширение или категорию файла.

– $\$i$ — возможно, индекс или порядковый номер файла.

Таким образом, данная конструкция проверяет, существует ли конкретный файл с определенной структурой имени.

7. Объясните различия между конструкциями `while` и `until`.

Конструкция `while`: цикл выполняется, пока условие истинно. Иначе говоря, цикл выполняется снова и снова, пока условие возвращает успех (нулевой код завершения).

Конструкция `until`: цикл выполняется, пока условие ложно. Таким образом, цикл выполняется повторно, пока условие возвращает неудачу (ненулевой код завершения).

5 Вывод

Были изучены основы программирования в оболочке ОС UNIX. Мы научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.