Отчёт для внешнего курса

Часть 3

Юсупова Ксения Равилевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Закончить выполнение внешнего курса

2 Выполнение лабораторной работы

2.1 Замена текста в Vim (рис. 2.1)

Решение:

```
:%s/Windows/Linux/
```

Объяснение: Команда производит замену первого вхождения “Windows” на “Linux” в каждой строке файла. Символ % указывает на применение ко всему файлу, а отсутствие флага g обеспечивает замену только первого совпадения в строке.

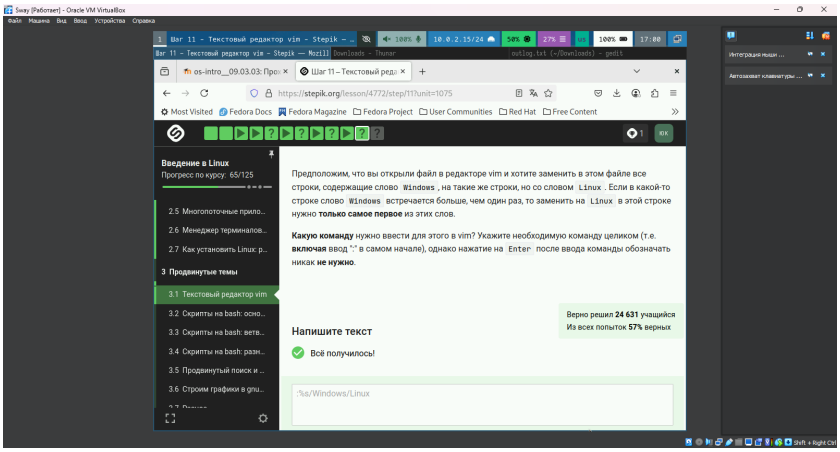

Рис. 2.1: Пример замены текста

2.2 Конструкция case в bash (рис. 2.2)

Пример кода:

```
case $1 in
  0) echo "zero";;
  1) echo "one";;
  *) echo "other";;
esac
```

Разбор: Конструкция case обеспечивает ветвление по значению переменной $1. Символ * обрабатывает все случаи, не указанные явно. Вертикальные черты ;; обозначают конец каждого блока условий.

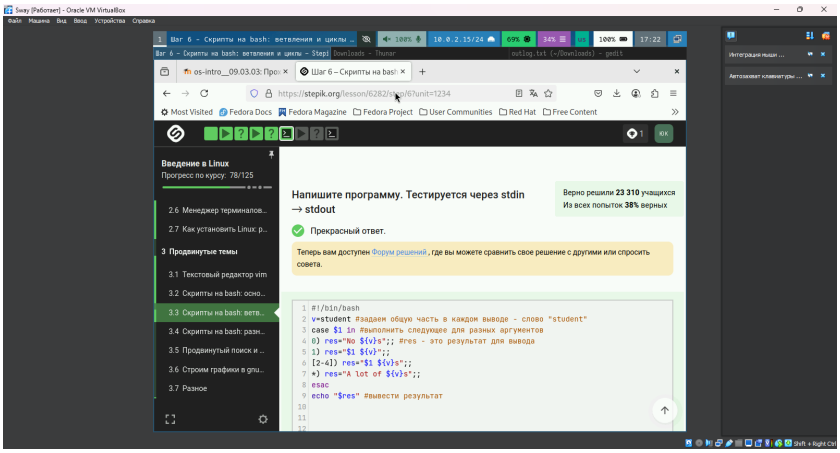

Рис. 2.2: Пример case-конструкции

2.3 Работа с аргументами (рис. 2.3)

Правильное решение:

```
echo "Arguments: \$1=$1 \$2=$2"
```

Ключевые моменты: Экранирование символа $ позволяет вывести его как текст, а не как начало переменной. Позиционные аргументы $1 и $2 содержат первый и второй параметры скрипта соответственно.

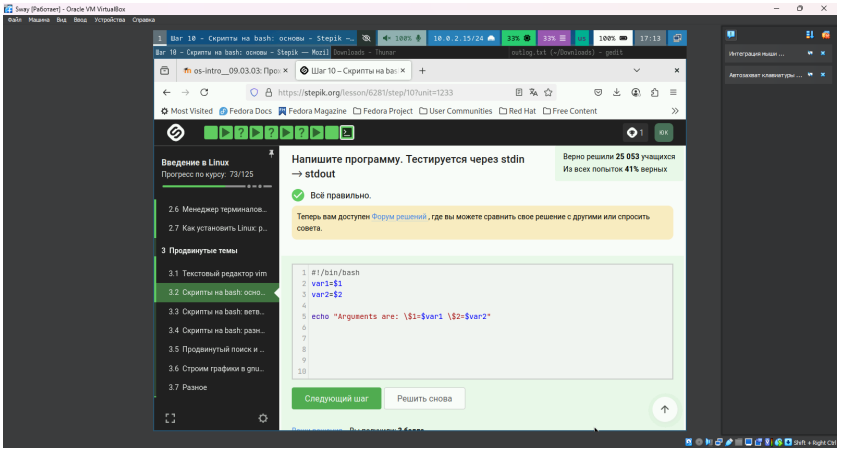

Рис. 2.3: Пример работы с аргументами

2.4 Проверка условий (рис. 2.4)

Верный синтаксис:

```
if [ $? -eq 0 ]; then
   echo "Success"
fi
```

Типичные ошибки: 1. Отсутствие пробелов внутри квадратных скобок

2. Неправильное сравнение (= вместо -eq для чисел)

3. Использование двойных скобок без пробелов

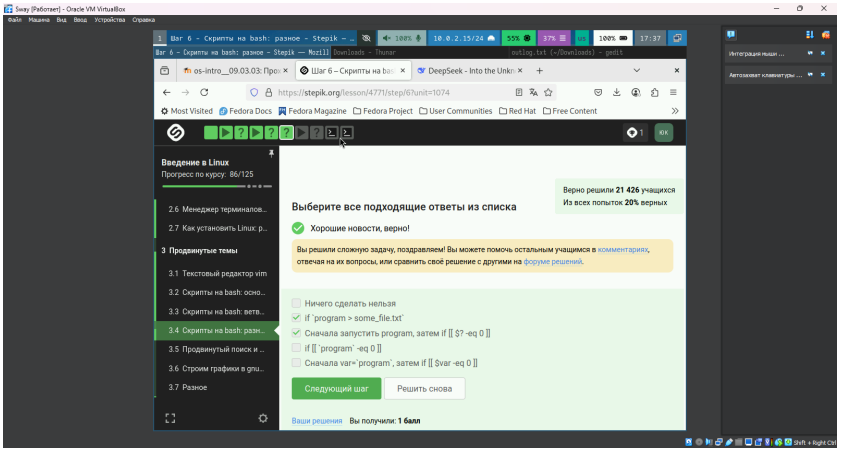

Рис. 2.4: Пример проверки условий

2.5 Подстановка команд (рис. 2.5)

Особенности: - `$(pwd)` подставляет вывод команды

- $? содержит код возврата последней команды
- Для вывода текущего каталога нужен `echo $(pwd)`

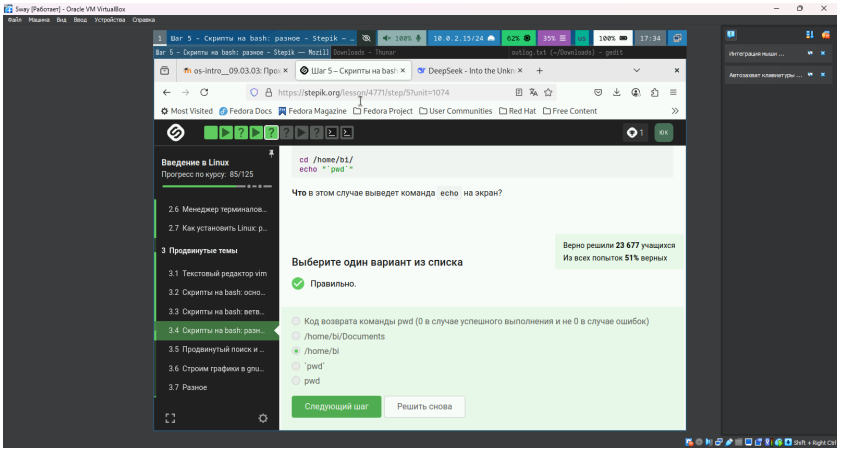

Рис. 2.5: Пример подстановки команд

2.6 Анализ ветвления (рис. 2.6)

Правильные варианты: - “two → four”

- “four → four”

Обоснование: Порядок вывода зависит от значения переменной $var и структуры условий в скрипте. Разные входные данные могут давать разную последовательность.

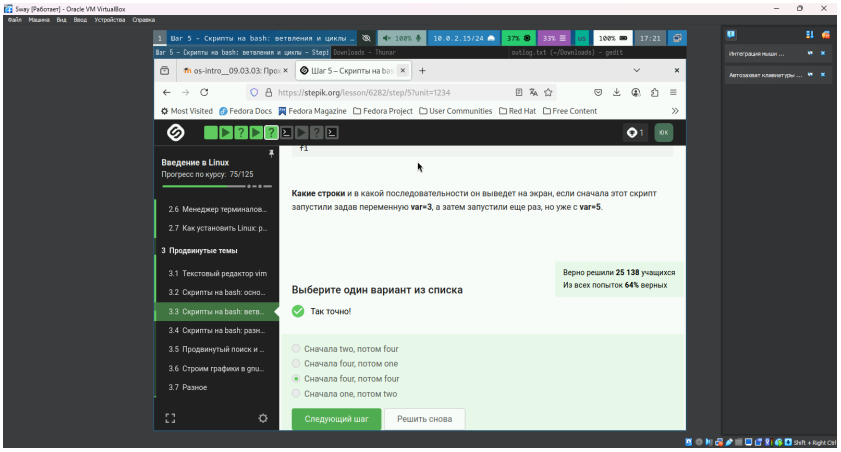

Рис. 2.6: Пример анализа ветвления

2.7 Работа с циклами (рис. 2.7)

Ответ: 3 вывода “start” и 2 вывода “finish”

Логика: Цикл выполняется 3 раза, выводя “start” на каждой итерации. “finish” выводится после первых двух итераций, но не после последней.

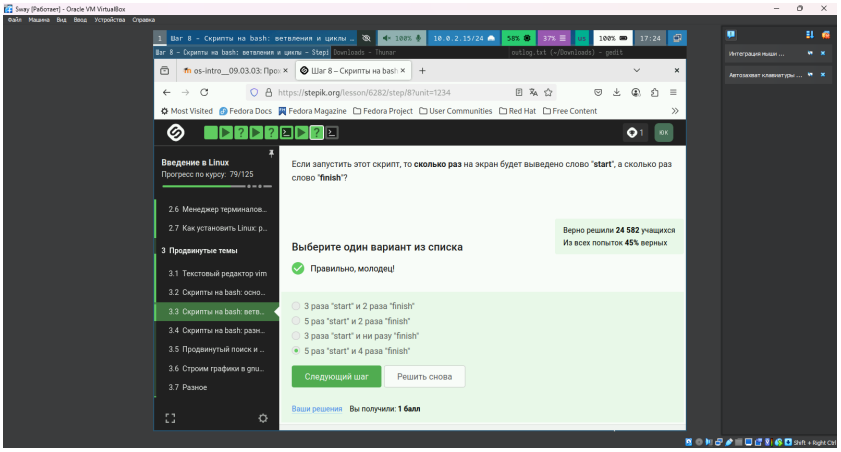

Рис. 2.7: Пример работы с циклами

2.8 Отладка скриптов (рис. 2.8)

Рекомендации:

1. Использовать `set -x` для отладки
2. Проверять коды возврата
3. Тестировать на разных входных данных

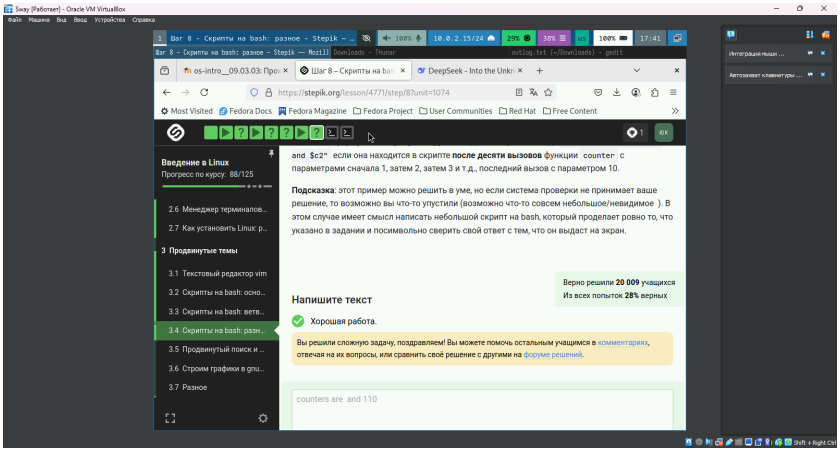

Рис. 2.8: Пример отладки

2.9 Работа с gnuplot (рис. 2.9)

Особенности: - Система проверки анализирует только команды скрипта - Для успешной проверки требуется максимально упрощенный синтаксис

Типичные проблемы: - Слишком сложные графики - Использование нестандартных модулей

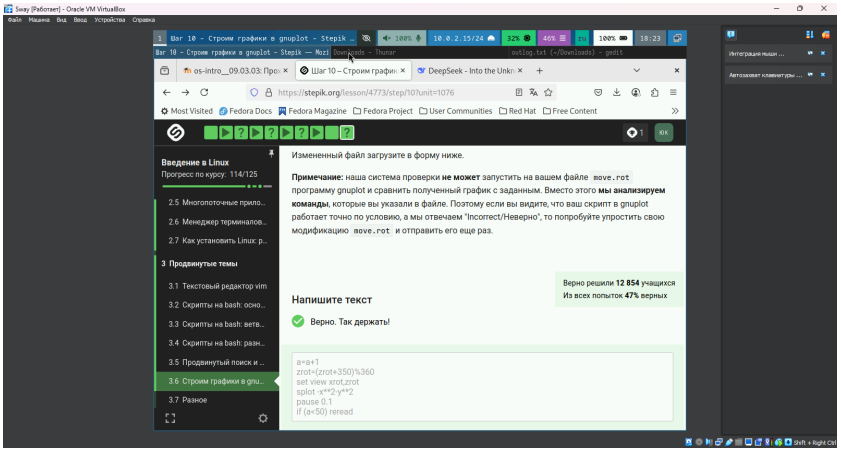

Рис. 2.9: Работа с gnuplot

2.10 Поиск файлов (рис. 2.10)

Различия команд find:

```
find -name "star*"   # Чувствителен к регистру
find -iname "star*"  # Игнорирует регистр
```

Правильные ответы:

- Star_Wars.avi (учет регистра)
- STARS.txt (разный шаблон поиска)

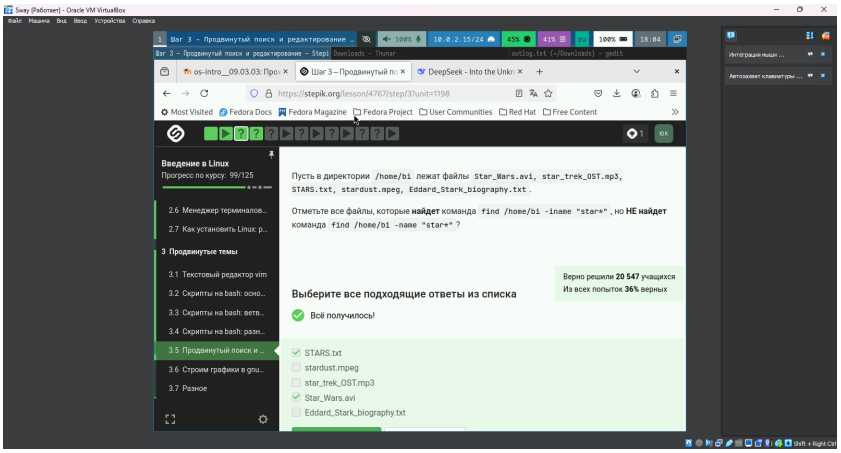

Рис. 2.10: Различия поиска файлов

2.11 Навигация в Vim (рис. 2.11)

Ключевые команды:

- `w` - перемещение по словам (word)
- `W` - перемещение по WORD (игнорирует пунктуацию)
- `$` - конец строки

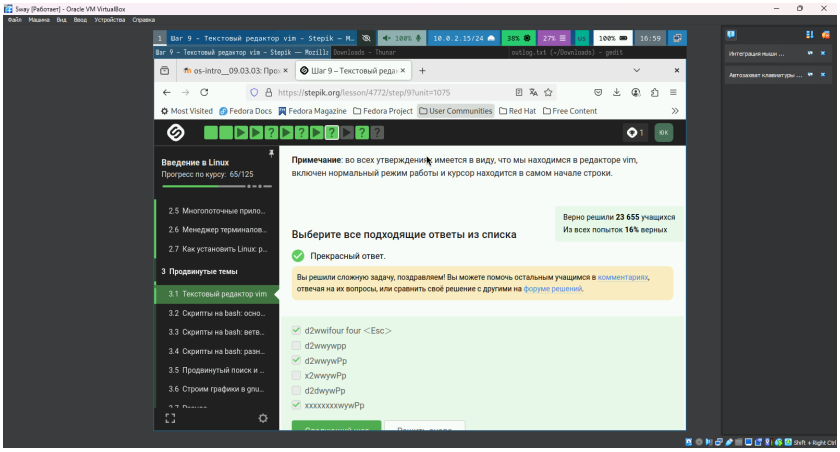

Рис. 2.11: Навигация в Vim

2.12 Ввод данных в bash (рис. 2.12)

Скрипт:

```
#!/bin/bash
while true; do
  read -p "Enter name: " name
  [[ -z $name ]] && { echo "bye"; break; }
  read -p "Enter age: " age
  echo "Name: $name, Age: $age"
done
```

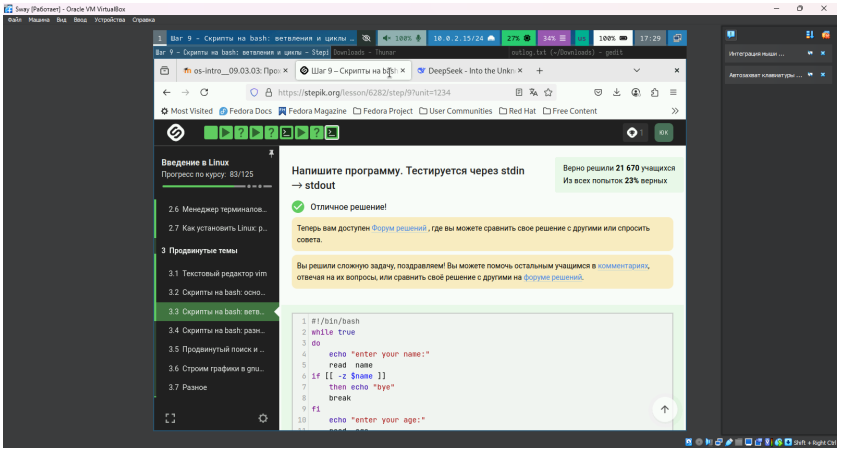

Рис. 2.12: Ввод данных

2.13 Особенности синтаксиса (рис. 2.13)

Критические моменты:

```
[ $var = "value" ]  # Требует пробелы
[[ $var == value ]] # Допускает без кавычек
```

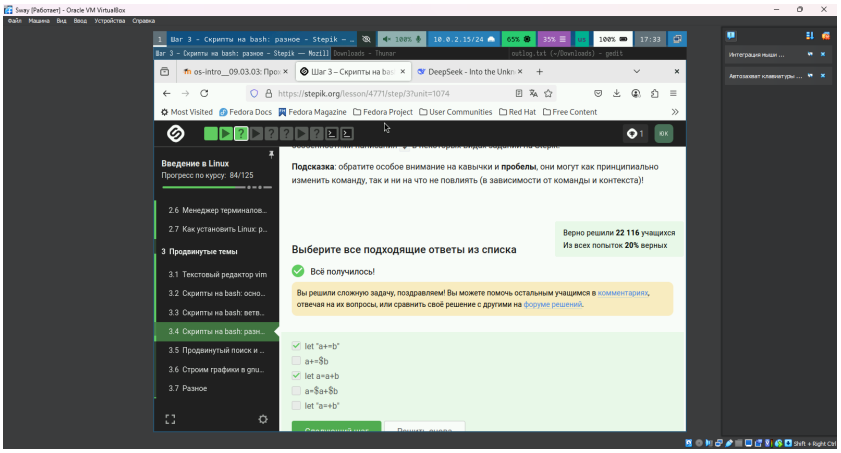

Рис. 2.13: Синтаксис bash

3 Выводы

В ходе работы мы закончили выполнение внешнего курса