Лабораторная работа №14

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Полякова Юлия Александровна

Содержание

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться 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: Написать командный файл, реализующий упрощённый механизм семафоров. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.(рис. 1).

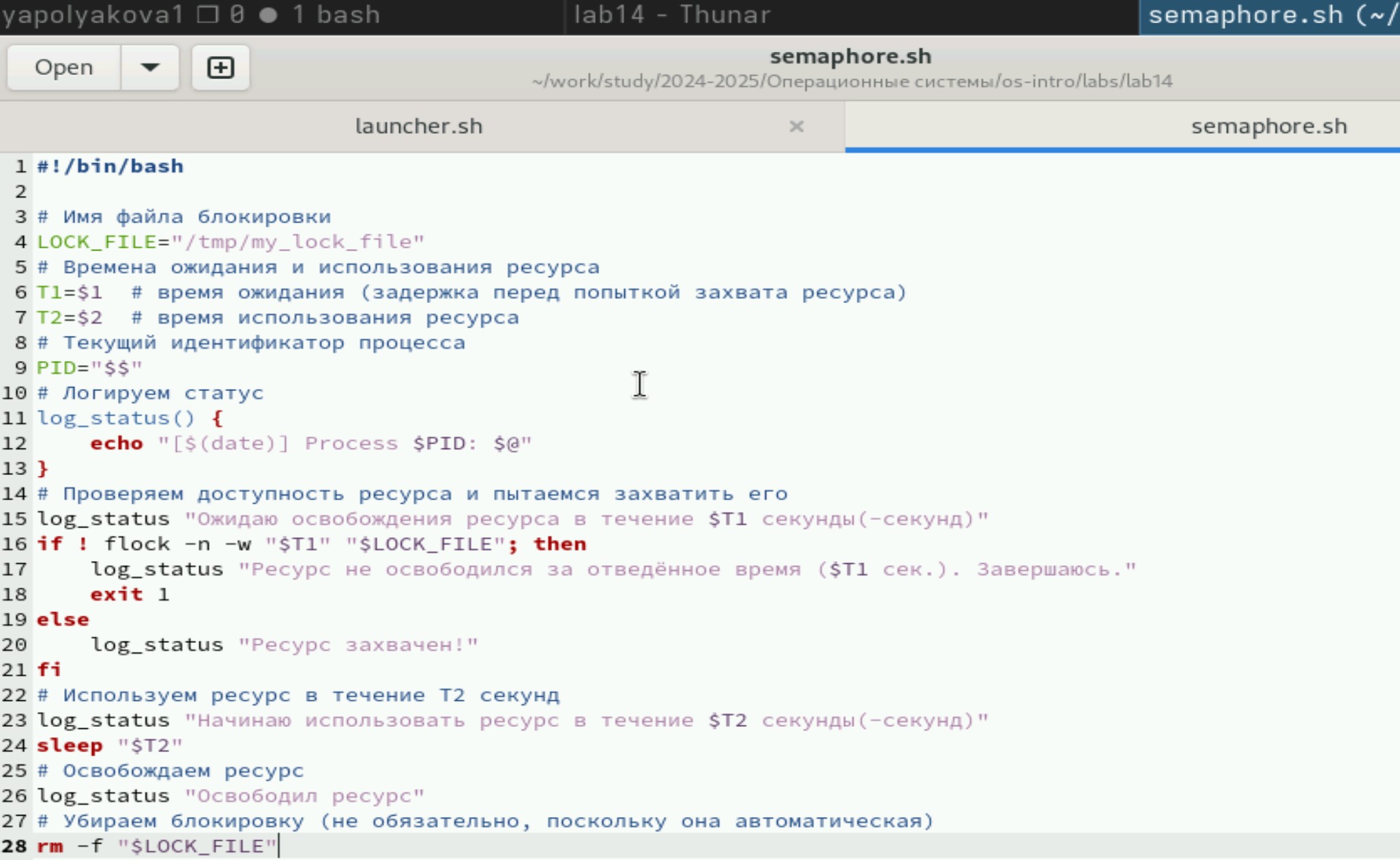


Рис. 1: Листинг semaphore.sh

1. Для запуска пишем отдельную программу (рис. 2)

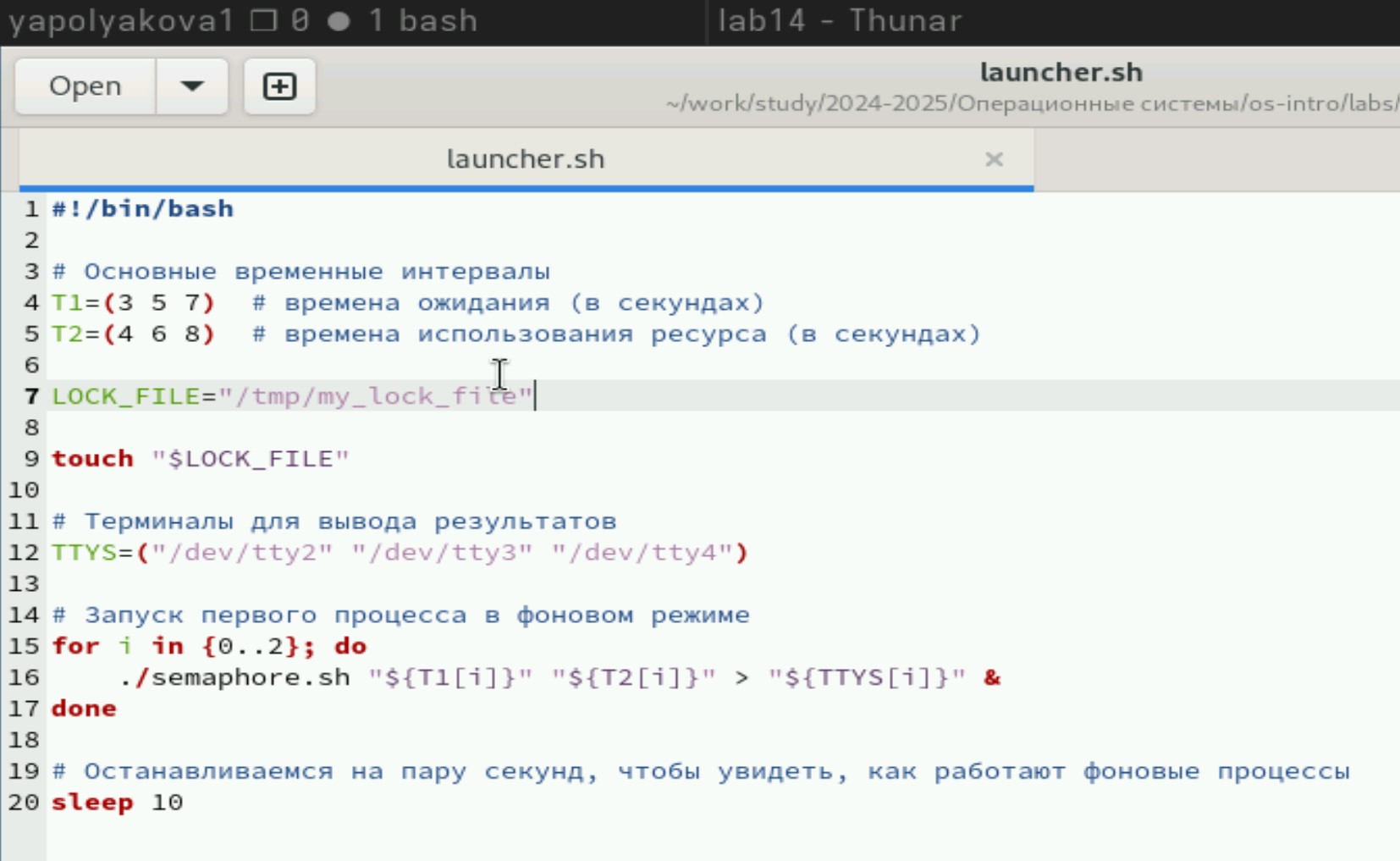


Рис. 2: Листинг launcher.sh

1. Запускаем первый скрипт так (рис. 3)

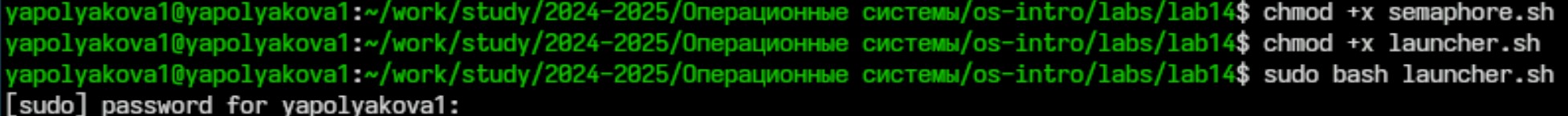


Рис. 3: Запуск launcher.sh

1. Рассмотрим второе задание: Реализовать команду man с помощью командного файла. (рис. 4)

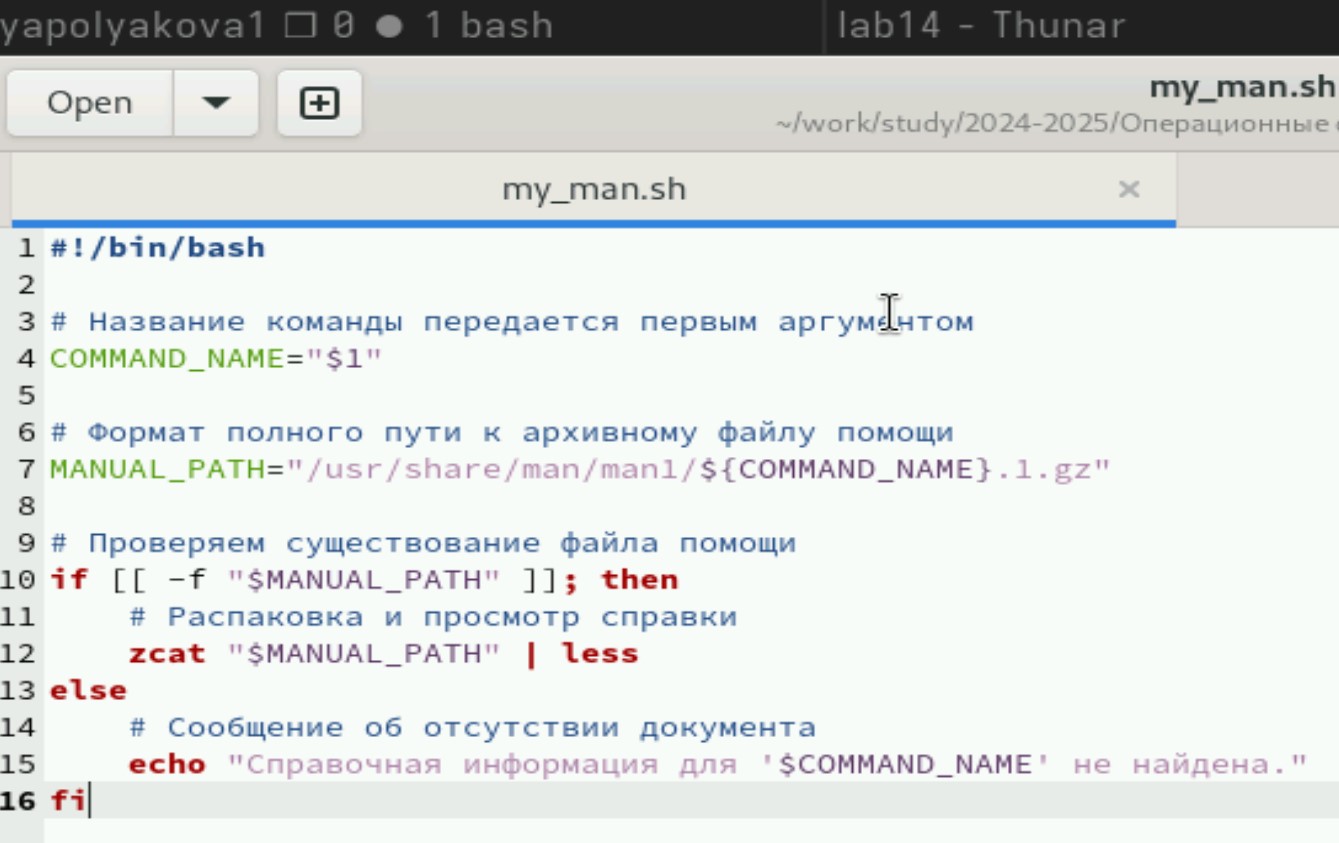


Рис. 4: Листинг my\_man.sh

1. Запуск второго задания с существующей и несуществующей командой (рис. 5)

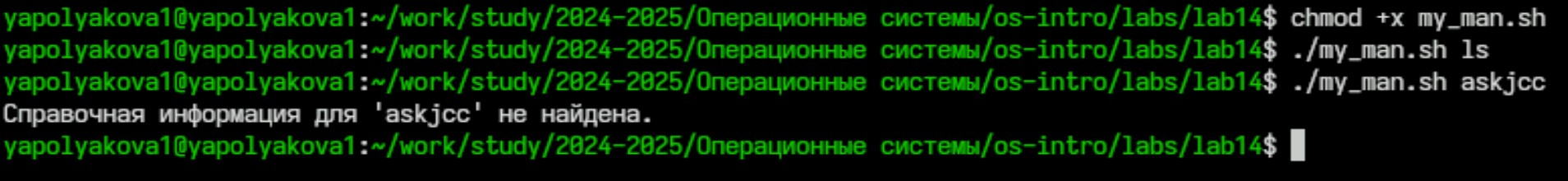


Рис. 5: Запуск my\_man.sh

1. Открывается справка, если команда существует. Здесь ls (рис. 6)

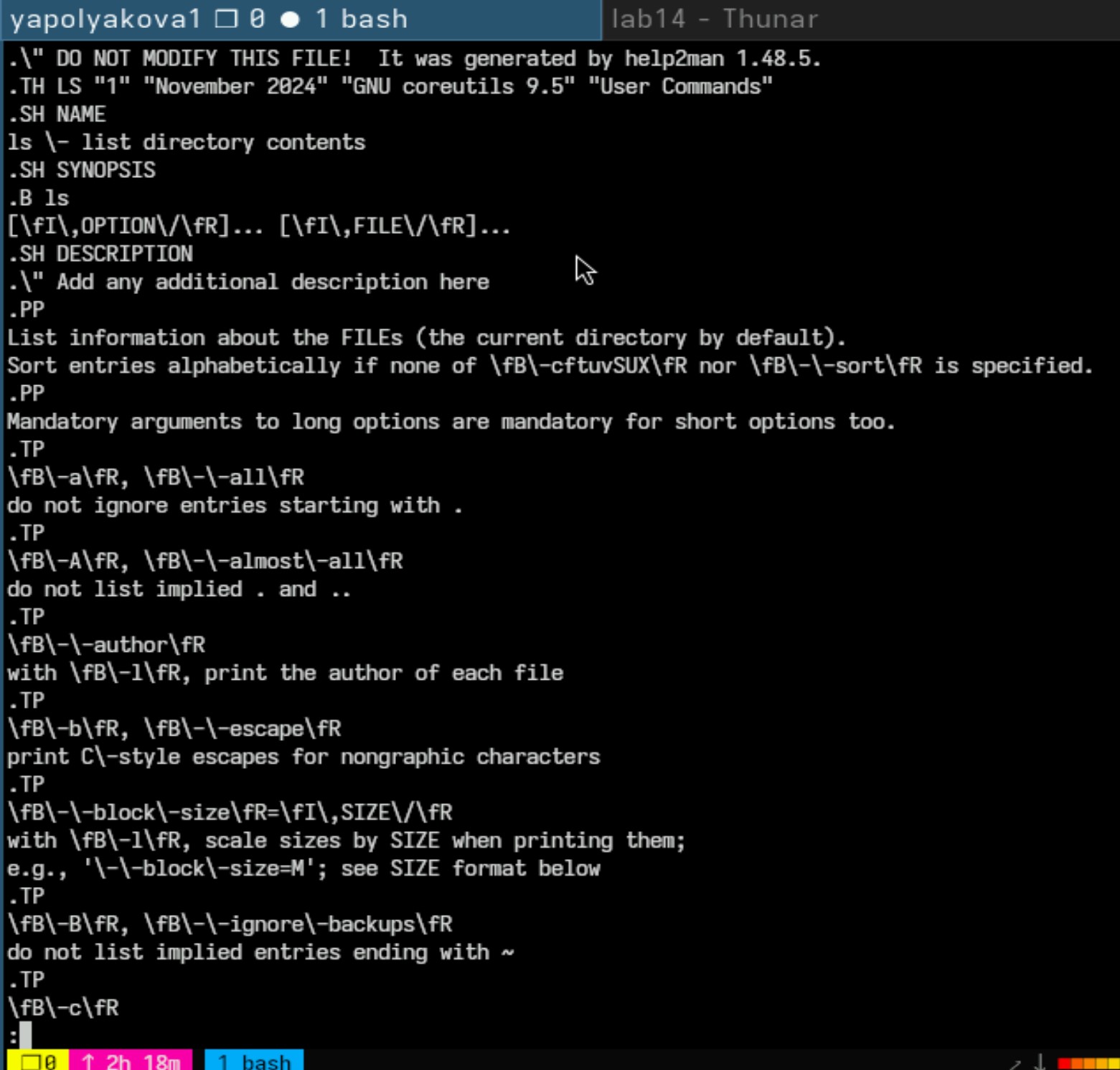


Рис. 6: Справка после запуска

1. Третье задание: Используя встроенную переменную $RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. (рис. 7)

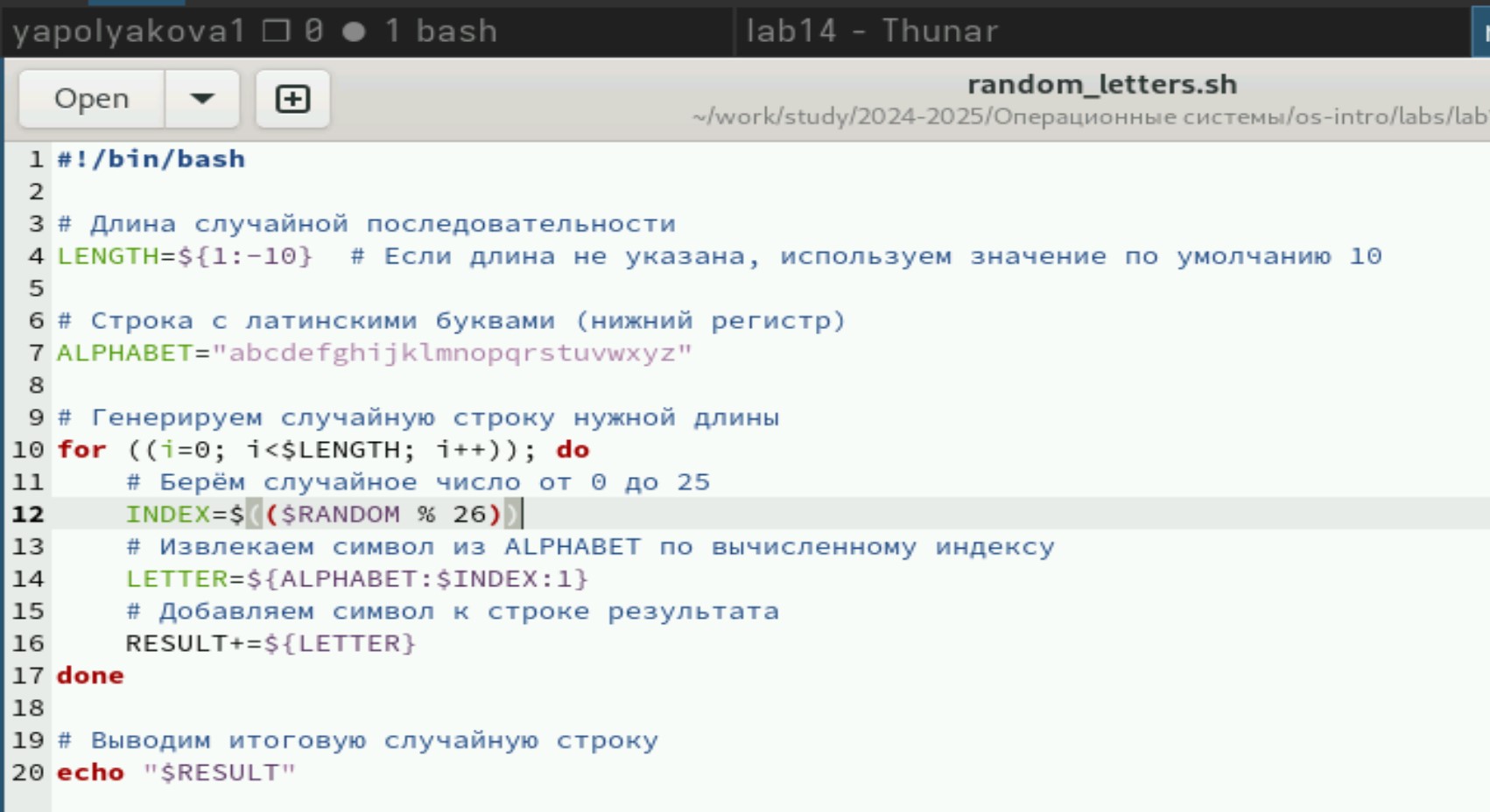


Рис. 7: Листинг random\_letters.sh

1. Как и со всеми предыдущими файлами, открываем доступ и запускаем (рис. 8)

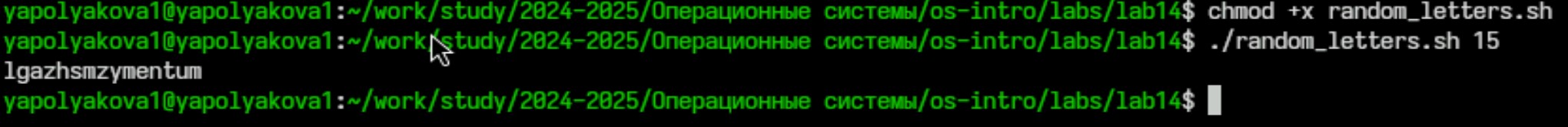


Рис. 8: Запуск random\_letters.sh

# 4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

while [$1 != "exit"]

**Ошибка:** Пробелы и кавычки отсутствуют. Правильная форма должна выглядеть следующим образом:

while [ "$1" != "exit" ]

1. Как объединить (конкатенация) несколько строк в одну?

Конкатенация строк осуществляется простым соединением переменных или строковых значений с использованием оператора + или же путем простого следования друг за другом.

Пример:

string="Hello "  
string+="World!"  
echo "$string" # Output: Hello World!

Альтернативный способ — использование команды printf:

var1="Hello"  
var2="World"  
result=$(printf "%s%s\n" "$var1" "$var2")  
echo "$result" # Output: HelloWorld

1. Утилита seq

Утилита **seq** используется для генерации последовательностей чисел. Она принимает аргументы, определяющие начальное значение, конечное значение и шаг последовательности.

Примеры использования:

* Последовательность от 1 до 10:
* seq 1 10
* Последовательность от 1 до 10 с шагом 2:
* seq 1 2 10

Альтернативные способы реализации функционала seq в Bash:

* Использование цикла for:
* for i in {1..10}; do echo "$i"; done
* Используя цикл while:
* i=1  
  while [ $i -le 10 ]; do  
   echo "$i"  
   let i++  
  done

1. Какой результат даст выражение `?

Результатом целочисленного деления является округленное вниз число. То есть:

$((10 / 3)) = 3

1. Основные отличия оболочек ZSH и BASH

* **ZSH**: Более расширенная функциональность по умолчанию, поддержка улучшенных автодополнений, псевдонимов, регулярных выражений прямо в шаблонах имен файлов, автоматическое исправление ошибок в командах (correc), встроенный механизм расширения путей (^, %) и т.п.
* **BASH**: Менее гибкая по функциональности, стандартная и широко используемая оболочка Unix-подобных операционных систем, доступная почти повсеместно.

Основные различия:

* **Автодополнение**: ZSH предлагает гораздо больше возможностей автоматического дополнения.
* **Коррекция ввода**: ZSH автоматически пытается исправить неправильно введённые команды.
* **Расширение путей**: ZSH поддерживает более удобные механизмы работы с путями.
* **Интерактивность**: ZSH имеет более развитые возможности интерактивного взаимодействия.

1. Проверка синтаксиса конструкции

for ((a=1; a <= LIMIT; a++))

Конструкция написана верно. Это правильный синтаксис циклов в стиле C в shell.

1. Сравнение Bash с другими языками программирования

Преимущества Bash:

* **Простота**: Bash удобен для написания простых скриптов автоматизации и управления системой.
* **Интеграция с Unix**: Легкость интеграции с файловыми системами, процессами и инструментами операционной системы.
* **Широкая доступность**: Доступен практически на всех системах Linux и Unix.
* **Скриптовая природа**: Позволяет быстро решать повседневные задачи администрирования и разработки.

Недостатки Bash:

* **Производительность**: Медленнее других языков программирования при сложных операциях и больших объемах данных.
* **Ограниченность типов данных**: Отсутствие поддержки классов, объектов и полноценных структур данных.
* **Отсутствие строгого контроля типов**: Ошибки легко пропустить из-за отсутствия проверки типов на этапе компиляции.
* **Недостаточная масштабируемость**: Сложно поддерживать большие проекты на Bash из-за ограничений среды исполнения.

# 5 Вывод

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