**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт компьютерных наук
Кафедра автоматизированных систем управления

# ЛАБОРАТОРНАЯ РАБОТА № 4

по дисциплине: «ДПО Интаро Софт»
на тему: «Создание и использование сценариев(скриптов) в Linux».

Студент АС-23-1 (группа) _____ (подпись, дата) Кирнос Б. В. (фамилия, инициалы)

Руководитель

к.т.н., доцент кафедры АСУ (ученая степень, ученое звание) _____ (подпись, дата) Кургасов В.В. (фамилия, инициалы)

Липецк 2024

Оглавление



**Цель работы**: получение практических навыков по написанию Bash- скриптов для ОС Linux.

## Задание 1

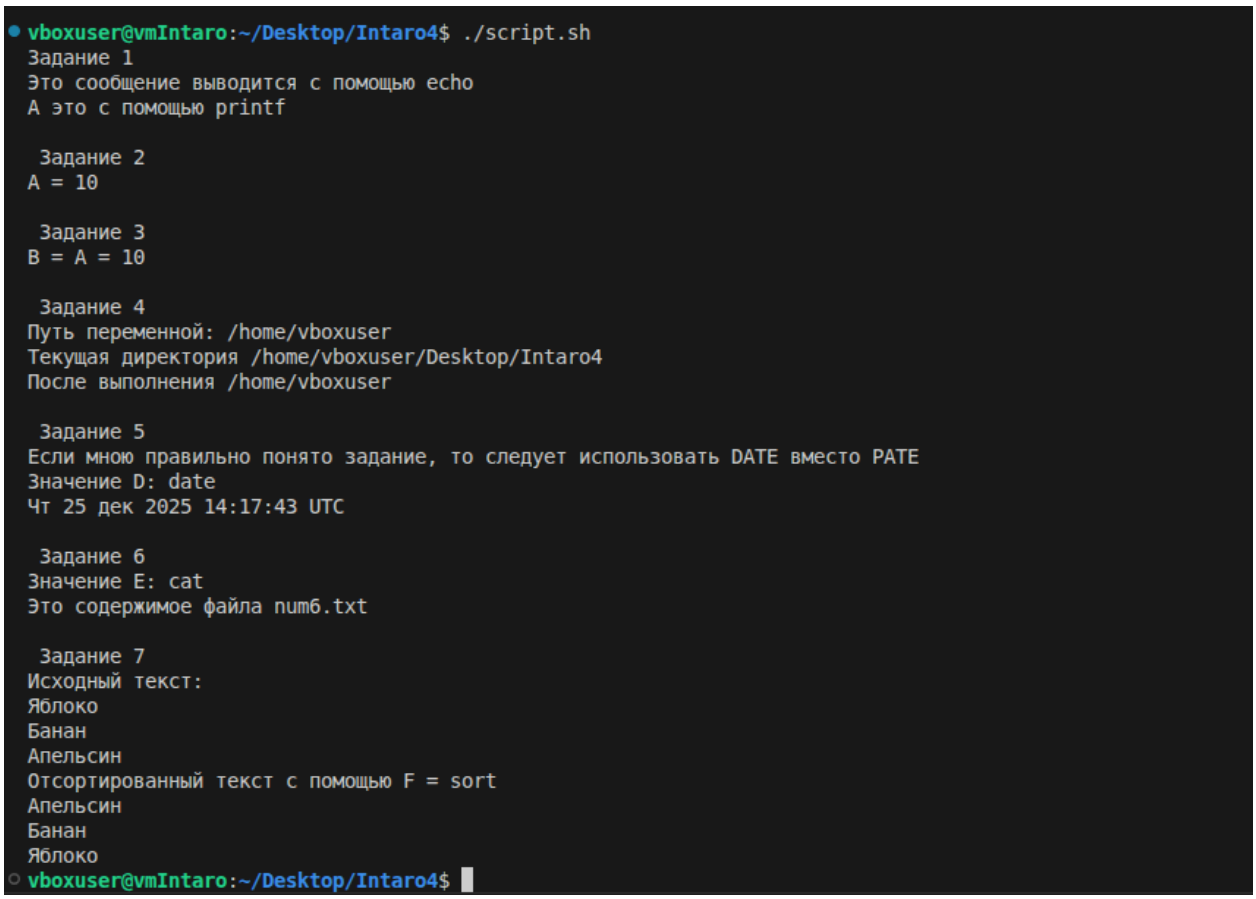


Рис.1 – Результат выполнения скрипта для задания 1

Код скрипта:

```
printf "Задание 1 \n"
echo "Это сообщение выводится с помощью echo";
printf "А это с помощью printf \n";

printf "\n Задание 2 \n";
A=10;
echo "A = $A"

printf "\n Задание 3 \n";
B=$A;
echo "B = A = $B";

printf "\n Задание 4 \n";
C=$HOME;
echo "Путь переменной: $C";
echo "Текущая директория $(pwd)";
cd "$C";
echo "После выполнения $(pwd)";
```

```
printf "\n Задание 5 \n";
D="date";
echo "Если мною правильно понято задание, то следует использовать DATE вместо PATE";
echo "Значение D: $D";
$D;

printf "\n Задание 6 \n";
echo "Это содержимое файла num6.txt" > num6.txt;
E="cat";
echo "Значение E: $E";
$E num6.txt;

printf "\n Задание 7 \n";
echo -e "Яблоко\nБанан\nАпельсин" > num7.txt;
echo "Исходный текст:";
cat num7.txt;
F="sort";
echo "Отсортированный текст с помощью F = $F";
$F num7.txt;
```

## Задание 2

```
vboxuser@vmIntaro:~/Desktop/Intaro4$ ./script.sh text.txt password
Задание 1
Введите значение переменной 1
Вы ввели: 1

 Задание 2
Как тебя зовут? Вогуа
Привет, Вогуа !

 Задание 3
Вседите число X 5
Вседите число Y 5
Сумма: 10 Частное: 1.00

 Задание 4
Радиус: 4
Высота: 3
Объем цилиндра: 150.7920

 Задание 5
Имя программы: ./script.sh
Количество аргументов: 2
Аргументы: text.txt password

 Задание 6
Файл для задачи 6 не указан или не существует.
```

```
 Задание 7
--- Файл: fileRead.txt ---
--- Файл: fileWrite.txt ---
--- Файл: sorted.txt ---
123
--- Файл: test.txt ---
123
--- Файл: text2.txt ---
text file 2

 Задание 8
Введите число для сравнения: 10
Число не превышает 100

 Задание 9
Введите год: 2026
2026 — не високосный

 Задание 10
Введите два числа и диапазон (мин макс):
1 2 1 10
V1=1, V2=2 (в диапазоне)
V1=2, V2=3 (в диапазоне)
V1=3, V2=4 (в диапазоне)
V1=4, V2=5 (в диапазоне)
V1=5, V2=6 (в диапазоне)
V1=6, V2=7 (в диапазоне)
V1=7, V2=8 (в диапазоне)
V1=8, V2=9 (в диапазоне)
V1=9, V2=10 (в диапазоне)

 Задание 11
/etc:
total 1164
drwxr-xr-x 141 root                root                12288 дек 25 16:07 .
drwxr-xr-x  23 root                root                 4096 дек 24 13:31 ..
-rw-r--r--   1 root                root                 3444 июл  5  2023 addu
ser.conf
drwxr-xr-x   3 root                root                 4096 авг  5 16:50 alsa
drwxr-xr-x   2 root                root                 4096 дек 24 13:40 alte
rnatives
-rw-r--r--   1 root                root                  335 апр  8  2024 anac
rontab
-rw-r--r--   1 root                root                  433 апр  8  2024 apg.
conf
drwxr-xr-x   5 root                root                 4096 авг  5 16:50 apm
drwxr-xr-x   2 root                root                 4096 дек 24 13:47 appa
rmor
drwxr-xr-x   9 root                root                12288 дек 24 13:48 appa
rmor.d
drwxr-xr-x   3 root                root                 4096 авг  5 16:51 appo
rt
drwxr-xr-x   9 root                root                 4096 дек 24 13:31 apt
drwxr-xr-x   3 root                root                 4096 авг  5 16:51 avah
i
-rw-r--r--   1 root                root                 2319 мар 31  2024 bash
.bashrc
-rw-r--r--   1 root                root                   45 янв 24  2020 bash
_completion
drwxr-xr-x   2 root                root                 4096 дек 25 00:25 bash
_completion.d
-rw-r--r--   1 root                root                  367 авг  2  2022 bind
resvport.blacklist
```

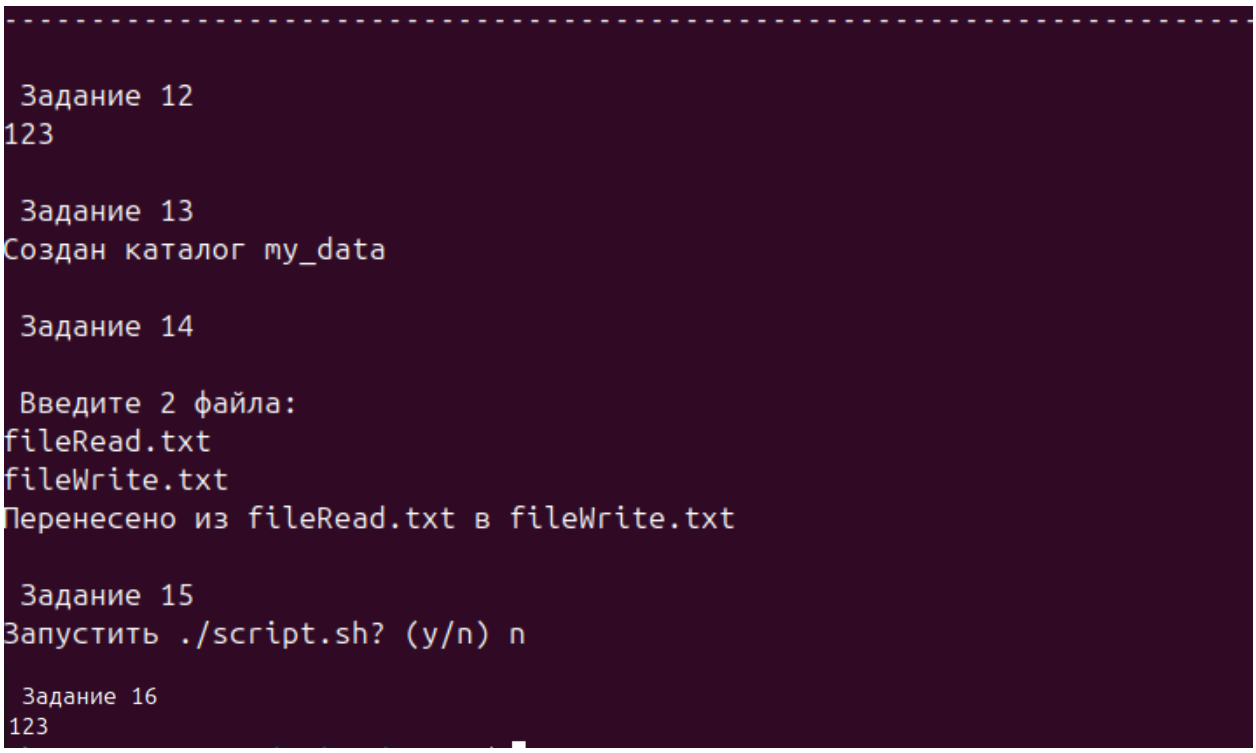


Рис.2 – Результат выполнения скрипта для задания 2

Код скрипта:

```
echo "Задание 1";
echo -n "Введите значение переменной ";
read NUM1_VAR;
echo "Вы ввели: $NUM1_VAR";

printf "\n Задание 2 \n";
echo -n "Как тебя зовут? ";
read USERNAME;
echo "Привет, $USERNAME !";

printf "\n Задание 3 \n";
echo -n "Вседите число X "; read X;
echo -n "Вседите число Y "; read Y;
SUMM_EXPR=$(expr $X + $Y);
DIV_BC=$(echo "scale=2; $X / $Y" | bc);
echo "Сумма: $SUMM_EXPR Частное: $DIV_BC"
```

```
printf "\n Задание 4 \n";
echo -n "Радиус: " ; read RADIUS;
echo -n "Высота: " ; read HEIGHT;
VOL=$(echo "scale=2; 3.1415 * $RADIUS * $RADIUS * $HEIGHT" | bc)
echo "Объем цилиндра: $VOL";

printf "\n Задание 5 \n";
echo "Имя программы: $0";
echo "Количество аргументов: $#";
echo "Аргументы: $@";

printf "\n Задание 6 \n";
if [ -n "$1" ] && [ -f "$1" ]; then
   cat "$1"
   sleep 2
   clear
else
   echo "Файл для задачи 6 не указан или не существует."
fi

printf "\n Задание 7 \n";
for file in *.txt; do
   if [ -e "$file" ]; then
      echo "--- Файл: $file ---"
      cat "$file"
   fi
done

printf "\n Задание 8 \n";
LIMIT=100
echo -n "Введите число для сравнения: " ; read NUM
if [ "$NUM" -gt "$LIMIT" ]; then
   echo "Число больше $LIMIT"
```

```
else
   echo "Число не превышает $LIMIT"
fi

printf "\n Задание 9 \n";
echo -n "Введите год: " ; read YEAR
if [ $((YEAR % 4)) -eq 0 ] && ([ $((YEAR % 100)) -ne 0 ] || [ $((YEAR % 400)) -eq 0 ]); then
   echo "$YEAR — високосный"
else
   echo "$YEAR — не високосный"
fi

printf "\n Задание 10 \n";
echo "Введите два числа и диапазон (мин макс):"
read V1 V2 MIN MAX
while [ "$V1" -ge "$MIN" ] && [ "$V1" -le "$MAX" ] && [ "$V2" -ge "$MIN" ] && [ "$V2" -
le "$MAX" ]; do
   echo "V1=$V1, V2=$V2 (в диапазоне)"
   ((V1++))
   ((V2++))
done

printf "\n Задание 11 \n";
PASS="password"
if [ "$2" == "$PASS" ]; then
   ls -laR /etc | more;
else
   echo "Неверный пароль (передайте его вторым аргументом)."
fi

printf "\n Задание 12 \n";
FILE_TEST="test.txt"
if [ -f "$FILE_TEST" ]; then cat "$FILE_TEST"; else echo "Файла $FILE_TEST нет"; fi
```

```
printf "\n Задание 13 \n";
NAME="my_data"
if [ -d "$NAME" ]; then
   [ -r "$NAME" ] && ls "$NAME"
elif [ -f "$NAME" ]; then
   cat "$NAME"
else
   mkdir "$NAME" && echo "Создан каталог $NAME"
fi

printf "\n Задание 14 \n";
printf "\n Введите 2 файла: \n";
read F1;
read F2;
if [ -r "$F1" ] && [ -w "$F2" ]; then
   cat "$F1" > "$F2"
   echo "Перенесено из $F1 в $F2"
else
   echo "Ошибка атрибутов или доступа."
fi

printf "\n Задание 15 \n";
CMD="./script.sh"
if [ -x "$CMD" ]; then
   echo -n "Запустить $CMD? (y/n) "; read ANS
   [ "$ANS" == "y" ] && $CMD
fi

printf "\n Задание 16 \n";
if [ -s "$1" ]; then
   sort -k1,1 "$1" > sorted.txt
   cat sorted.txt
fi
```

**Задание 3**

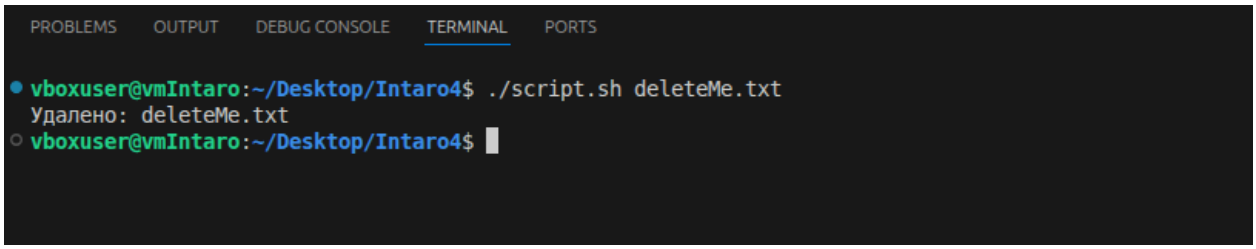


Рис.3 – Результат выполнения скрипта задания 3-1

На рисунке показан результат удаления конкретного файла, однако логика для удаления всего содержимого (*) и всего кроме (-) реализована.

Код скрипта 1:

```
TARGET_DIR="$HOME"

EXCLUDE_PATTERN=""
for arg in "$@"; do
    if [[ "$arg" == -* ]]; then
        EXCLUDE_PATTERN="${arg#-}"
        break
    fi
done

for item in "$@"; do
    if [[ "$item" == -* ]]; then
        continue
    fi

    FILE_PATH="$TARGET_DIR/$item"

    if [ -e "$FILE_PATH" ]; then

        if [[ -n "$EXCLUDE_PATTERN" && "$item" == "$EXCLUDE_PATTERN"* ]]; then
            echo "Пропуск (исключено): $item"
            continue
        fi

        rm -rf "$FILE_PATH"
        echo "Удалено: $item"
    fi
done
```


Рис.3 – Результат выполнения скрипта задания 3-2

Код скрипта 2:

```
#!/bin/bash

USER_NAME=$1
grep "^$USER_NAME:" /etc/passwd | cut -d: -f6
```

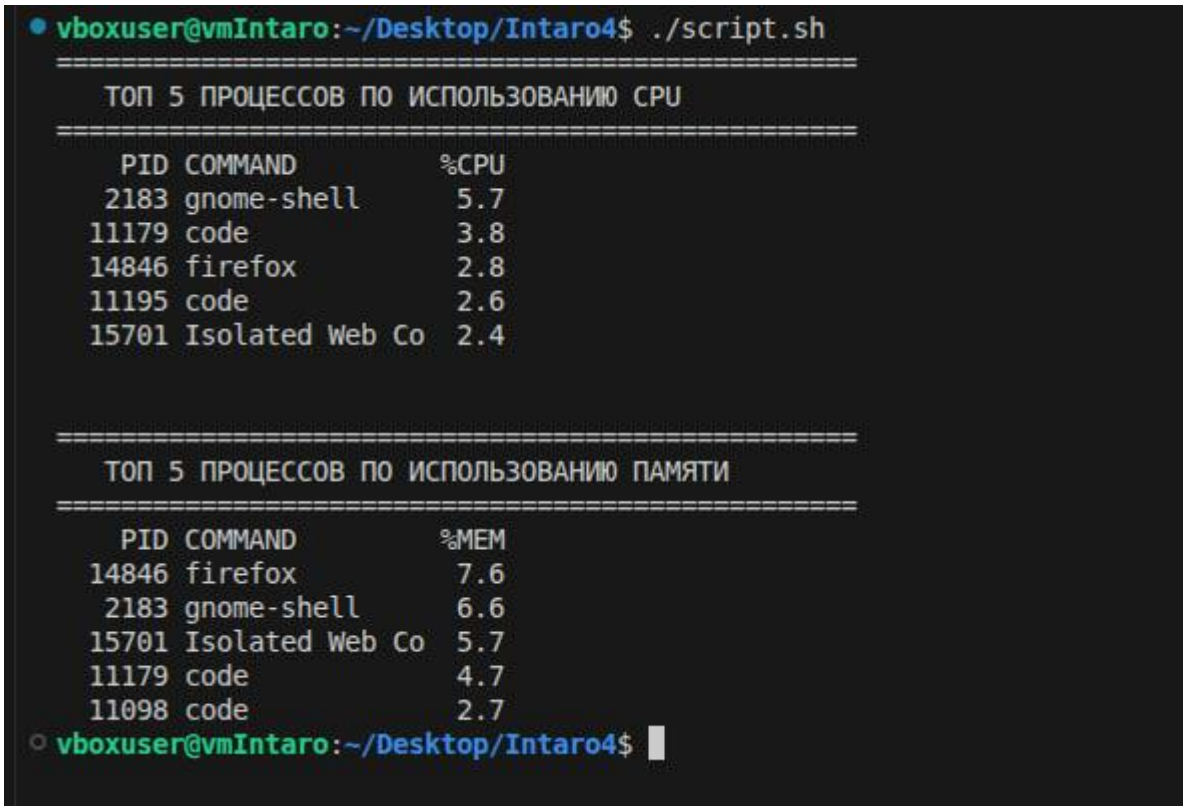


Рис.3 – Результат выполнения скрипта задания 3-3

Код скрипта 3:

```
#!/bin/bash

TOP_N=5

echo "=============================================="
echo "   ТОП $TOP_N ПРОЦЕССОВ ПО ИСПОЛЬЗОВАНИЮ CPU"
```

```
echo "================================================="
ps -eo pid,comm,%cpu --sort=-%cpu | head -n $((TOP_N + 1))

echo -e "\n"

echo "================================================="
echo "   ТОП $TOP_N ПРОЦЕССОВ ПО ИСПОЛЬЗОВАНИЮ ПАМЯТИ"
echo "================================================="
ps -eo pid,comm,%mem --sort=-%mem | head -n $((TOP_N + 1))
```

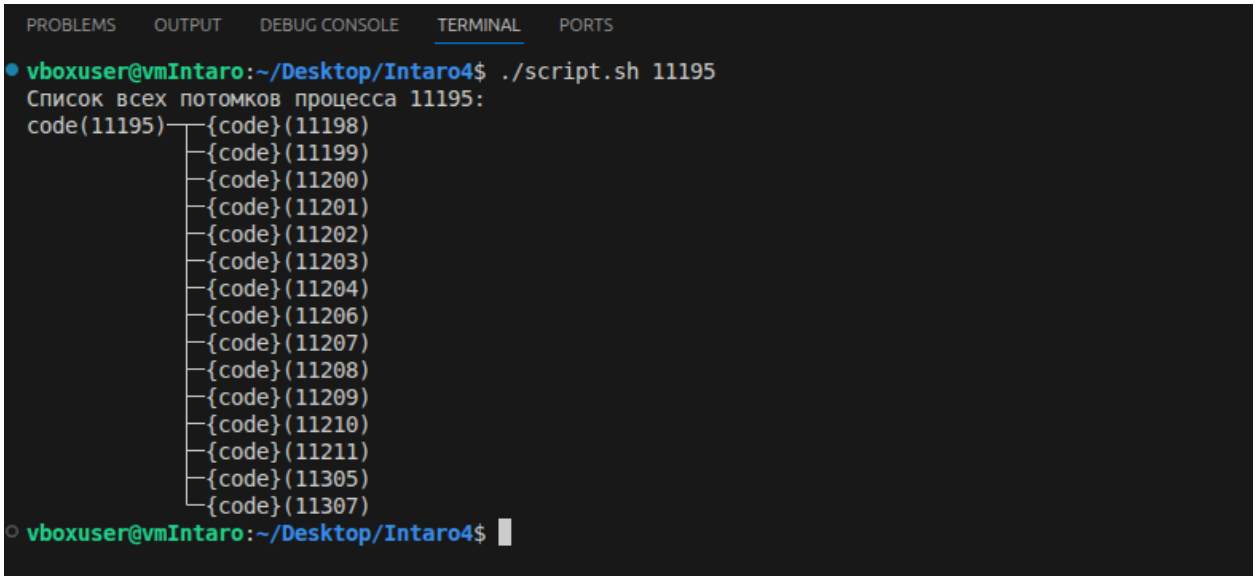


Рис.3 – Результат выполнения скрипта задания 3-4

Код скрипта 4:

```
#!/bin/bash
if [ -z "$1" ]; then
    echo "Ошибка: Не указан PID."
    echo "Использование: $0 <PID>"
    exit 1
fi

PID=$1
if ! ps -p "$PID" > /dev/null 2>&1; then
    echo "Ошибка: Процесс с PID $PID не существует."
    exit 1
```

```
fi

echo "Список всех потомков процесса $PID:"

pstree -p "$PID"
```


Рис.3 – Результат выполнения скрипта задания 3-5

Код скрипта 5:

```
TARGET_DIR="/tmp"

FIRST_DAY_OF_MONTH=$(date +%Y-%m-01)

echo "Поиск файлов в $TARGET_DIR..."

find "$TARGET_DIR" -type f -newermt "$FIRST_DAY_OF_MONTH" -mtime +7 2>/dev/null
| while read -r FILE; do

   if grep -q "test" "$FILE"; then
      sed -i 's/test/tset/g' "$FILE"
      echo "Обработан файл: $FILE"
   fi

done

echo "Работа завершена."
```

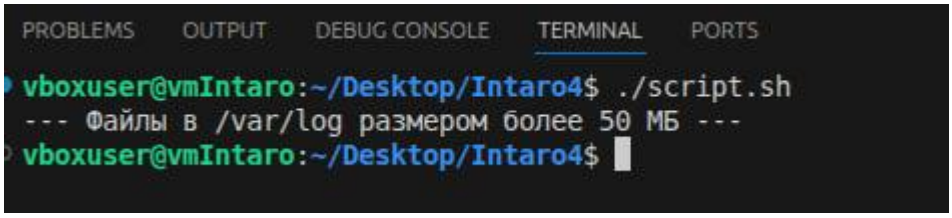


Рис.3 – Результат выполнения скрипта задания 3-6

Код скрипта 6:

```
#!/bin/bash
LOG_DIR="/var/log"

echo "--- Файлы в $LOG_DIR размером более 50 МБ ---"
find "$LOG_DIR" -type f -size +50M -exec ls -lh {} + 2>/dev/null | awk '{print $5, $9}'
```

Код скрипта 7:

```
if [ "$#" -lt 2 ]; then
   echo "Использование: $0 '<команда>' <сервер1> <сервер2> ..."
   echo "Пример: $0 'uptime' 192.168.1.10 192.168.1.11"
   exit 1
fi

REMOTE_CMD=$1

shift

echo "Запуск команды: '$REMOTE_CMD'"
echo "----------------------------------------"

for SERVER in "$@"; do
   echo "[Сервер: $SERVER]"

   ssh -o ConnectTimeout=5 "$SERVER" "$REMOTE_CMD"

   if [ $? -eq 0 ]; then
      echo "Статус: Успешно"
   else
      echo "Статус: Ошибка выполнения на $SERVER"
   fi
   echo "----------------------------------------"
done
```

**Вывод**

В ходе выполнения лабраторной работы были изучены основы программирования на языке командной оболочки Bash.
Мною были освоены следующие ключевые навыки:

1. **Работа с переменными и окружением:** изучены механизмы присваивания значений, работа с арифметическими операциями (утилиты expr и bc) и обработка пользовательского ввода с помощью read.
2. **Управление потоком выполнения:** реализованы алгоритмы с использованием условных операторов (if-else), циклов (for, while) и операторов выбора.
3. **Автоматизация системных задач:** разработаны скрипты для администрирования файловой системы (поиск файлов по атрибутам через find), управления процессами (ps, pstree, pgrep) и обработки системных файлов (парсинг /etc/passwd).
4. **Текстовая обработка:** освоены инструменты потокового редактирования (sed) и фильтрации данных (grep, awk), что позволило автоматизировать поиск и замену строк в файлах по сложным критериям.
5. **Системное администрирование:** реализованы механизмы удаленного выполнения команд через протокол SSH и мониторинг ресурсов системы (анализ логов и потребления памяти).

Практическая значимость работы заключается в понимании принципов автоматизации рутинных задач в среде Linux, что является критически важным навыком для системного администрирования и разработки ПО.

**Контрольные вопросы**

**1. В чём отличие пользовательских переменных от переменных среды?**

- **Пользовательские переменные** доступны только в текущем экземпляре оболочки (скрипта).
- **Переменные среды (Environment variables)** экспортируются и доступны всем дочерним процессам, запущенным из этой оболочки. Для превращения обычной переменной в переменную среды используется команда export.

**2. Математические операции в SHELL.** Поскольку оболочка воспринимает всё как строки, для математики нужны специальные утилиты:

- $(( expression )) — встроенная арифметика Bash (только целые числа).
- expr — внешняя утилита для простых операций.
- bc — калькулятор произвольной точности (поддерживает дробные числа).

**3. Условные операторы в SHELL.** Основная конструкция: if [ условие ]; then ... elif ... else ... fi. Также используется case для множественного выбора по шаблону.

**4. Принципы построения простых и составных условий.**

- **Простые:** проверяют один атрибут (например, -f file — проверка существования файла).
- **Составные:** объединяются логическими операторами:
  - -a или && (логическое И).
  - -o или || (логическое ИЛИ).
  - ! (логическое НЕ).

**5. Циклы в SHELL.**

- for — перебор элементов списка или диапазона.
- while — выполняется, пока условие истинно.
- until — выполняется, пока условие ложно.

**6. Массивы и модули в SHELL.**

- **Массивы:** объявляются как array=(val1 val2). Доступ к элементу: ${array[0]}.
- **Модули:** в Shell нет системы модулей как в Python, но можно подключать внешние файлы с функциями с помощью команды source или ..

**7. Чтение параметров командной строки.** Осуществляется через позиционные параметры: $0 — имя скрипта, $1, $2... — аргументы. $* и $@ — все параметры сразу, $# — их количество.

**8. Как различать ключи и параметры?**

- **Ключи (опции):** обычно начинаются с дефиса (-a, --help) и меняют поведение команды.
- **Параметры (аргументы):** данные, над которыми выполняется команда (имена файлов, пути, строки).

**9. Чтение данных из файлов.** Обычно реализуется через цикл while и команду read: while read line; do ... done < file.txt. Также можно использовать cat, awk или sed.

**10. Стандартные дескрипторы файлов.**

1. 0 (stdin) — стандартный ввод (клавиатура).
2. 1 (stdout) — стандартный вывод (экран).
3. 2 (stderr) — стандартный поток ошибок.

**11. Перенаправление вывода.**

- > — запись в файл (перезапись).
- >> — добавление в конец файла.
- 2> — перенаправление только ошибок.
- &> — перенаправление и вывода, и ошибок в один файл.

**12. Подавление вывода.** Осуществляется перенаправлением в «черную дыру» Linux — специальное устройство /dev/null. Пример: command > /dev/null 2>&1.

**13. Отправка сигналов скриптам.** Используется команда kill -SIGNAL PID. Популярные сигналы: SIGINT (Ctrl+C), SIGTERM (завершение), SIGKILL (принудительное завершение). Внутри скрипта сигналы можно перехватывать командой trap.

**14. Использование функций.** Функции объявляются так: name() { ... }. Вызываются по имени. Аргументы внутри функции доступны через те же $1, $2, но они не зависят от аргументов самого скрипта.

**15. Обработка текстов.** Для этого используются «три кита» Unix:

- grep — поиск по шаблону.
- sed — потоковое редактирование (замена, удаление).
- awk — мощная обработка колонок и отчетов.

**16. Отправка сообщений в терминал пользователя.**

- echo и printf — вывод в текущий терминал.
- write — сообщение конкретному пользователю.
- wall — сообщение всем пользователям системы.

**17. BASH и SHELL – синонимы?** Нет. **Shell** — это общее название командных интерпретаторов (стандарт POSIX). **Bash** — это конкретная, наиболее популярная реализация Shell (усовершенствованный вариант оригинального Sh). Все скрипты Bash являются Shell-скриптами, но не все Shell-скрипты запустятся в чистом Sh, если в них использованы "башизмы".

**18. PowerShell в ОС Windows: назначение и особенности.**

- **Назначение:** автоматизация администрирования Windows и облачных систем.
- **Особенности:** в отличие от Bash, который работает с текстом, PowerShell основан на платформе .NET и работает с **объектами**. Это позволяет извлекать свойства (например, дату создания файла) напрямую, без парсинга текстовых строк.