4. Изучение сценариев bash

Задание 1: Вывод путей из переменной PATH построчно с использованием цикла `for in`

Переменная `PATH` содержит список директорий, в которых система ищет исполняемые файлы. Эти директории разделены символом `:`. Для того чтобы вывести каждую директорию на отдельной строке, можно использовать цикл `for in`, изменив разделитель полей (`IFS`) на `:`.

Скрипт:

```
#!/bin/bash

# Устанавливаем разделитель полей на двоеточие
IFS=':'

# Цикл перебирает каждую директорию в PATH
for path in $PATH
do
    echo "$path"
done
```

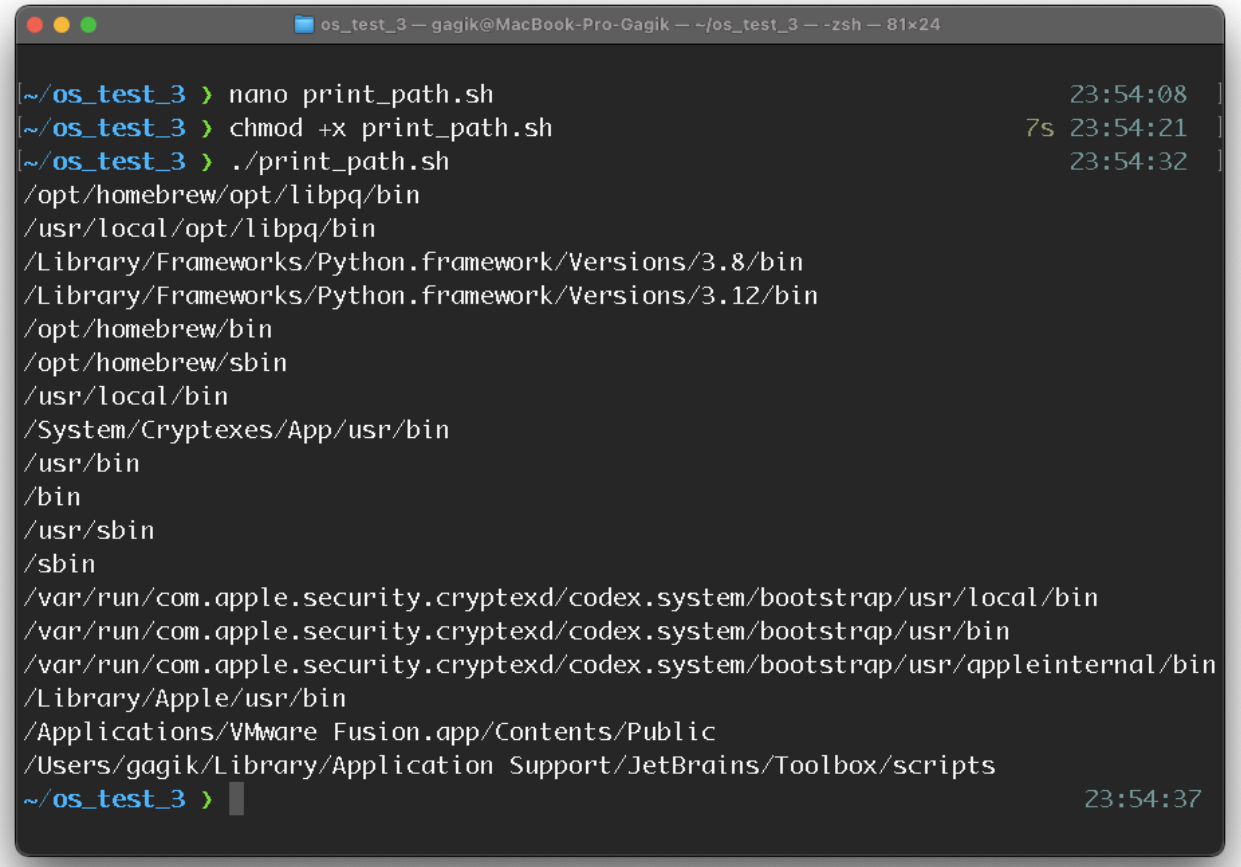
Объяснение:

1. **Шебанг:** `#!/bin/bash` указывает системе, что скрипт должен выполняться с помощью Bash.
2. **Установка IFS:** Переменная `IFS` (Internal Field Separator) задаёт разделитель полей для циклов и других операций.

Устанавливая `IFS=':'`, мы указываем, что разделителем будет двоеточие, что соответствует формату переменной `PATH`.

3. **Цикл `for`:** Перебирает каждую директорию в `PATH`, разделённую `:`.
4. **Вывод:** Команда `echo "$path"` выводит текущую директорию на экран.

Запуск скрипта:



Задание 2: Вывод имен файлов из каталога и его подкаталогов с указанием уровня вложенности

Для выполнения этого задания необходимо рекурсивно обходить директории и выводить имена файлов с указанием уровня вложенности. Уровень вложенности будет отображаться числом в начале строки в формате `n)`, где `n` — уровень начиная с 1.

Скрипт:

```
#!/bin/bash

# Функция для рекурсивного обхода директорий
list_files() {
    local dir="$1"    # Текущая директория
    local level="$2"  # Текущий уровень вложенности

    # Перебираем все элементы в директории
    for item in "$dir"/*
    do
        # Проверяем, существует ли элемент (избегаем
ошибок при отсутствии файлов)
        if [ -e "$item" ]; then
            if [ -d "$item" ]; then
                # Если элемент — директория, выводим с
уровнем и рекурсивно вызываем функцию
                echo "${level}) $(basename "$item")/"
                list_files "$item" $((level + 1))
            elif [ -f "$item" ]; then
                # Если элемент — файл, выводим с уровнем
                echo "${level}) $(basename "$item")"
            fi
        fi
    done
}

# Проверка, передан ли каталог в качестве аргумента
```

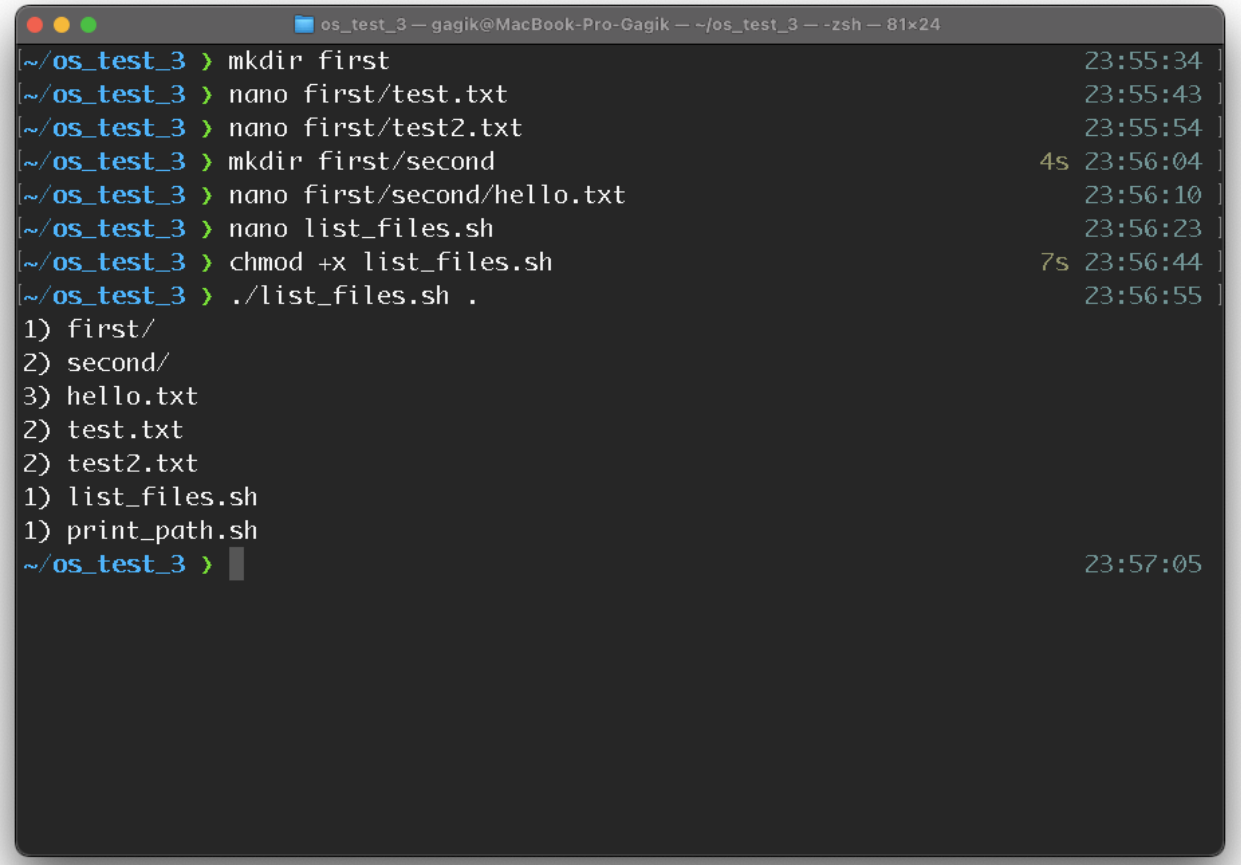
```
if [ -z "$1" ]; then
    echo "Использование: $0 <каталог>"
    exit 1
fi

# Проверка, существует ли переданный путь и является ли он каталогом
if [ ! -d "$1" ]; then
    echo "Ошибка: '$1' не является каталогом или не существует."
    exit 1
fi

# Запуск функции с начальным уровнем 1
list_files "$1" 1
```

Объяснение:

1. **Шебанг:** `#!/bin/bash` указывает, что скрипт будет выполняться с помощью Bash.
2. **Функция `list_files`:**
    - **Параметры:**
        - `$1` — путь к текущей директории.
        - `$2` — текущий уровень вложенности.
    - **Цикл `for`:** Перебирает все элементы в текущей директории.
    - **Проверка существования элемента:** `if [ -e "$item" ]` избегает ошибок при отсутствии файлов или директорий.
    - **Проверка типа элемента:**
        - Если элемент — директория (`-d`), выводит её имя с уровнем и рекурсивно вызывает `list_files` для этой директории, увеличивая уровень на 1.

        - Если элемент — файл (`-f`), выводит его имя с текущим уровнем.
3. **Проверка аргументов:**
    - Скрипт ожидает, что в качестве первого аргумента будет указан путь к каталогу.
    - Если аргумент не передан или путь не является директорией, скрипт выводит сообщение об ошибке и завершает работу.
4. **Запуск функции:**
    - Вызывается функция `list_files` с переданным каталогом и начальным уровнем `1` .

Пример использования:


os_test_3 — gagik@MacBook-Pro-Gagik — ~/os_test_3 — -zsh — 81×24
~/os_test_3 ❯ mkdir first 23:55:34
~/os_test_3 ❯ nano first/test.txt 23:55:43
~/os_test_3 ❯ nano first/test2.txt 23:55:54
~/os_test_3 ❯ mkdir first/second 4s 23:56:04
~/os_test_3 ❯ nano first/second/hello.txt 23:56:10
~/os_test_3 ❯ nano list_files.sh 23:56:23
~/os_test_3 ❯ chmod +x list_files.sh 7s 23:56:44
~/os_test_3 ❯ ./list_files.sh . 23:56:55
1) first/
2) second/
3) hello.txt
2) test.txt
2) test2.txt
1) list_files.sh
1) print_path.sh
~/os_test_3 ❯ 23:57:05

5. Изучение сценариев bash

Задание 1: Скрипт для вывода содержимого текстового файла с сохранением форматирования и отображением символов `\`

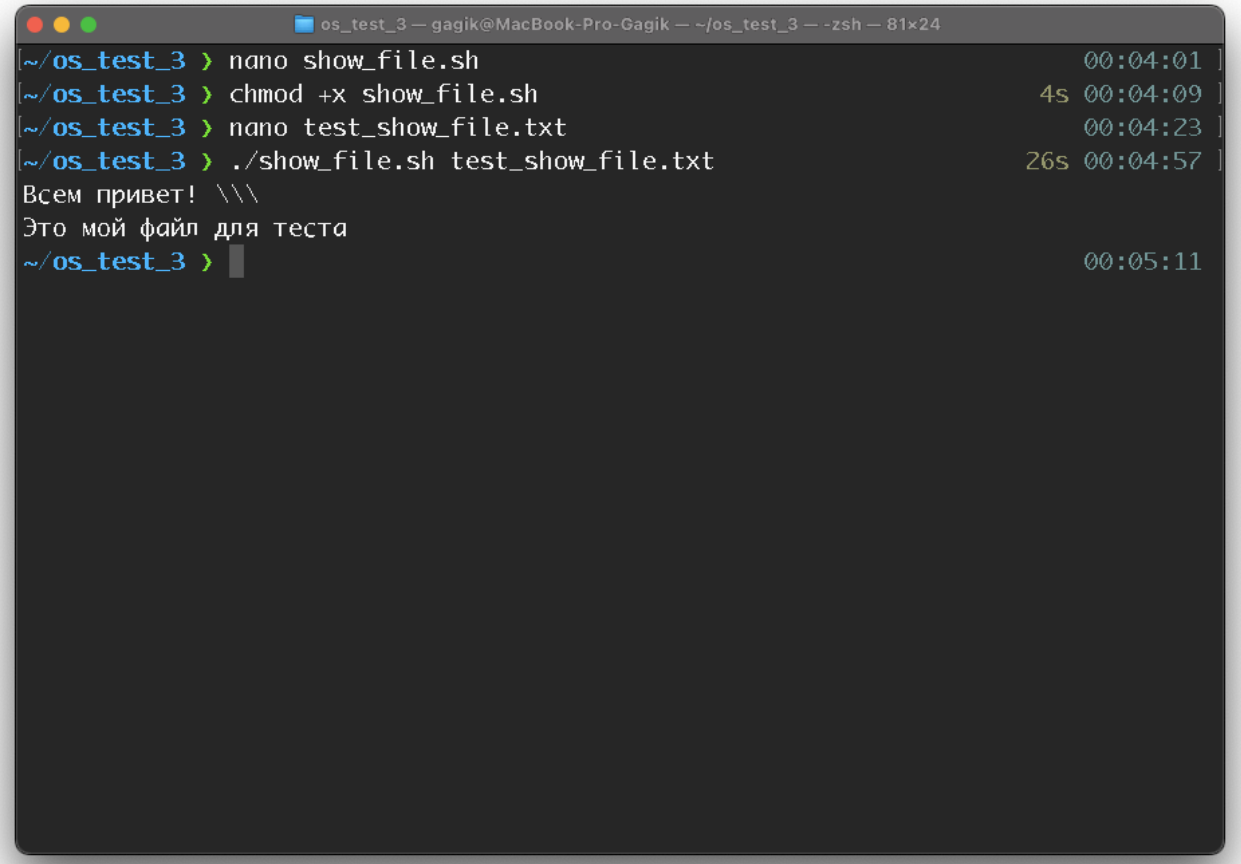
Описание:
Этот скрипт принимает имя текстового файла в качестве аргумента, открывает его для чтения с помощью `exec`, затем читает файл построчно с помощью `read` и выводит каждую строку на экран. Чтобы символы `\` отображались корректно, используется команда `printf` вместо `echo`.

Скрипт:

```
#!/bin/bash

# Проверка, передан ли аргумент
if [ $# -ne 1 ]; then
    echo "Использование: $0 <имя_файла>"
    exit 1
fi

filename="$1"

# Проверка, существует ли файл и доступен ли для чтения
if [ ! -e "$filename" ]; then
    echo "Ошибка: Файл '$filename' не существует."
    exit 1
```

```
elif [ ! -r "$filename" ]; then
    echo "Ошибка: Нет доступа для чтения файла
'$filename'."
    exit 1
fi

# Открытие файла для чтения с дескриптором 3
exec 3< "$filename"

# Чтение файла построчно
while IFS= read -r line <&3
do
    # Используем printf для сохранения форматирования и
отображения символов '\'
    printf '%s\n' "$line"
done

# Закрытие дескриптора файла
exec 3<&-
```

Использование:



Задание 2: Скрипт для создания файла с обработкой ошибок при существовании файла

Описание:
Этот скрипт принимает имя файла в качестве первого аргумента и пытается создать его. Если файл уже существует, скрипт выводит сообщение об ошибке. Для обработки ошибок используется сигнал `ERR` с помощью команды `trap`.

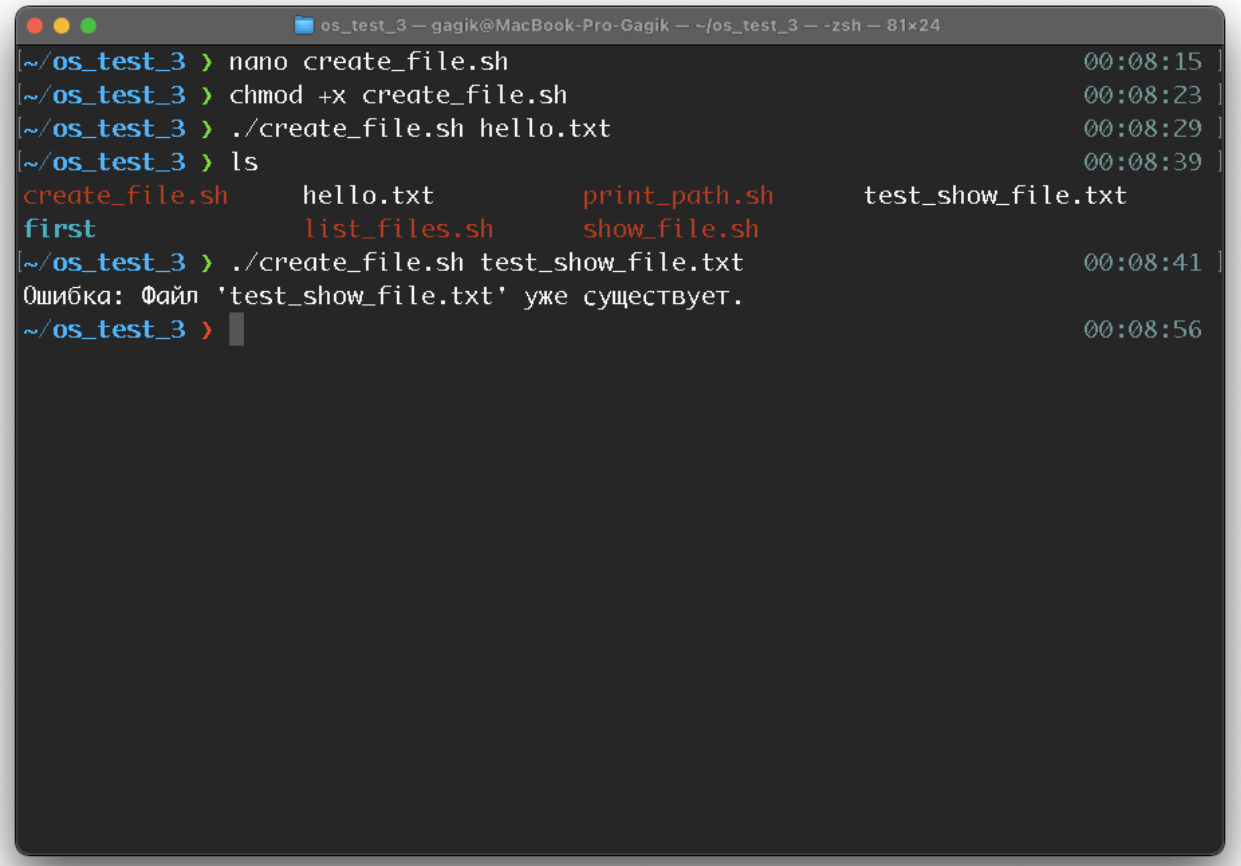
Пояснение:

1. **Функция `error_handler`:** Эта функция выводит сообщение об ошибке и завершает скрипт с кодом 1.
2. **Команда `trap`:** Устанавливает ловушку на сигнал `ERR`, которая вызовет `error_handler` при возникновении ошибки в любом месте скрипта.
3. **Проверка аргументов:** Убедимся, что скрипту передан ровно один аргумент.
4. **Создание файла:** Команда `touch` пытается создать файл. Если файл уже существует, `touch` не вызывает ошибку, поэтому для корректной обработки можно использовать проверку с `test -e`. Однако, чтобы соответствовать заданию по использованию сигнала `ERR`, можно использовать команду, которая возвращает ошибку, если файл существует. Например, `cp /dev/null "$filename"` не сработает, поэтому лучше использовать команду `> "$filename"` с проверкой.

```
#!/bin/bash

# Функция обработки ошибок
error_handler() {
    echo "Ошибка: Файл '$filename' уже существует."
    exit 1
}

# Установка ловушки на сигнал ERR
trap 'error_handler' ERR

# Проверка, передан ли аргумент
if [ $# -ne 1 ]; then
    echo "Использование: $0 <имя_файла>"
    exit 1
fi
```

```bash
filename="$1"

# Проверка, существует ли файл. Если да, вызываем ошибку.
[ -e "$filename" ] && false

# Создание файла
exec 1> "$filename"
echo "Файл '$filename' создан."
```

Использование:



6. Изучение процессов POSIX

Описание программы `prntxt`

Программа `prntxt` принимает следующие параметры командной строки:

- `-n` или `--number` <число повторов> (обязательный параметр)
- `-t` или `--timeout` [<таймаут>] (опциональный параметр, по умолчанию 1 секунда)
- `--` для отделения опций от текста
- <TEXT> — произвольный текст для вывода

При отсутствии таймаута программа выводит текст без задержки. В случае отсутствия обязательных параметров или некорректного ввода, программа выводит сообщение об ошибке и подсказку по использованию.

Исходный код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include <string.h>

// Функция для вывода подсказки
void print_usage() {
    fprintf(stderr, "Usage: prntxt -n|--number <N> [-t|--
timeout [T]] -- <TEXT>\n");
}
```

```c
int main(int argc, char *argv[]) {
    int opt;
    int option_index = 0;
    long number = 0;
    int timeout_present = 0;
    long timeout = 1; // Значение по умолчанию
    char *text = NULL;

    // Определение длинных опций
    static struct option long_options[] = {
        {"number",   required_argument, 0, 'n'},
        {"timeout",  optional_argument, 0, 't'},
        {0,          0,                 0,  0 }
    };

    // Парсинг опций
    while ((opt = getopt_long(argc, argv, "n:t::",
long_options, &option_index)) != -1) {
        switch (opt) {
            case 'n':
                number = strtol(optarg, NULL, 10);
                if (number <= 0) {
                    fprintf(stderr, "Error: <число
повторов> должно быть положительным.\n");
                    print_usage();
                    return EXIT_FAILURE;
                }
                break;
            case 't':
                timeout_present = 1;
                if (optarg) {
                    timeout = strtol(optarg, NULL, 10);
                    if (timeout < 0) {
                        fprintf(stderr, "Error: <таймаут>
```

```
должно быть неотрицательным.\n");
                        print_usage();
                        return EXIT_FAILURE;
                    }
                }
                // Обработка случая, когда таймаут передан
отдельно (например, -t 1)
                else if (optind < argc && argv[optind][0]
!= '-') {
                    timeout = strtol(argv[optind], NULL,
10);
                    if (timeout < 0) {
                        fprintf(stderr, "Error: <таймаут>
должно быть неотрицательным.\n");
                        print_usage();
                        return EXIT_FAILURE;
                    }
                    optind++;
                }
                else {
                    timeout = 1; // Значение по умолчанию,
если таймаут указан без параметра
                }
                break;
            case '?':
            default:
                print_usage();
                return EXIT_FAILURE;
        }
    }

    // Проверка обязательного параметра -n/--number
    if (number == 0) {
        fprintf(stderr, "Error: Не указано число повторов
(-n|--number).\n");
```

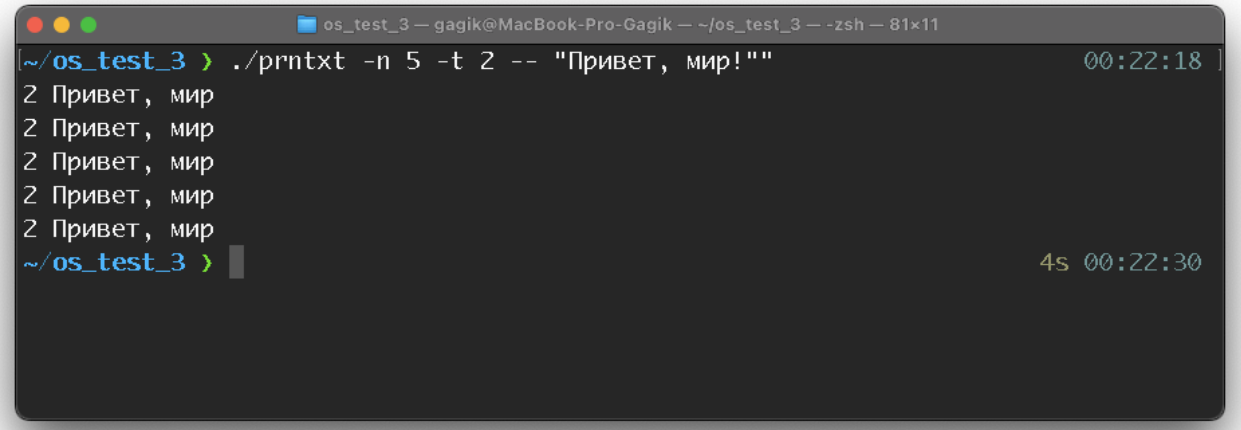
```
        print_usage();
        return EXIT_FAILURE;
    }

    // Проверка наличия '--' и текста после него
    if (optind < argc && strcmp(argv[optind], "--") == 0)
{
        optind++; // Пропустить '--'
    }

    if (optind >= argc) {
        fprintf(stderr, "Error: Не указан текст для
вывода.\n");
        print_usage();
        return EXIT_FAILURE;
    }

    // Объединение оставшихся аргументов в один текст
    size_t text_length = 0;
    for (int i = optind; i < argc; i++) {
        text_length += strlen(argv[i]) + 1;
    }

    text = malloc(text_length);
    if (!text) {
        perror("malloc");
        return EXIT_FAILURE;
    }
    text[0] = '\0';

    for (int i = optind; i < argc; i++) {
        strcat(text, argv[i]);
        if (i < argc - 1) {
            strcat(text, " ");
        }
```

```c
    }

    // Вывод текста
    for (long i = 0; i < number; i++) {
        printf("%s\n", text);
        fflush(stdout);
        if (timeout_present && i < number - 1) {
            sleep(timeout);
        }
    }

    free(text);
    return EXIT_SUCCESS;
}
```

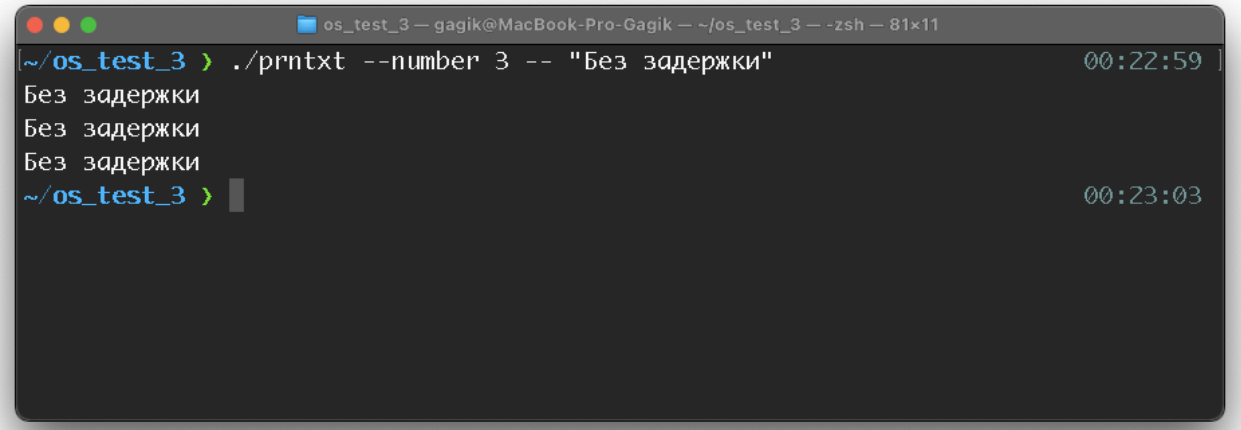
Использование

1. Компиляция программы

2. Запуск программы с таймаутом



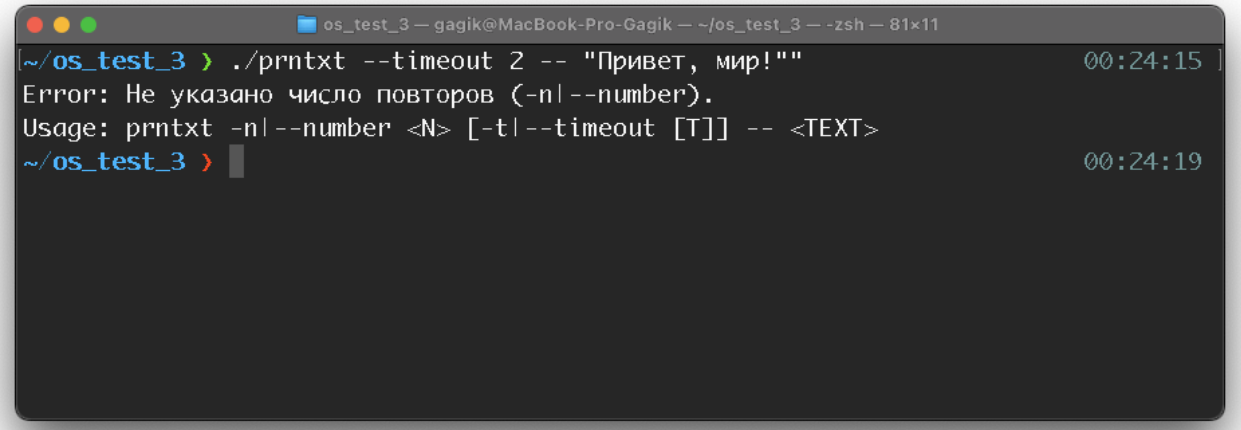
3. Запуск программы без таймаута



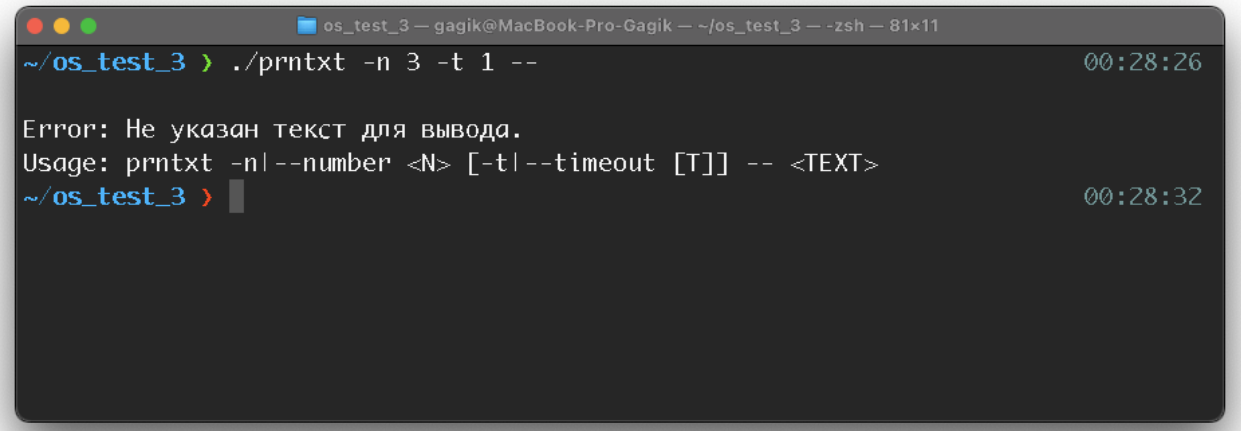
Обработка ошибок

Программа `prntxt` включает обработку различных ошибок, связанных с некорректным вводом параметров.

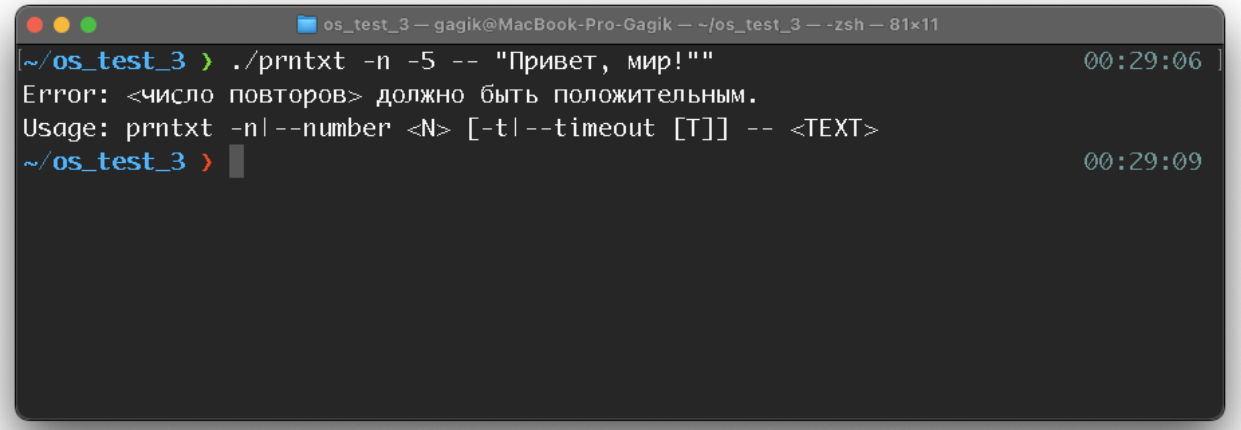
1. Отсутствие обязательной опции `-n` или `--number`



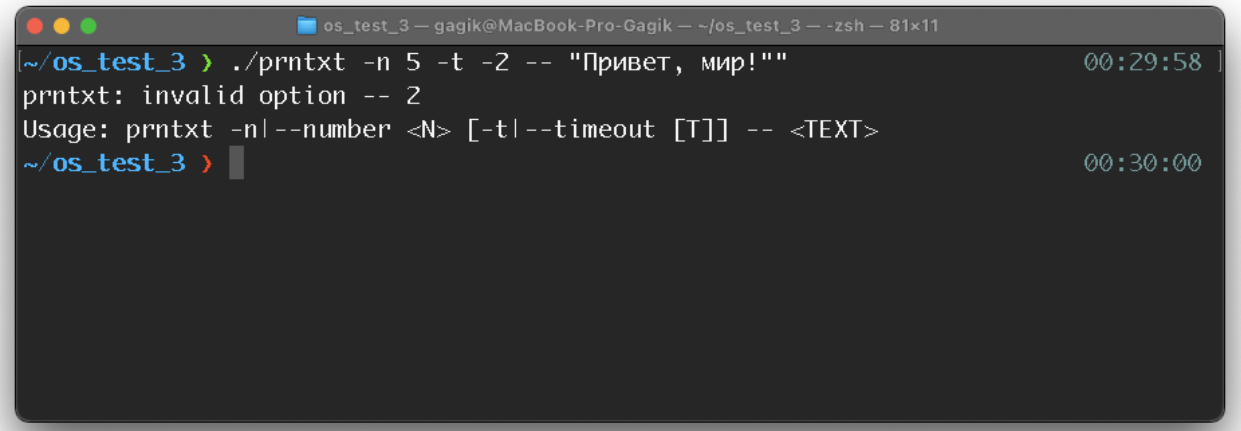
2. Отсутствие текста для вывода после `--`



3. Некорректное значение для числа повторов (`-n`)



4. Некорректное значение для таймаута (`-t`)



Вывод

В ходе выполнения лабораторной работы была успешно разработана программа `prntxt` на языке C для операционной системы Linux. Программа позволяет выводить заданный текст определённое количество раз с возможностью установки таймаута между выводами. Реализована поддержка как коротких, так и длинных опций командной строки, что обеспечивает гибкость использования утилиты. Также была внедрена система обработки

ошибок, которая повышает надёжность и удобство использования программы, предоставляя пользователю понятные сообщения и подсказки при некорректных вводах.