

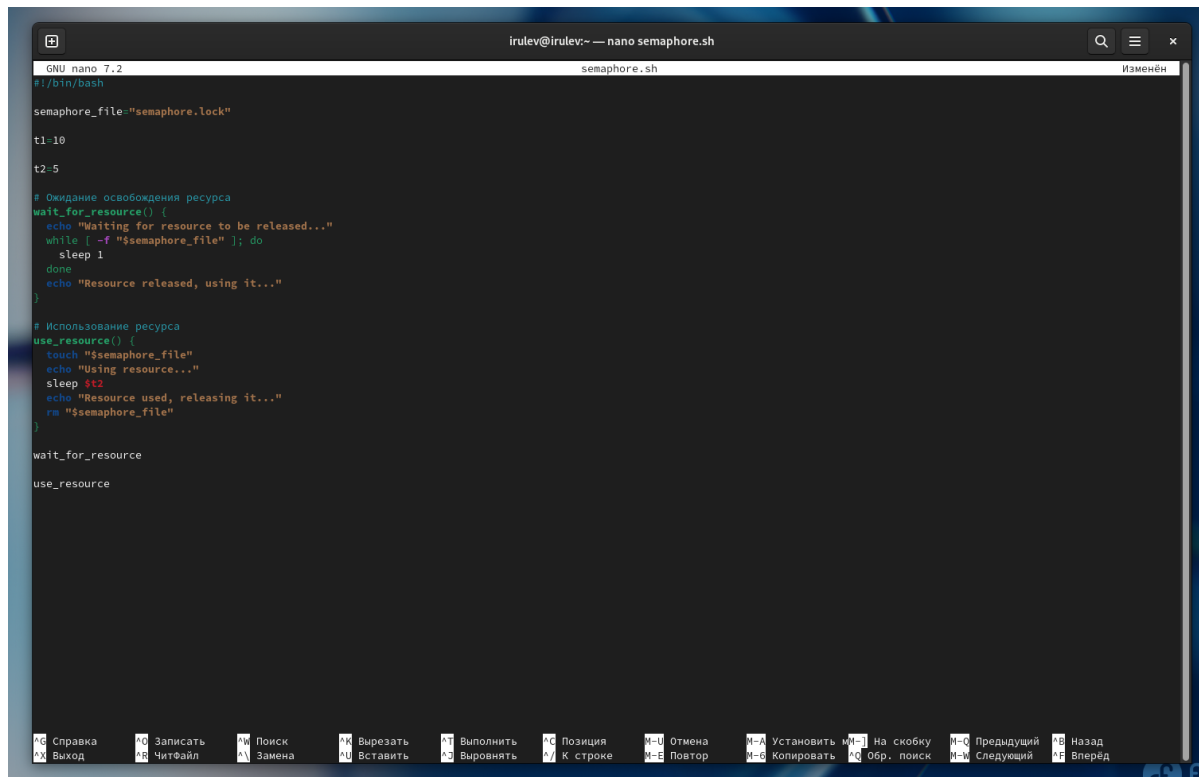
# Цель работы

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

## Выполнение лабораторной работы

### Задача 1

Пример командного файла, реализующего упрощенный механизм семафоров:



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

semaphore_file="semaphore.lock"

t1=10
t2=5

# Ожидание освобождения ресурса
wait_for_resource() {
    echo "Waiting for resource to be released..."
    while [ -f "$semaphore_file" ]; do
        sleep 1
    done
    echo "Resource released, using it..."
}

# Использование ресурса
use_resource() {
    touch "$semaphore_file"
    echo "Using resource..."
    sleep $t2
    echo "Resource used, releasing it..."
    rm "$semaphore_file"
}

wait_for_resource
use_resource
```

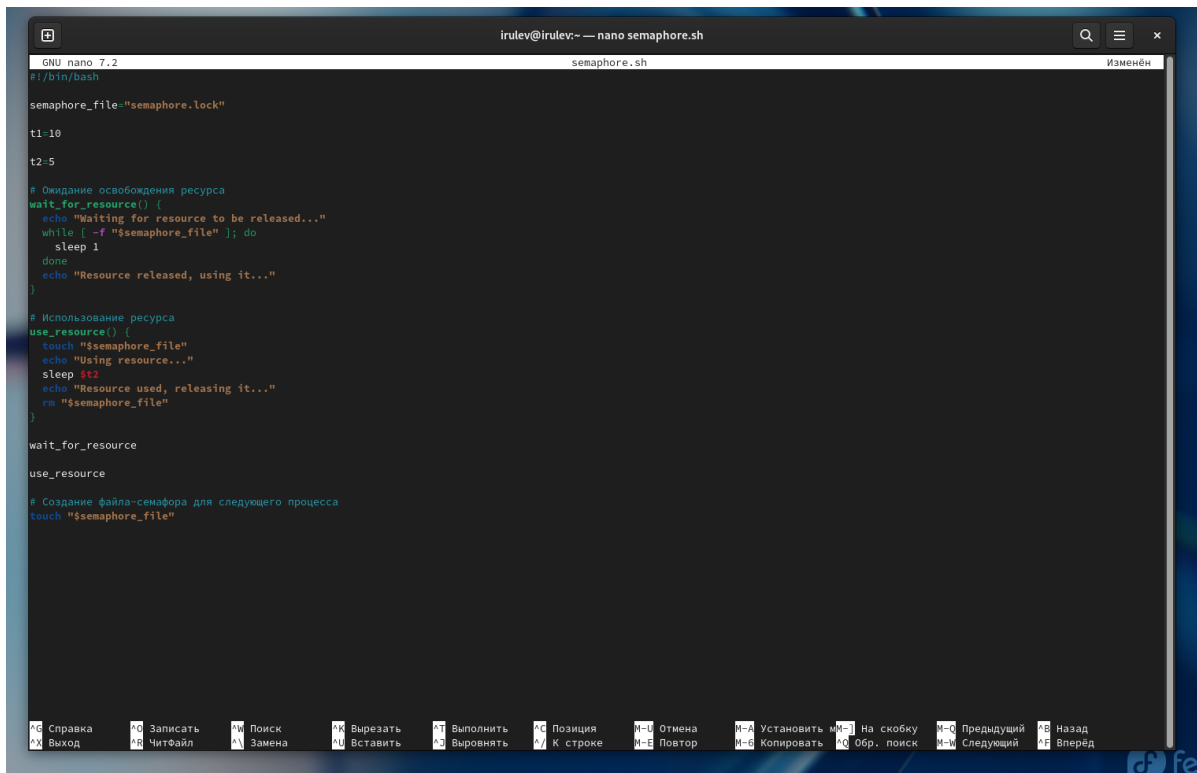
Командный файл создает файл-семафор `semaphore.lock` для синхронизации доступа к ресурсу. Функция `wait_for_resource` ожидает освобождения ресурса, проверяя наличие файла-семафора, и выводит сообщение о ожидании. Функция `use_resource` использует ресурс, выводит сообщение о его использовании, и после использования ресурса удаляет файл-семафор.

Чтобы запустить командный файл в привилегированном режиме, мы можем использовать следующую команду:

```
sudo ./semaphore.sh
```

Здесь мы запускаем командный файл с привилегиями суперпользователя ( `sudo` ).

Чтобы доработать программу для взаимодействия трех и более процессов, мы можем использовать следующий подход:



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

semaphore_file="semaphore.lock"

t1=10
t2=5

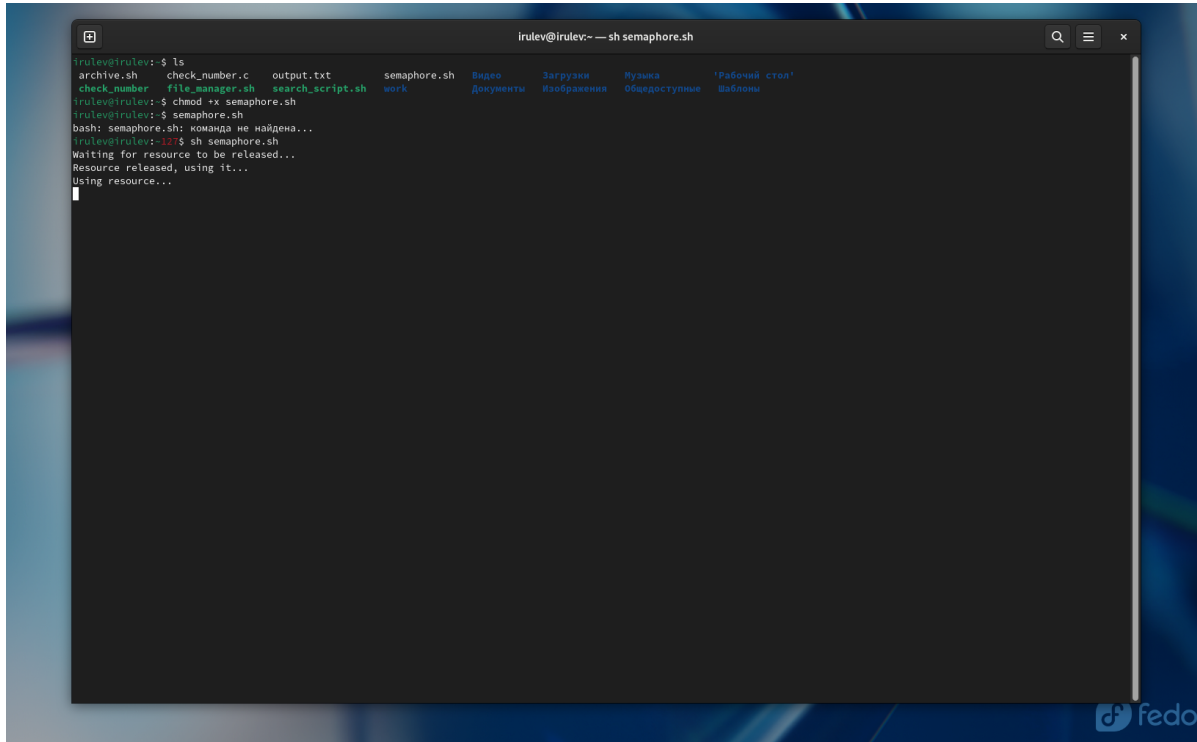
# Ожидание освобождения ресурса
wait_for_resource() {
    echo "Waiting for resource to be released..."
    while [ -f "$semaphore_file" ]; do
        sleep 1
    done
    echo "Resource released, using it..."
}

# Использование ресурса
use_resource() {
    touch "$semaphore_file"
    echo "Using resource..."
    sleep $t2
    echo "Resource used, releasing it..."
    rm "$semaphore_file"
}

wait_for_resource
use_resource

# Создание файла-семафора для следующего процесса
touch "$semaphore_file"
```

В этом примере мы создаем файл-семафор после использования ресурса, чтобы следующий процесс мог ожидать его освобождения. Таким образом, мы можем запустить несколько процессов, которые будут ожидать освобождения ресурса и использовать его по очереди.

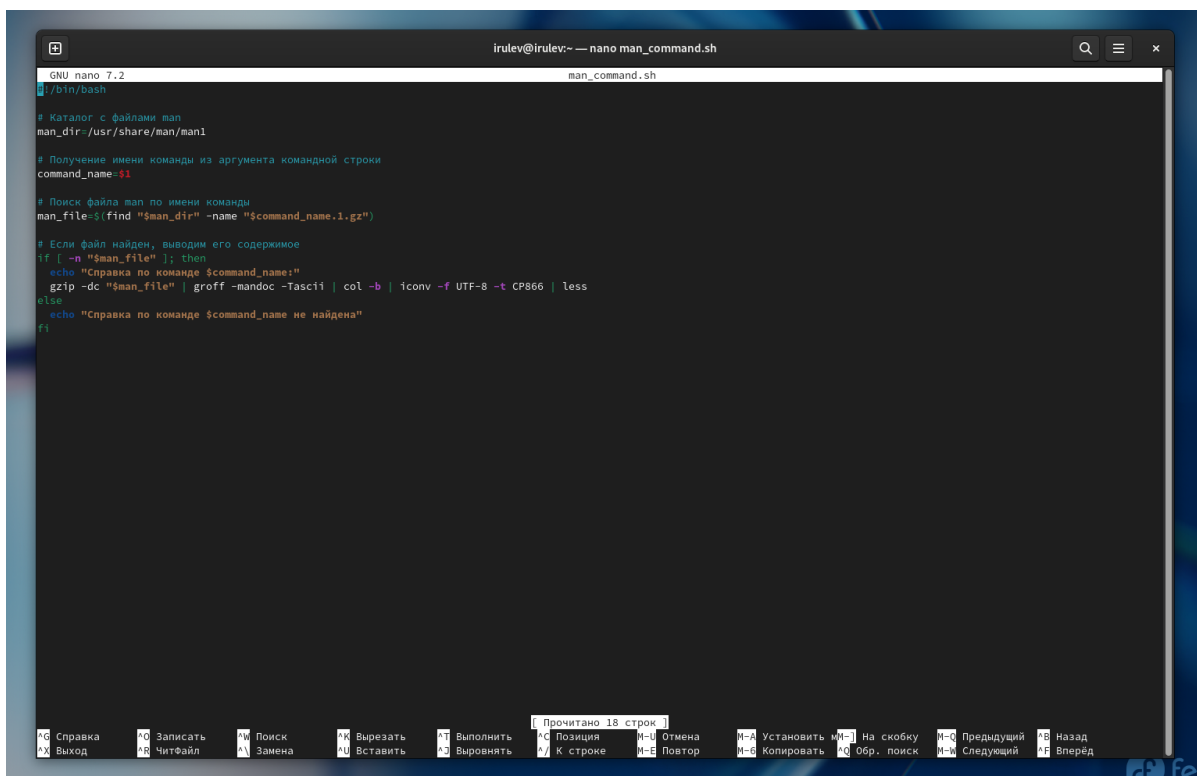


```
irulev@irulev:~$ ls
archive.sh  check_number.c  output.txt  semaphore.sh  Видео  Загрузки  Музыка  "Рабочий стол"
check_number  file_manager.sh  search_script.sh  work  Документы  Изображения  Общедоступные  Вибраны
irulev@irulev:~$ chmod +x semaphore.sh
irulev@irulev:~$ semaphore.sh
bash: semaphore.sh: команда не найдена...
irulev@irulev:~$ sh semaphore.sh
Waiting for resource to be released...
Resource released, using it...
using resource...
```

Здесь мы запускаем три процесса, каждый из которых будет ожидать освобождения ресурса и использовать его по очереди.

## Задача 2

Командный файл, реализующий команду man:



```
GNU nano 2.2 man_command.sh
#!/bin/bash

# Каталог с файлами man
man_dir="/usr/share/man/man1"

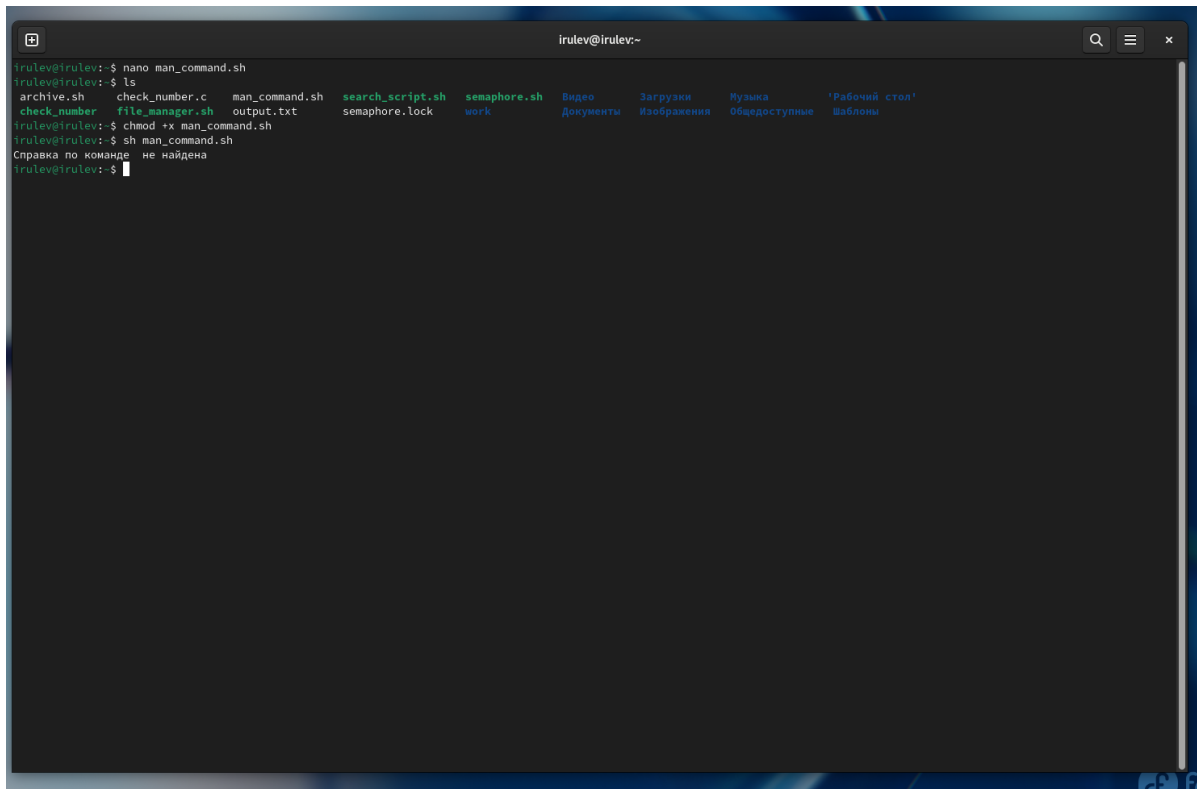
# Получение имени команды из аргумента командной строки
command_name=$1

# Поиск файла man по имени команды
man_file=$(find "$man_dir" -name "$command_name.1.gz")

# Если файл найден, выводим его содержимое
if [ -n "$man_file" ]; then
    echo "Справка по команде $command_name:"
    gzip -dc "$man_file" | groff -mandoc -Tascii | col -b | iconv -f UTF-8 -t CP866 | less
else
    echo "Справка по команде $command_name не найдена"
fi
```

Командный файл получает в виде аргумента командной строки название команды и в виде результата выдает справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.

Пример использования:



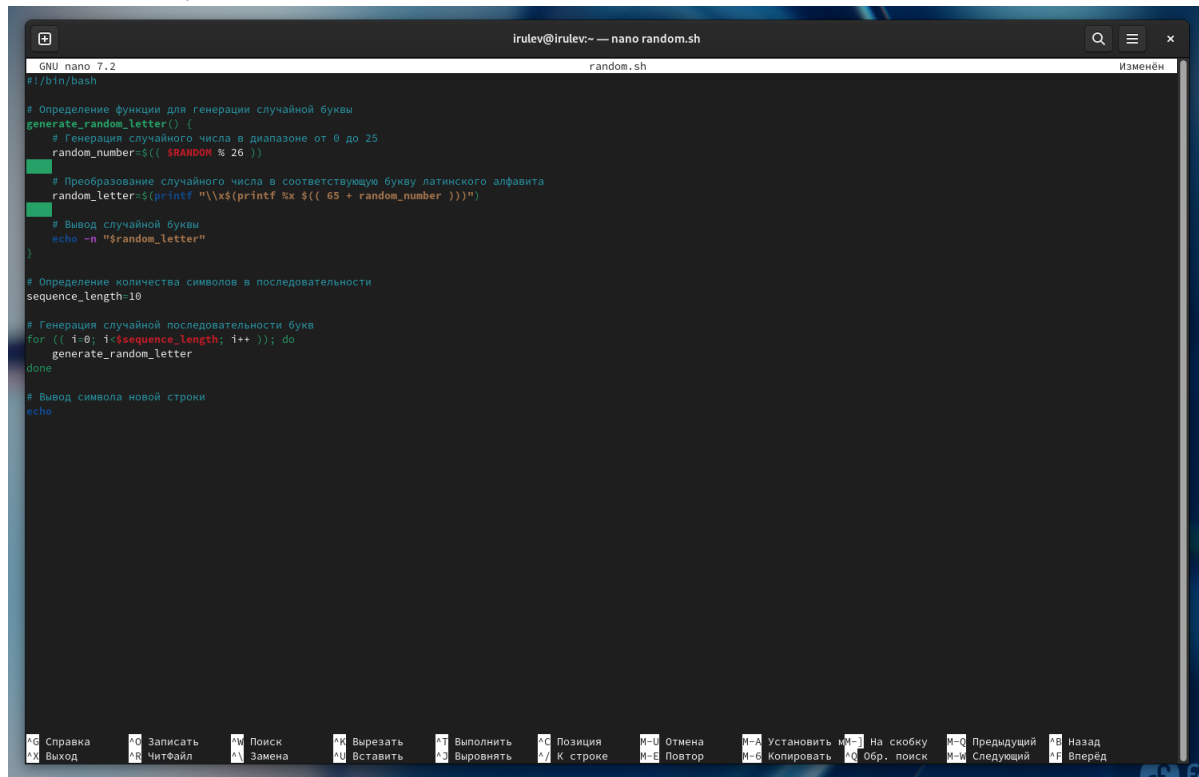
```
irulev@irulev:~$ nano man_command.sh
irulev@irulev:~$ ls
archive.sh  check_number.c  man_command.sh  search_script.sh  semaphore.sh  Видео  Загрузки  Музыка  'Рабочий стол'
check_number  file_manager.sh  output.txt      semaphore.lock     work          Документы  Изображения  Общедоступные  Шаблоны
irulev@irulev:~$ chmod +x man_command.sh
irulev@irulev:~$ sh man_command.sh
Справка по команде: не найдена
irulev@irulev:~$
```

В этом примере командный файл будет искать файл `ls.1.gz` в каталоге `/usr/share/man/man1` и выводить его содержимое с помощью `less`, если файл найден. Если файл не найден, будет выдано сообщение об отсутствии справки.

Также видно, что в тексте присутствуют иероглифы. Решение устранения пока найти не удалось.

## Задача 3

Реализация скрипта:



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

# Определение функции для генерации случайной буквы
generate_random_letter() {
    # Генерация случайного числа в диапазоне от 0 до 25
    random_number=$(( $RANDOM % 26 ))

    # Преобразование случайного числа в соответствующую букву латинского алфавита
    random_letter=$(printf "\\x$(printf %x $(( 65 + random_number )))")

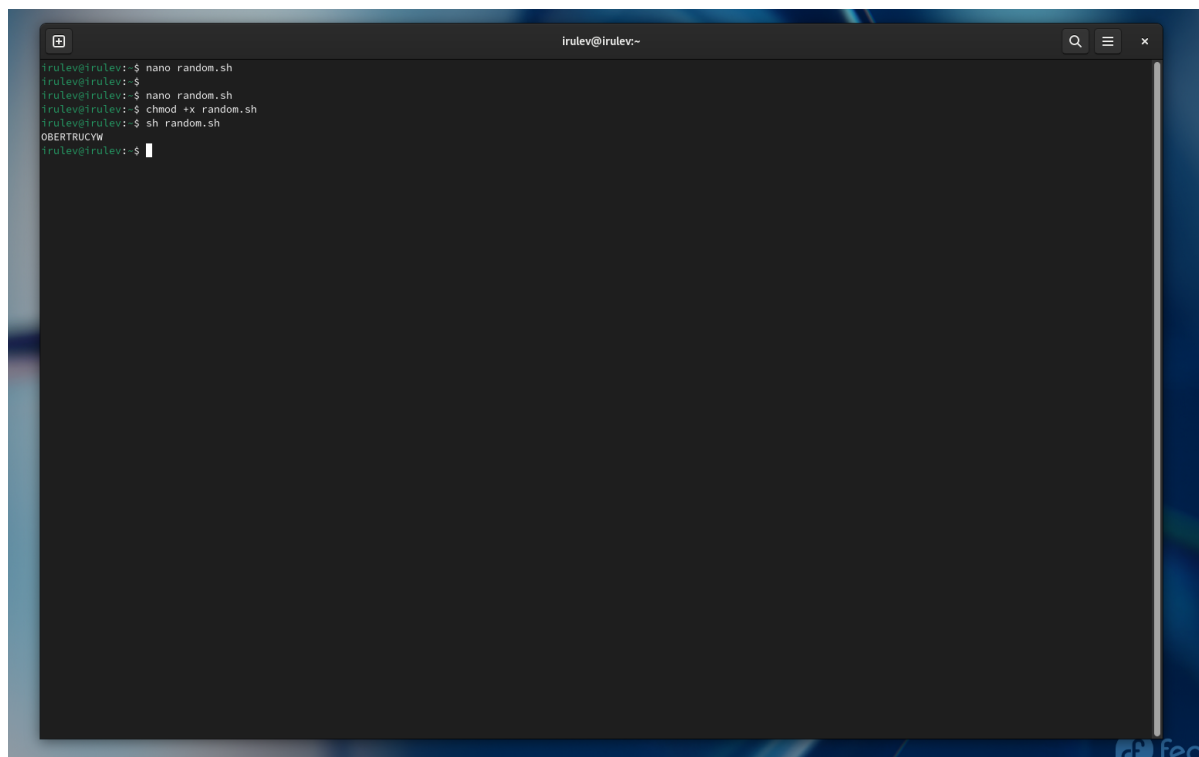
    # Вывод случайной буквы
    echo -n "$random_letter"
}

# Определение количества символов в последовательности
sequence_length=10

# Генерация случайной последовательности букв
for (( i=0; i<$sequence_length; i++ )); do
    generate_random_letter
done

# Вывод символа новой строки
echo
```

Этот скрипт генерирует случайную последовательность букв латинского алфавита, используя переменную `$RANDOM` для генерации случайного числа между 0 и 25, а затем используя команду `printf` для преобразования числа в букву латинского алфавита (A-Z). Функция `generate_random_letter` вызывается 10 раз для генерации последовательности из 10 случайных букв.



```
irulev@irulev:~$ nano random.sh
irulev@irulev:~$
irulev@irulev:~$ nano random.sh
irulev@irulev:~$ chmod +x random.sh
irulev@irulev:~$ sh random.sh
QBERTRUCYW
irulev@irulev:~$
```

# Выводы

---

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

## Ответы на контрольные вопросы

---

1. Синтаксическая ошибка в строке `while [ $1 != "exit" ]` заключается в отсутствии пробелов между скобками и переменными/операторами. Это должно быть `while [ "$1" != "exit" ]`.
2. Чтобы объединить несколько строк в одну, вы можете использовать следующий синтаксис: `result="${str1}${str2}${str3}"`.
3. Утилита `seq` генерирует последовательность чисел. Его можно заменить циклом `for` или использовать синтаксис `{start..end}` в `bash`. например, `for ((i=1; i<=10; i++)); do echo $i; done` или «эхо {1..10}».
4. Результатом выражения `$((10/3))` является `3`.
5. Основные различия между `zsh` и `bash`:
  - `zsh` имеет более продвинутые функции завершения и подстановки.
  - `zsh` имеет более мощный синтаксис для сценариев оболочки.
  - `zsh` имеет лучшую поддержку Unicode и интернационализации.
  - `zsh` имеет более настраиваемую подсказку.
1. Синтаксис `for ((a=1; a <= LIMIT; a++))` верен.
2. Bash часто сравнивают с другими языками сценариев, такими как Perl, Python и Ruby. Преимущество Bash заключается в тесной интеграции со средой командной строки Unix/Linux, что делает его мощным инструментом для задач системного администрирования и автоматизации. Однако ему может не хватать некоторых функций и гибкости языков программирования более общего назначения.