# Отчёт по прохождению внешнего курса

**3 этап**

Ярослав Антонович Меркулов

# Содержание

**1 Цель работы** **4**

**2 Выполнение этапа** **5**

**3 Выводы** **17**

# Список иллюстраций

2.1 Задание 1 . . . . . 5
2.2 Задание 2 . . . . . 6
2.3 Задание 3 . . . . . 6
2.4 Задание 4 . . . . . 7
2.5 Задание 5 . . . . . 8
2.6 Задание 6 . . . . . 8
2.7 Задание 7 . . . . . 9
2.8 Задание 8 . . . . . 10
2.9 Задание 9 . . . . . 11
2.10 Задание 10 . . . . . 12
2.11 Задание 14 . . . . . 14
2.12 Задание 15 . . . . . 14
2.13 Задание 16 . . . . . 15
2.14 Задание 17 . . . . . 15
2.15 Задание 18 . . . . . 16

# 1 Цель работы

Пройти третий этап внешнего курса “Введение в Linux”.

# 2 Выполнение этапа

1. Теоретический вопрос (рис. 2.1).

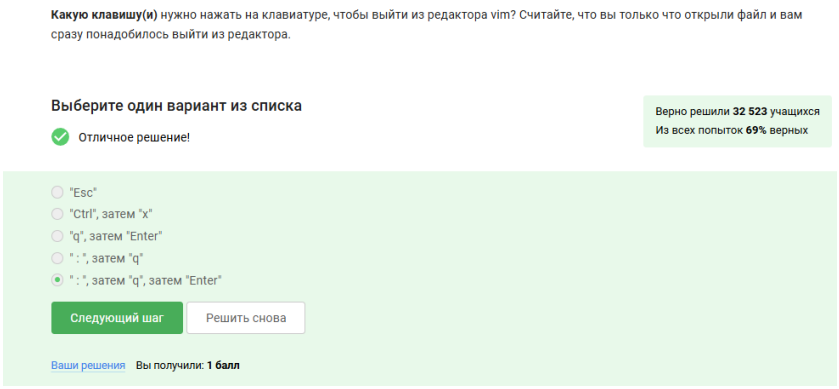



Рис. 2.1: Задание 1

2. Вот объяснение команды :%s/Windows/Linux в vim:

: — вводит командный режим в vim. % — означает, что команда применяется ко всему файлу (все строки). s — команда замены (substitute). /Windows/ — шаблон, который ищется в строке; в данном случае слово “Windows”. /Linux — текст, на который заменяется найденный шаблон; в данном случае “Linux”.(рис. 2.2).

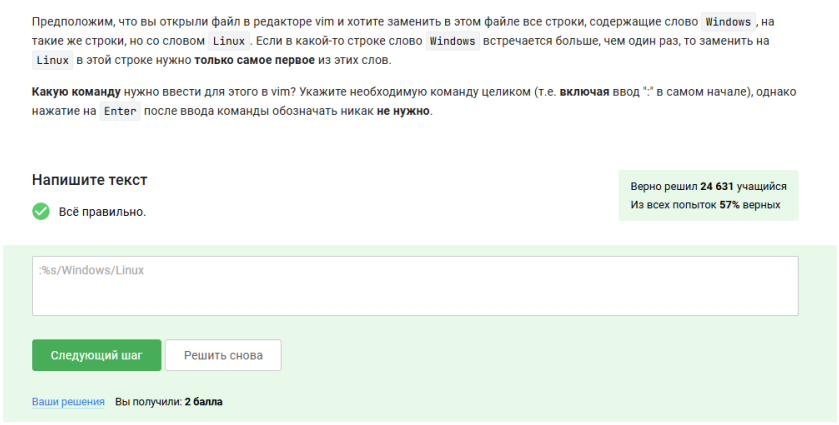



Рис. 2.2: Задание 2

3. Команды из набора A (первой оболочки bash) не будут доступны, потому что они были введены в другой сессии. Команды из набора B (оболочка sh) не будут доступны, потому что они были введены в другой сессии sh. Команды из набора C (последняя запущенная оболочка bash) будут доступны при перемещении по истории.(рис. 2.3).

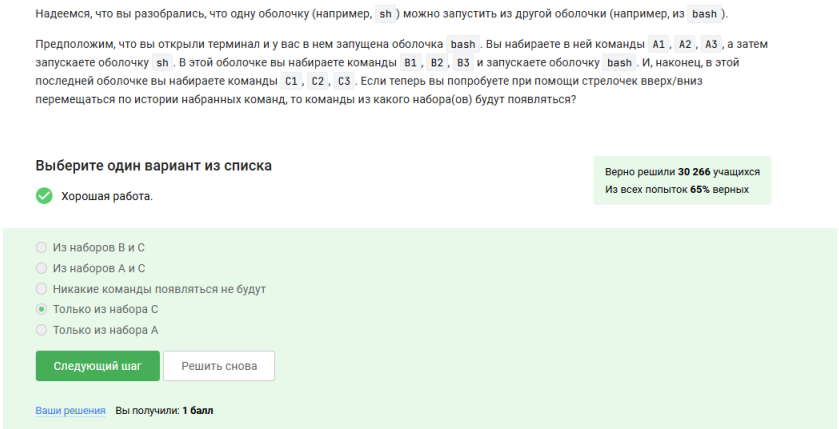



Рис. 2.3: Задание 3

4. Файл создан в каталоге /home/bi. Потом просто перешли в другой каталог. Это не поменяло путь к файлу (рис. 2.4).

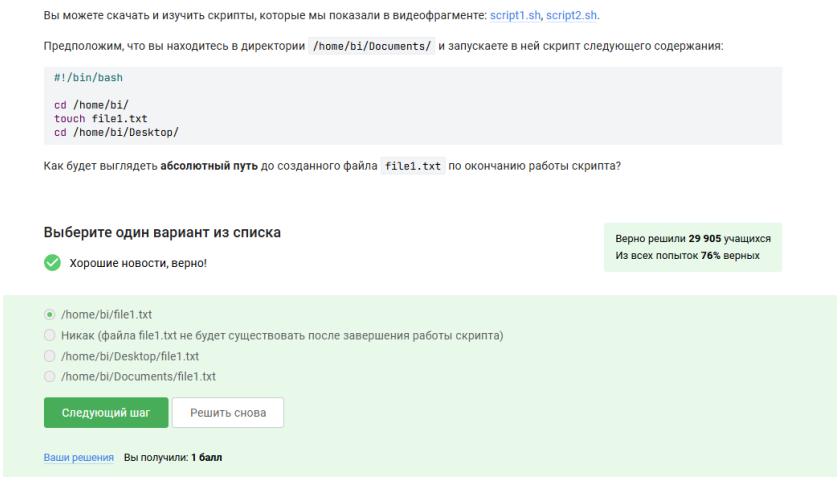



Рис. 2.4: Задание 4

5. VARiable

Начинается с буквы, содержит только буквы. Подходит 123variable

Начинается с цифры. Не подходит variable_123

Начинается с буквы, содержит буквы, цифры и подчеркивание. Подходит variable

Начинается с буквы, только буквы. Подходит variable123

Начинается с буквы, содержит буквы и цифры. Подходит _variable

Начинается с подчеркивания, что допустимо. Подходит __variable

Начинается с двух подчеркиваний, что тоже допустимо. Подходит

(рис. 2.5).

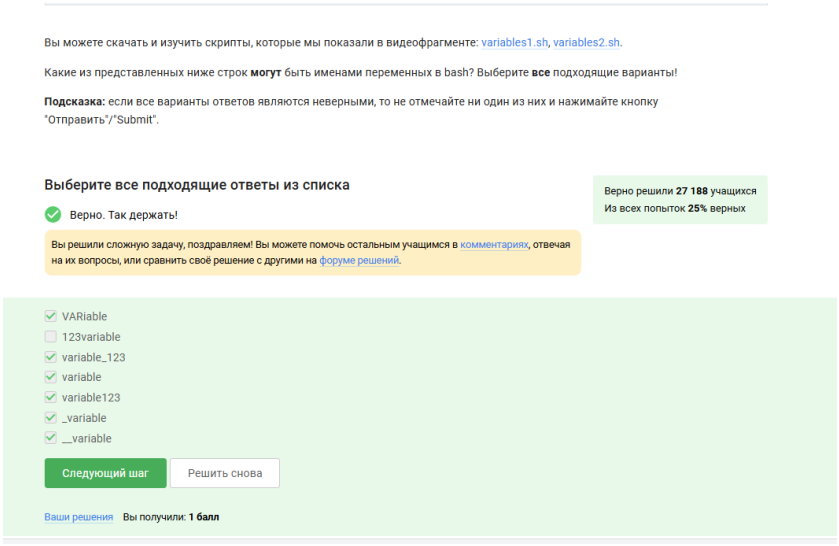



Рис. 2.5: Задание 5

6. Легкий скрипт, который выводит то, что получает(рис. 2.6).

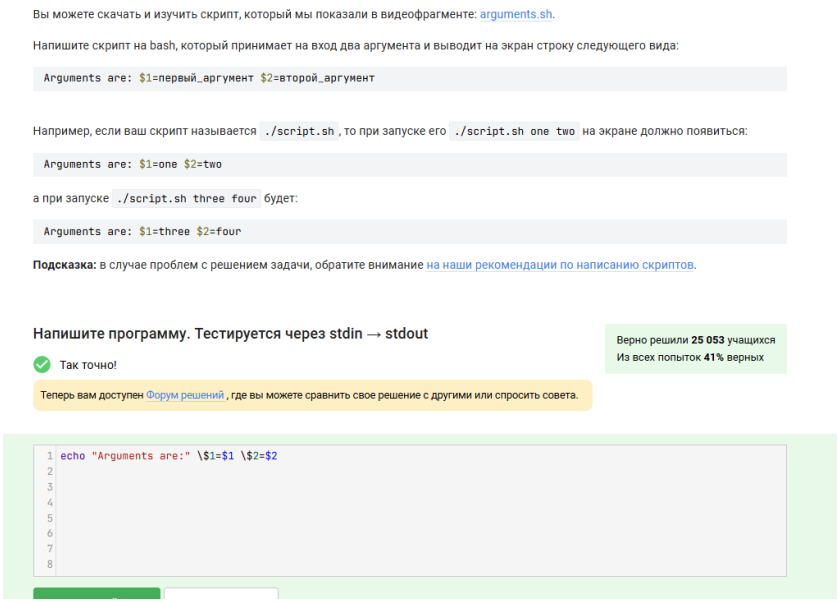



Рис. 2.6: Задание 6

7. Довольно сложное задание. На решение понадобилось достаточно много времени.(рис. 2.7).

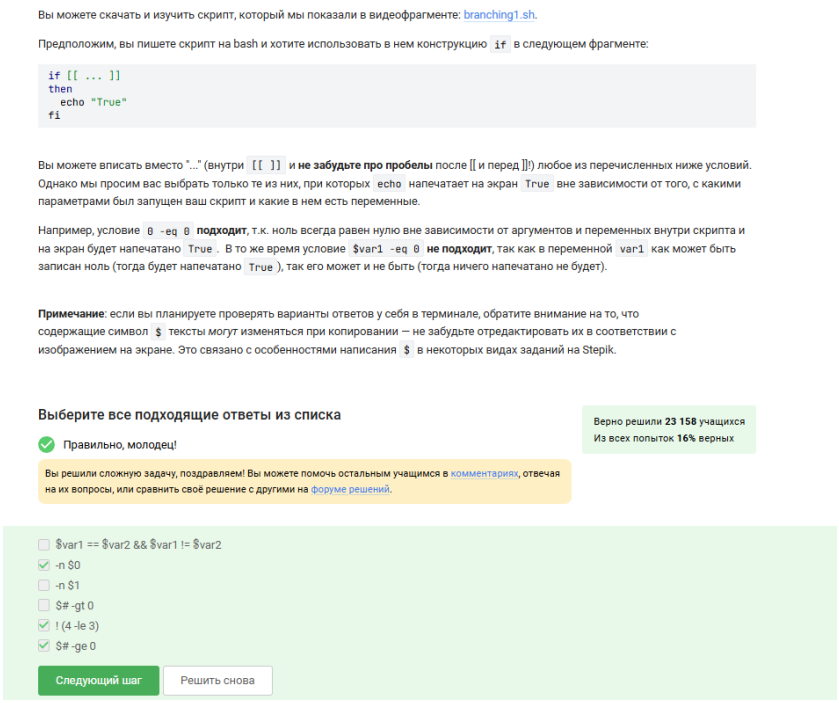



Рис. 2.7: Задание 7

8. Первый запуск: var=3 Подставим var=3 в условия:

if [[] — 3 > 5? Нет, условие ложно. elif [[] — 3 3? Нет, равно, условие ложно. elif [ ] — 3 == 4? Нет, условие ложно. Все условия ложны, значит выполнится блок else:

echo “four”

Второй запуск: var=5 Подставим var=5:

if [] — 5 > 5? Нет, равно, условие ложно. elif [] — 5 3? Нет, условие ложно. elif [] — 5 == 4? Нет, условие ложно. Все условия ложны, снова выполняется блок else:

echo “four”(рис. 2.8).

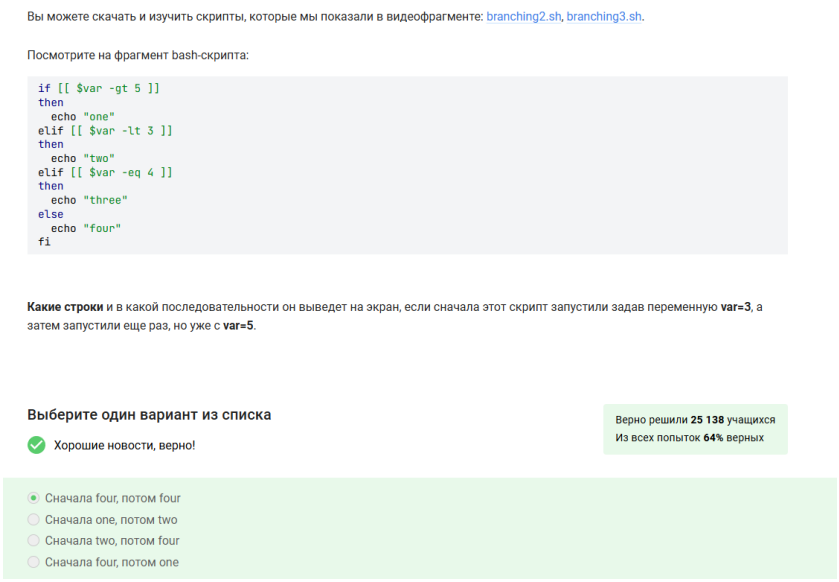
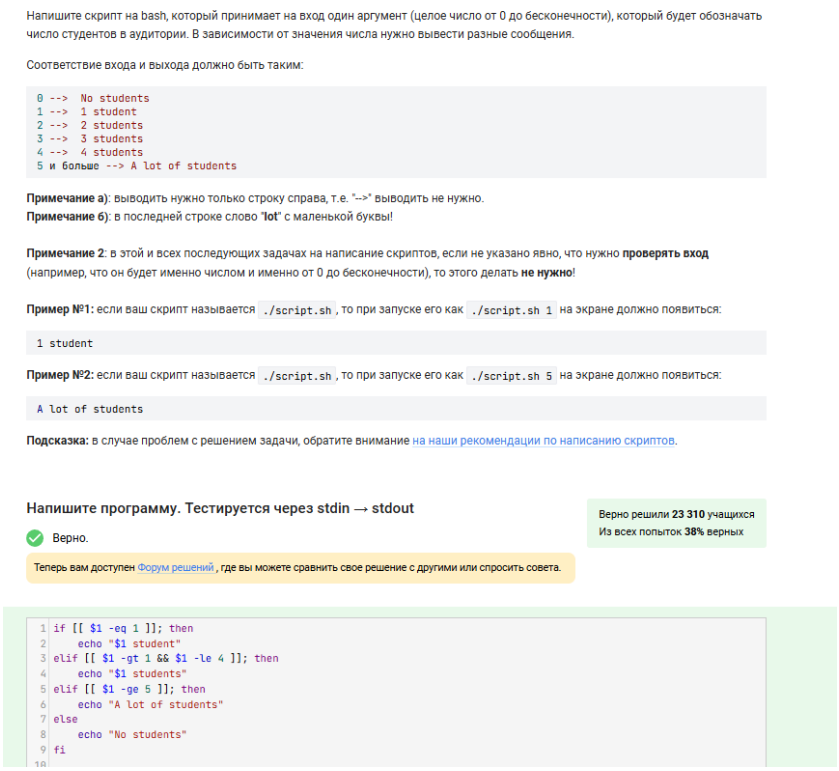



Рис. 2.8: Задание 8

9. Простая программа на bash, с обычными ветвлениями через if/elif/else(рис. 2.9).



Напишите скрипт на bash, который принимает на вход один аргумент (целое число от 0 до бесконечности), который будет обозначать число студентов в аудитории. В зависимости от значения числа нужно вывести разные сообщения.

Соответствие входа и выхода должно быть таким:

```
0 -->  No students
1 -->  1 student
2 -->  2 students
3 -->  3 students
4 -->  4 students
5 и больше --> A lot of students
```

**Примечание а)**: выводить нужно только строку справа, т.е. "-->" выводить не нужно.
**Примечание б)**: в последней строке слово "**lot**" с маленькой буквы!

**Примечание 2**: в этой и всех последующих задачах на написание скриптов, если не указано явно, что нужно **проверять вход** (например, что он будет именно числом и именно от 0 до бесконечности), то этого делать **не нужно**!

**Пример №1:** если ваш скрипт называется `./script.sh`, то при запуске его как `./script.sh 1` на экране должно появиться:

```
1 student
```

**Пример №2:** если ваш скрипт называется `./script.sh`, то при запуске его как `./script.sh 5` на экране должно появиться:

```
A lot of students
```

**Подсказка:** в случае проблем с решением задачи, обратите внимание на наши рекомендации по написанию скриптов.

**Напишите программу. Тестируется через stdin → stdout**

Верно решили **23 310** учащихся
Из всех попыток **38%** верных

Верно.

Теперь вам доступен Форум решений, где вы можете сравнить свое решение с другими или спросить совета.

```
if [[ $1 -eq 1 ]]; then
    echo "$1 student"
elif [[ $1 -gt 1 && $1 -le 4 ]]; then
    echo "$1 students"
elif [[ $1 -ge 5 ]]; then
    echo "A lot of students"
else
    echo "No students"
fi
```

Рис. 2.9: Задание 9

10. Подсчитаем сколько раз выводятся слова на каждой итерации:

1 a start Нет finish 2 , start Нет finish 3 b start Нет finish 4 , start Нет finish 5 c_d start Да continue (пропуск finish)

(рис. 2.10).

Вы можете скачать и изучить скрипты, которые мы показали в видеофрагменте: loops1.sh, loops2.sh.

Посмотрите на фрагмент bash-скрипта:

```
for str in a , b , c_d
do
  echo "start"
  if [[ $str > "c" ]]
  then
    continue
  fi
  echo "finish"
done
```

Если запустить этот скрипт, то **сколько раз** на экран будет выведено слово "**start**", а сколько раз слово "**finish**"?

**Выберите один вариант из списка**

Верно решили **24 582** уча
Из всех попыток **45%** вер

Отличное решение!

- 5 раз "start" и 4 раза "finish"
- 3 раза "start" и 2 раза "finish"
- 3 раза "start" и 3 раза "finish"
- 5 раз "start" и ни разу "finish"

Следующий шаг | Решить снова

Рис. 2.10: Задание 10

11. Опять же довольно простая программа. Надо только написать бесконечный цикл, прописать условия выхода(рис. **??**).

Напишите скрипт на bash, который будет определять в какую возрастную группу попадают пользователи. При запуске скрипт должен вывести сообщение "**enter your name:**" и ждать от пользователя ввода имени (используйте `read`, чтобы прочитать его). Когда имя введено, то скрипт должен написать "**enter your age:**" и ждать ввода возраста (опять нужен `read`). Когда возраст введен, скрипт пишет на экран "**<Имя>, your group is <группа>**", где **<группа>** определяется на основе возраста по следующим правилам:

- младше либо равно 16: "**child**",
- от 17 до 25 (включительно): "**youth**",
- старше 25: "**adult**".

После этого скрипт опять выводит сообщение "**enter your name:**" и всё начинается по новой (бесконечный цикл!). Если в какой-то момент работы скрипта будет введено **пустое имя** или **возраст 0**, то скрипт должен написать на экран "**bye**" и закончить свою работу (выход из цикла!).

**Примеры** корректной работы скрипта:

№1

```
./script.sh
enter your name:
Egor
enter your age:
16
Egor, your group is child
enter your name:
Elena
enter your age:
0
bye
```

```
child=16
adult=25
stdout=0

while [[ $stdout != 1 ]]
    do
        echo "enter your name: "
        read name
    if [[ (-z $name) || ($name = 0) ]] ;then
        echo "bye"
        stdout=1
    elif [[ -n $name ]]; then
        while [[ $stdout != 1 ]] ;do
            echo "enter your age: "
            read age
            if [[ ($age -eq 0) || (-z $age) ]] ;then
                echo "bye"
                stdout=1
            elif [[ $age -le $child ]] ;then
                echo "$name, your group is child"
            elif [[ $age -gt $adult ]] ; then
                echo "$name, your group is adult" ;else
                if [[ ($age -ge 17) && ($age -le 25) ]] ;then
                    echo "$name, your group is youth" ;fi
            fi ;break
        done ;fi
done
```

12. Программа для поиска НОД. Уже более сложная. Так же используется бесконечный цикл, if else, while.(рис. **??**).

Напишите скрипт на bash, который будет искать наибольший общий делитель (НОД, greatest common divisor, GCD) двух чисел. При запуске ваш скрипт не должен ничего писать на экран, а просто ждет ввода двух натуральных чисел через пробел (для этого можно использовать `read` и указать ему две переменные -- см. пример в видеофрагменте). После ввода чисел скрипт считает их НОД и выводит на экран сообщение **"GCD is <посчитанное значение>"**, например, для чисел 15 и 25 это будет "GCD is 5". После этого скрипт опять входит в режим ожидания двух натуральных чисел. Если в какой-то момент работы пользователь ввел вместо этого пустую строку, то нужно написать на экран "**bye**" и закончить свою работу.

Вычисление НОД несложно реализовать с помощью алгоритма Евклида. Вам нужно написать функцию `gcd`, которая принимает на вход два аргумента (назовем их **M** и **N**). **Если аргументы равны**, то мы нашли НОД -- он равен **M** (или **N**), нужно выводить соответствующее сообщение на экран (см. выше). Иначе нужно сравнить аргументы между собой. Если **M больше N**, то запускаем ту же функцию `gcd`, но в качестве первого аргумента передаем (**M-N**), а в качестве второго **N**. Если же наоборот, **M** меньше **N**, то запускаем функцию `gcd` с первым аргументом **M**, а вторым (**N-M**).

Пример корректной работы скрипта:

```
./script.sh
10 15
GCD is 5
7 3
GCD is 1

bye
```

**Примечание:** в вызове функции из себя самой нет ничего страшного или неправильного, т.ч. смело вызывайте `gcd` прямо внутри `gcd`!

**Примечание 2**: для завершения работы функции в произвольном месте, можно использовать инструкцию `return` (все инструкции функции после `return` выполняться не будут). В отличии от `exit` эта команда завершит только функцию, а не выполнение всего скрипта целиком. Однако в данной задаче можно обойтись и без использования `return!`

**Подсказка:** в случае проблем с решением задачи, обратите внимание на наши рекомендации по написанию скриптов.

```bash
# put your shell (bash) code here

while [ true ]
do
    read n1 n2
if [ -z $n1 ]; then
    echo "bye"
    break
else
    gcd () {
    remainder=1
    if [ $n2 -eq 0 ]
    then
    echo "bye"
    fi
    while [ $remainder -ne 0 ]
    do
    remainder=$((n1%n2))
    n1=$n2
    n2=$remainder
    done
    }
    gcd $1 $2
    echo "GCD is $n1"
fi
done
```

13. Калькулятор на bash(рис. **??**).

Напишите **калькулятор** на bash. При запуске ваш скрипт должен ожидать ввода пользователем команды (при этом на экран выводить ничего не нужно). Команды могут быть трех типов:

1. Слово "**exit**". В этом случае скрипт должен вывести на экран слово "bye" и завершить работу.
2. **Три аргумента через пробел** -- первый операнд (целое число), операция (одна из "+", "-", "*", "/", "%", "**") и второй операнд (целое число). В этом случае нужно произвести указанную операцию над заданными числами и вывести результат на экран. После этого переходим в режим ожидания новой команды.
3. **Любая другая команда** из одного аргумента или из трех аргументов, но с операцией не из списка. В этом случае нужно вывести на экран слово "**error**" и завершить работу.

Чтобы проверить работу скрипта, вы можете записать сразу несколько команд в файл и передать его скрипту на stdin (т.е. выполнить `./script.sh < input.txt`). В этом случае он должен вывести сразу все ответы на экран.

Например, если входной файл будет следующего содержания:

```
10 + 1
2 ** 10
exit
```

то на экране будет:

```
11
1024
bye
```

Если же на вход поступит следующий файл:

```
3 - 5
2/10
exit
```

```bash
# put your shell (bash) code here
#!/bin/bash
while [[ True ]]
do
  read birinchi amal ikkinchi
  if [[ $birinchi == "exit" ]]
  then
    echo "bye"
    break
  elif [[ "$birinchi" =~ "^[0-9]+$" && "$ikkinchi" =~ "^[0-9]+$" ]]
  then
    echo "error"
    break
  else
    case $amal in
"+") let "result = birinchi + ikkinchi";;
"-") let "result = birinchi - ikkinchi";;
"/") let "result = birinchi / ikkinchi";;
"*") let "result = birinchi * ikkinchi";;
"%") let "result = birinchi % ikkinchi";;
"**") let "result = birinchi ** ikkinchi";;
*) echo "error" ; break ;;
    esac
    echo "$result"
  fi
done
```

14. Объяснение параметров команды: -mindepth 2: искать файлы, начиная со второго уровня вложенности относительно /home/bi. -maxdepth 3: искать только до третьего уровня вложенности. -name “file*“: искать файлы, имена которых начинаются с”file”. Расположение файлов по уровням: /home/bi/dir1 — уровень 1 (относительно /home/bi) /home/bi/dir1/file1 — уровень 2 /home/bi/dir1/dir2 — уровень 2 /home/bi/dir1/dir2/file2 — уровень 3 /home/bi/dir1/dir2/dir3 — уровень 3 /home/bi/dir1/dir2/dir3/file3 — уровень 4

Анализ по условиям: -mindepth 2: ищем начиная со второго уровня, то есть начиная с /home/bi/... внутри. -maxdepth 3: ищем только до третьего уровня. Это значит, что ищем файлы, расположенные на уровнях 2 и 3.

Какие файлы подходят? file1: находится в /home/bi/dir1/file1, уровень 2 → подходит. file2: находится в /home/bi/dir1/dir2/file2, уровень 3 → подходит. file3: находится в /home/bi/dir1/dir2/dir3/file3, уровень 4 → не подходит, так как maxdepth=3.(рис. 2.11).

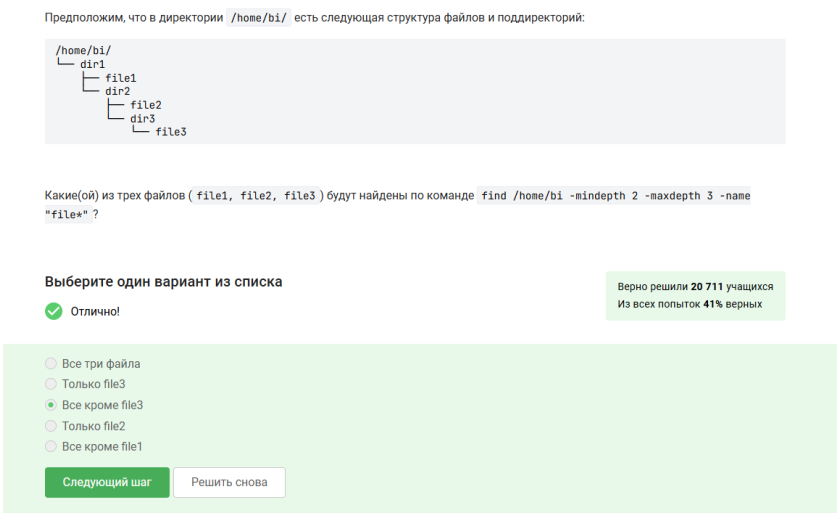



Рис. 2.11: Задание 14

15. Без -n, sed по умолчанию печатает каждую строку после обработки, независимо от команд внутри скрипта.(рис. 2.12).

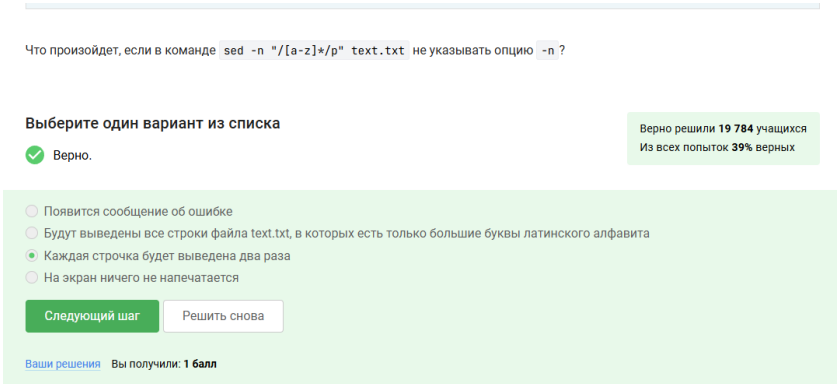



Рис. 2.12: Задание 15

16. Инструкция sed(рис. 2.13).

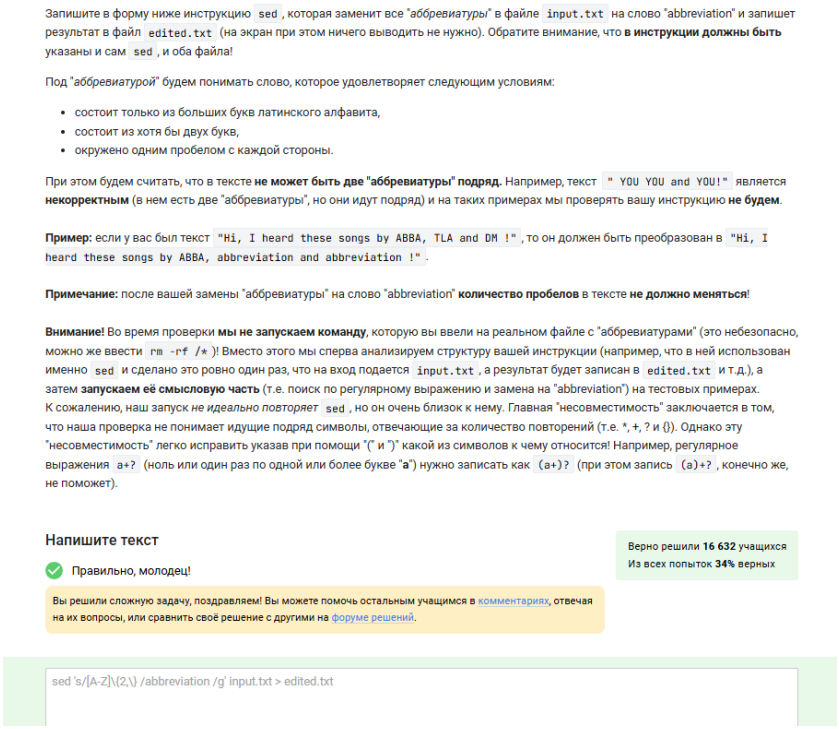



Рис. 2.13: Задание 16

17. gnuplot(рис. 2.14).

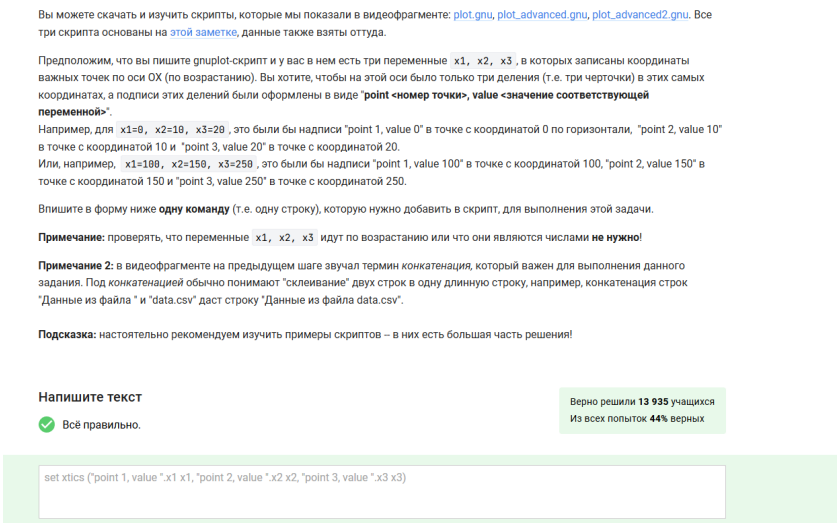



Рис. 2.14: Задание 17

18. Меняем работу программы под условие(рис. 2.15).

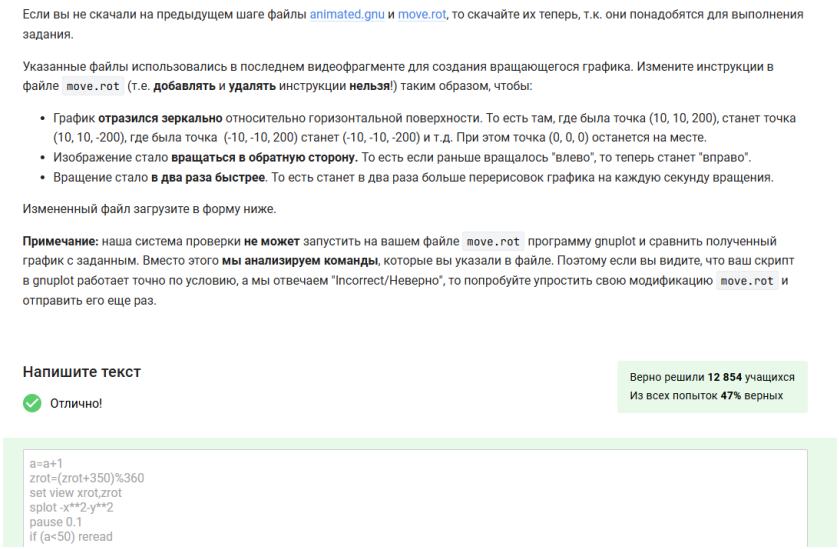



Рис. 2.15: Задание 18

# 3 Выводы

Были получены знания о Линуксе: vim, bash, gnuplot. Были выполнены тесты.