

Отчет по лабораторной работе №10

Операционные системы

Фадин В.В.

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Задача 1	6
2.2	Задача 2	7
2.3	Задача 3	10
2.4	Задача 4	11
3	Выводы	13
4	Ответы на контрольные вопросы	14

Список иллюстраций

2.1	Скрипт поиска	6
2.2	Запуск скрипта поиска	7
2.3	Создание программы на языке Си	8
2.4	Скрипт анализа кода	9
2.5	Запуск анализа кода	9
2.6	Скрипт создания и удаления файлов	10
2.7	Выполнение скрипта создания и удаления файлов	10
2.8	Скрипт архиватора	11
2.9	Модификация скрипта архиватора	11
2.10	Модификация скрипта архиватора	12

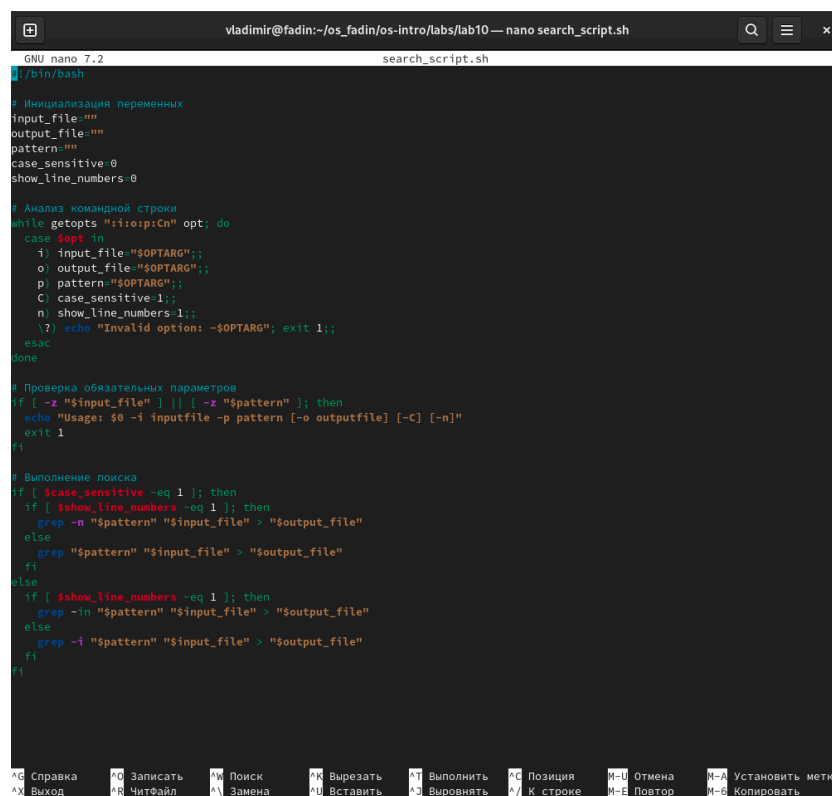
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Задача 1



```
GNU nano 7.2 search_script.sh
#!/bin/bash

# Инициализация переменных
input_file=""
output_file=""
pattern=""
case_sensitive=0
show_line_numbers=0

# Анализ командной строки
while getopts ":i:op:Cn" opt; do
  case $opt in
    i) input_file="$OPTARG";;
    o) output_file="$OPTARG";;
    p) pattern="$OPTARG";;
    C) case_sensitive=1;;
    n) show_line_numbers=1;;
    \?) echo "Invalid option: -$OPTARG"; exit 1;;
    *) ;;
  esac
done

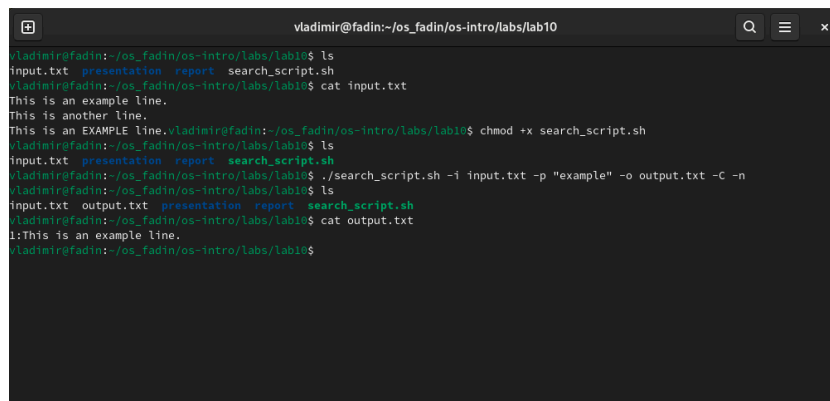
# Проверка обязательных параметров
if [ -z "$input_file" ] || [ -z "$pattern" ]; then
  echo "Usage: $0 -i inputfile -p pattern [-o outfile] [-C] [-n]"
  exit 1
fi

# Выполнение поиска
if [ $case_sensitive -eq 1 ]; then
  if [ $show_line_numbers -eq 1 ]; then
    grep -n "$pattern" "$input_file" > "$output_file"
  else
    grep "$pattern" "$input_file" > "$output_file"
  fi
else
  if [ $show_line_numbers -eq 1 ]; then
    grep -in "$pattern" "$input_file" > "$output_file"
  else
    grep -i "$pattern" "$input_file" > "$output_file"
  fi
fi
```

Рис. 2.1: Скрипт поиска

В этом скрипте мы используем `getopts` для анализа командной строки и извлечения значений параметров. Затем мы проверяем, были ли указаны обязательные параметры `-i` и `-p`. Если они указаны, мы выполняем поиск с помощью `grep`, учитывая параметры `-C` и `-n`. Результат поиска выводится в файл, указанный

параметром -o, если он был указан.

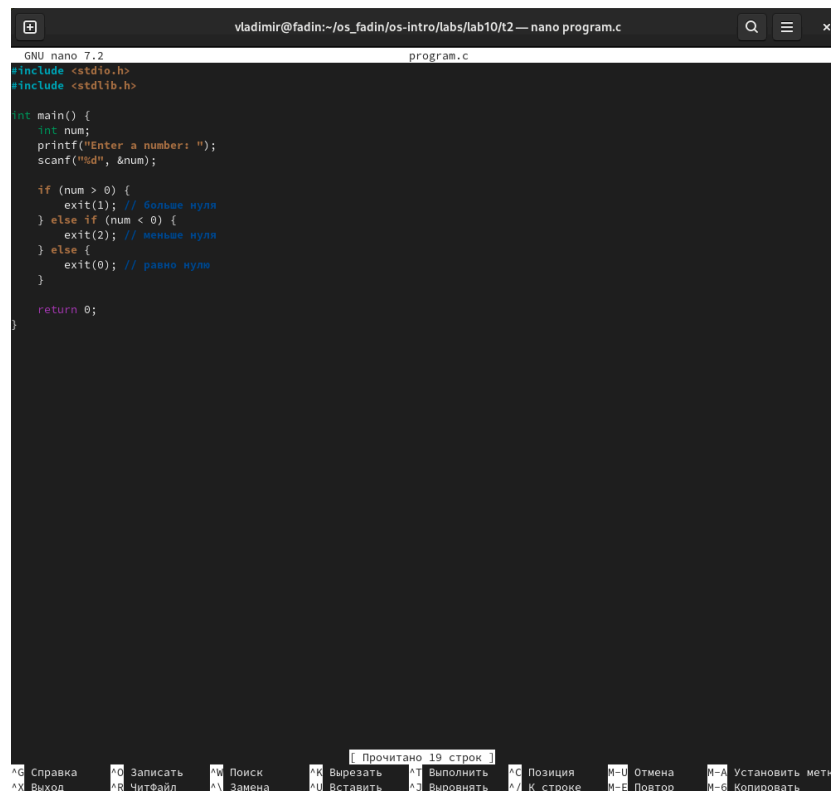


```
vladimir@fadin:~/os_fadin/os-intro/labs/lab10$ ls
input.txt  presentation  report  search_script.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10$ cat input.txt
This is an example line.
This is another line.
This is an EXAMPLE line.vladimir@fadin:~/os_fadin/os-intro/labs/lab10$ chmod +x search_script.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10$ ls
input.txt  presentation  report  search_script.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10$ ./search_script.sh -i input.txt -p "example" -o output.txt -C -n
input.txt  output.txt  presentation  report  search_script.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10$ cat output.txt
1:This is an example line.
vladimir@fadin:~/os_fadin/os-intro/labs/lab10$
```

Рис. 2.2: Запуск скрипта поиска

2.2 Задача 2

Вот пример программы на языке Си, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю:



```
GNU nano 7.2 program.c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num > 0) {
        exit(1); // больше нуля
    } else if (num < 0) {
        exit(2); // меньше нуля
    } else {
        exit(0); // равно нулю
    }

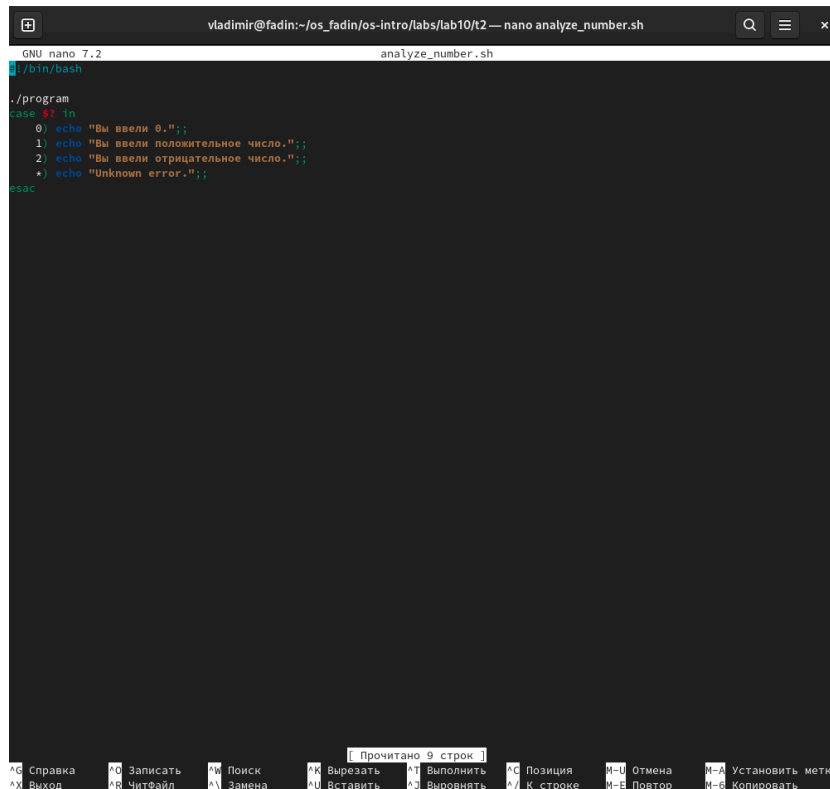
    return 0;
}
```

Рис. 2.3: Создание программы на языке Си

Далее компилируем программу:

```
gcc program.c -o program
```

Теперь создадим командный файл, который будет вызывать эту программу и анализировать код завершения:



```
GNU nano 7.2 analyze_number.sh
./bin/bash

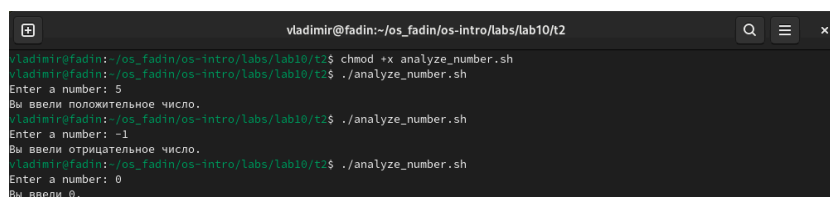
./program
case $? in
0) echo "Вы ввели 0.";;
1) echo "Вы ввели положительное число.";;
2) echo "Вы ввели отрицательное число.";;
*) echo "Unknown error.";;
esac
```

Рис. 2.4: Скрипт анализа кода

Сохраняем командный файл, например, как `analyze_number.sh`, и делаем его исполняемым:

```
chmod +x analyze_number.sh
```

Теперь мы можем запустить командный файл:



```
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t2$ chmod +x analyze_number.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t2$ ./analyze_number.sh
Enter a number: 5
Вы ввели положительное число.
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t2$ ./analyze_number.sh
Enter a number: -1
Вы ввели отрицательное число.
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t2$ ./analyze_number.sh
Enter a number: 0
Вы ввели 0.
```

Рис. 2.5: Запуск анализа кода

Программа будет запрашивать ввод числа, и после ввода она будет завершаться с кодом, соответствующим типу введенного числа. Командный файл будет анализировать код завершения и выводить соответствующее сообщение.

2.3 Задача 3

Вот пример командного файла, который создает указанное число файлов и может удалять их:



```
GNU nano 7.2 create_files.sh
#!/bin/bash

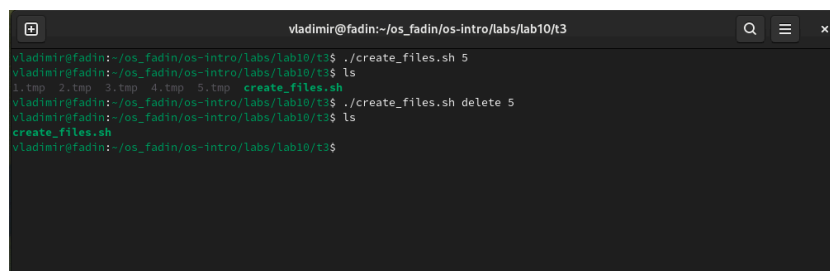
# Проверка аргументов
if [ $# -eq 1 ]; then
    # Создание файлов
    for i in $(seq 1 $1); do
        touch "$i.tmp"
    done
elif [ $# -eq 2 ] && [ "$1" = "delete" ]; then
    # Удаление файлов
    for i in $(seq 1 $2); do
        rm -f "$i.tmp"
    done
else
    echo "Usage: $0 <number_of_files> or $0 delete <number_of_files>"
    exit 1
fi
```

Рис. 2.6: Скрипт создания и удаления файлов

Командный файл принимает один аргумент - количество файлов, которые нужно создать. Он использует цикл `for` с командой `seq` для создания файлов с именами от `1.tmp` до `<number_of_files>.tmp`.

Если аргументом является строка `"delete"`, то командный файл удаляет все созданные файлы, используя цикл `for` с командой `rm -f`.

Пример использования:



```
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t3$ ./create_files.sh 5
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t3$ ls
1.tmp 2.tmp 3.tmp 4.tmp 5.tmp create_files.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t3$ ./create_files.sh delete 5
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t3$ ls
create_files.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t3$
```

Рис. 2.7: Выполнение скрипта создания и удаления файлов

В первом примере командный файл создает 5 файлов с именами `1.tmp`, `2.tmp`, ..., `5.tmp`. Во втором примере он удаляет эти файлы, если они существуют.

2.4 Задача 4

Вот пример командного файла, который запаковывает в архив все файлы в указанной директории:



```
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4 — nano pack_files.sh
GNU nano 7.2 pack_files.sh
#!/bin/bash

# Проверка аргументов
if [ $# -ne 1 ]; then
    echo "Usage: $0 <directory>"
    exit 1
fi

# Установка директории
directory=$1

# Создание имени архива
archive_name="${directory##*/}.tar.gz"

# Запаковка файлов в архив
tar -czf "$archive_name" "$directory"
```

Рис. 2.8: Скрипт архиватора

Командный файл принимает один аргумент - путь к директории, которую нужно запаковать.

Чтобы модифицировать его, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад, мы можем использовать команду `find`:



```
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4 — nano pack_files.sh
GNU nano 7.2 pack_files.sh
#!/bin/bash

# Проверка аргументов
if [ $# -ne 1 ]; then
    echo "Usage: $0 <directory>"
    exit 1
fi

# Установка директории
directory=$1

# Создание имени архива
archive_name="${directory##*/}.tar.gz"

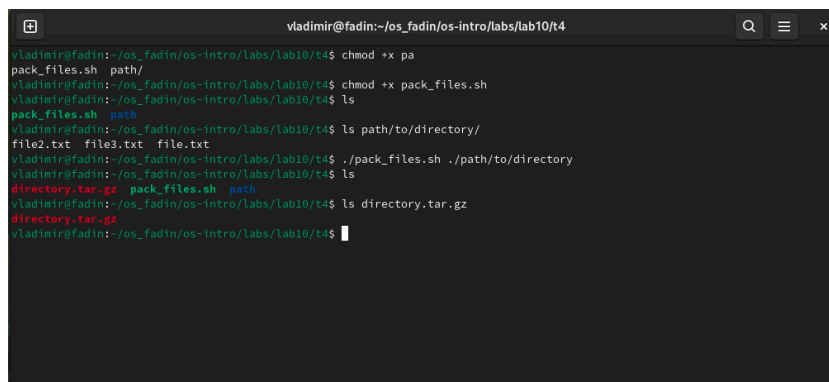
# Запаковка файлов в архив, измененных менее недели тому назад
find "$directory" -type f -mtime -7 -exec tar -rf "$archive_name" {} \;
```

Рис. 2.9: Модификация скрипта архиватора

В этом модифицированном командном файле мы используем команду `find` для поиска файлов в указанной директории, которые были изменены менее

недели тому назад (`-mtime -7`). Затем мы используем опцию `-exes` для запуска команды `tar` с каждым найденным файлом, добавляя его в архив.

Пример использования:



```
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$ chmod +x pa
pack_files.sh path/
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$ chmod +x pack_files.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$ ls
pack_files.sh path
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$ ls path/to/directory/
file2.txt file3.txt file.txt
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$ ./pack_files.sh ./path/to/directory
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$ ls
directory.tar.gz pack_files.sh path
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$ ls directory.tar.gz
directory.tar.gz
vladimir@fadin:~/os_fadin/os-intro/labs/lab10/t4$
```

Рис. 2.10: Модификация скрипта архиватора

Командный файл создаст архив с именем `directory.tar.gz` в текущей директории, содержащий все файлы в указанной директории, которые были изменены менее недели тому назад.

3 Выводы

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

4 Ответы на контрольные вопросы

1. Команда `getopts` предназначена для парсинга параметров командной строки в shell-скриптах. Она позволяет извлекать и обрабатывать опции, передаваемые в скрипт, и их аргументы.
2. Метасимволы (*, ?, [,], (,), {, }, ~, .) используются в Linux для генерации имён файлов, известной как `globbing`. Они позволяют указать шаблоны для поиска файлов, что может быть полезно при выполнении различных операций с файлами, таких как поиск, удаление, копирование и т.д.
3. Операторы управления действиями в Linux включают в себя:
 - `&&` (логическое И) - выполняет вторую команду только если первая команда выполнена успешно.
 - `||` (логическое ИЛИ) - выполняет вторую команду только если первая команда не выполнена успешно.
 - `;` (точка с запятой) - разделяет команды, которые выполняются последовательно.
 - `&` (амперсанд) - запускает команду в фоне.
 - `|` (вертикальная черта) - перенаправляет вывод одной команды на вход другой.
4. Операторы, используемые для прерывания цикла, включают в себя:
 - `break` - прерывает выполнение цикла и продолжает выполнение скрипта после цикла.

- `continue` - прерывает текущую итерацию цикла и продолжает выполнение с следующей итерации.
5. Команды `false` и `true` используются для возвращения определенного статуса выполнения. `true` всегда возвращает 0 (успешное выполнение), а `false` всегда возвращает 1 (неуспешное выполнение). Они могут быть полезны в скриптах для управления потоком выполнения в зависимости от результатов предыдущих операций.
 6. Строка `if test -f man$s/$i.$s` встреченная в командном файле, проверяет, существует ли файл с именем `$i.$s` в директории `man$s`. Если файл существует, то условие `if` будет истинным, иначе - ложным.
 7. Конструкции `while` и `until` используются для создания циклов в shell-скриптах.
 - `while` - цикл будет продолжаться, пока условие является истинным.
 - `until` - цикл будет продолжаться, пока условие является ложным.