

Отчет о прохождении 3 этапа внешних курсов

Продвинутые темы

Беспутин Глеб Антонович, НКАбд-01-23

Содержание

1 Цель работы	6
2 Задание	7
3 Теоретическое введение	8
4 Выполнение лабораторной работы	9
5 Сертификат	35
6 Выводы	36
Список литературы	37

Список иллюстраций

4.1	Задание 1	9
4.2	Задание 2	10
4.3	Задание 3	11
4.4	Задание 4	12
4.5	Задание 5	12
4.6	Задание 6	13
4.7	Задание 7	13
4.8	Задание 8	14
4.9	Задание 9	14
4.10	Задание 10	15
4.11	Задание 11	16
4.12	Задание 12	16
4.13	Задание 13	17
4.14	Задание 14	18
4.15	Задание 14	18
4.16	Задание 15	20
4.17	Задание 16	20
4.18	Задание 16_2	21
4.19	Задание 17	21
4.20	Задание 18	22
4.21	Задание 18	22
4.22	Задание 19	23
4.23	Задание 19	23
4.24	Задание 20	24
4.25	Задание 21	24
4.26	Задание 22	25
4.27	Задание 23	25
4.28	Задание 24	26
4.29	Задание 24	26
4.30	Задание 25	27
4.31	Задание 26	27
4.32	Задание 27	28
4.33	Задание 28	28
4.34	Задание 29	29
4.35	Задание 30	30
4.36	Задание 31	31
4.37	Задание 32	32

4.38 Задание 33	32
4.39 Задание 34	33
4.40 Задание 35	33
5.1 Сертификат	35

Список таблиц

1 Цель работы

Ознакомиться с функционалом операционной системы Linux.

2 Задание

Просмотреть видео и на основе полученной информации пройти тестовые задания.

3 Теоретическое введение

Линукс - в части случаев GNU/Linux – семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты. Как и ядро Linux, системы на его основе, как правило, создаются и распространяются в соответствии с моделью разработки свободного и открытого программного обеспечения. Linux-системы распространяются в основном бесплатно в виде различных дистрибутивов – в форме, готовой для установки и удобной для сопровождения и обновлений, – и имеющих свой набор системных и прикладных компонентов, как свободных, так и проприетарных.

4 Выполнение лабораторной работы

3 Этап: (рис. 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, ??, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30, ??, 4.32, 4.33, 4.34, 4.35, 4.36, 4.37, 4.38, 4.39, 4.40, 5.1).

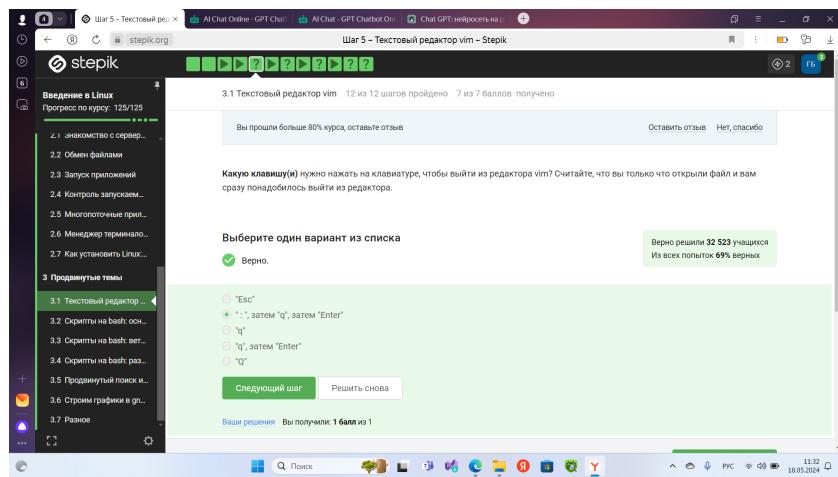


Рис. 4.1: Задание 1

Стоит упомянуть, что у редактора vim есть тьюториал, который позволяет разобраться с командами, необходимыми для стандартной работы. За выход из редактора отвечают следующие команды:

- ZQ - выйти без сохранения
- :q! - выйти без сохранения
- ZZ - записать файл и выйти (если файл не изменяли, то записываться он не будет)
- :wq - записать файл и выйти

- :x - записать файл и выйти
- :w - записать файл
- :sav filename - “сохранить как”
- :w! filename - “сохранить как”
- :w! - записать файл

Как мы видим, вариантов много, при этом каждый сможет найти тот, который подойдёт под конкретную ситуацию.

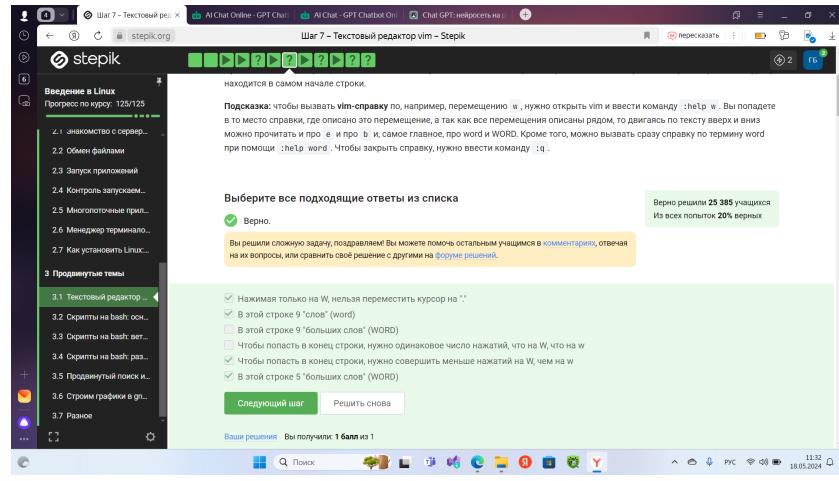


Рис. 4.2: Задание 2

`Strange_ TEXT is_here. 2=2 YES!`

Точка считается “маленьким словом”, так что всего их 9: `Strange_, is_here, ., 2, =, 2, !` и два лишних пробела.

И если посчитать нажатия на `w` и на `W`, то действительно после 10 штук попадем в одно место. 10 нажатий на `W`, это то же самое, что и 10 нажатий на `w`,

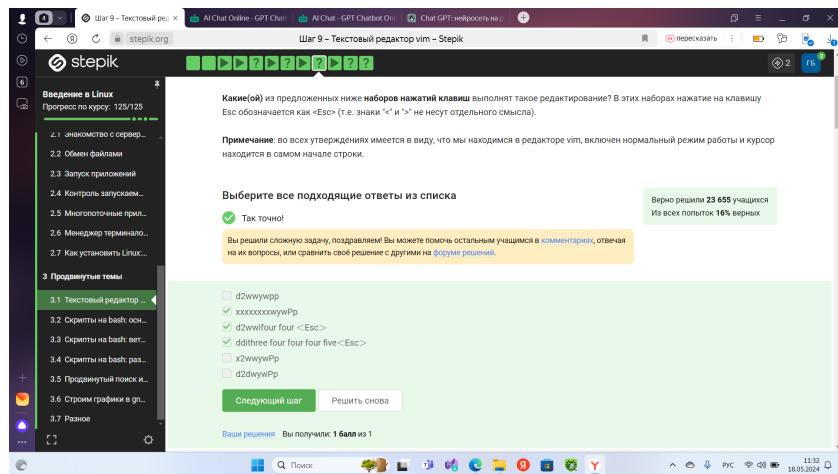


Рис. 4.3: Задание 3

```
d2wwifour four <<Esc>>
d2wwywPp
d2w$$bifour four <<Esc>>
```

- \$ — в конец текущей строки;
- w — на слово вправо;
- b — на слово влево;
- i — начать ввод перед курсором;
- p — вставка содержимого неименованного буфера под курсором;
- P — вставка содержимого неименованного буфера перед курсором;
- yy (также Y) — копирование текущей строки в неименованный буфер;
- yy — копирование числа строк начиная с текущей в неименованный буфер;

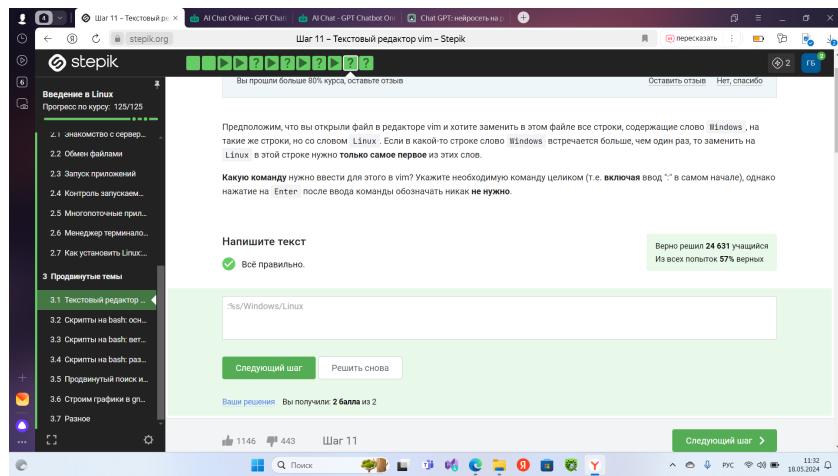


Рис. 4.4: Задание 4

Поиск и замена в редакторе работают по следующей схеме:

:{пределы}s/{что заменяем}/{на что заменяем}/{опции}

Для замены во всем файле можно использовать символ %.

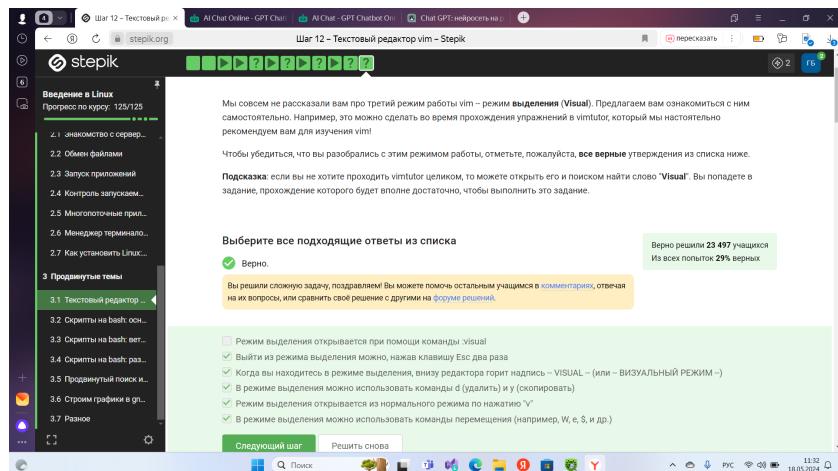


Рис. 4.5: Задание 5

Команда \$ – в конец текущей строки, W - до пробела вправо - то есть, перемещение.

Нажать Esc достаточно один раз, но да ладно.

Надпись visual - горит.

d — используется совместно с командами перемещения. Удаляет символы с текущего положения курсора до положения после ввода команды перемещения.
уу (также **Y**) — копирование текущей строки в буфер;

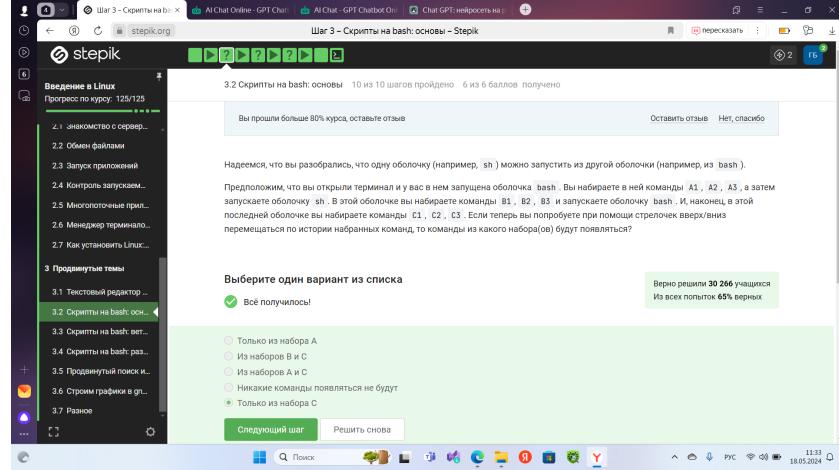


Рис. 4.6: Задание 6

Только из набора С потому что у каждой оболочки свой буфер, который при выходе из нее буде записываться в файл истории.

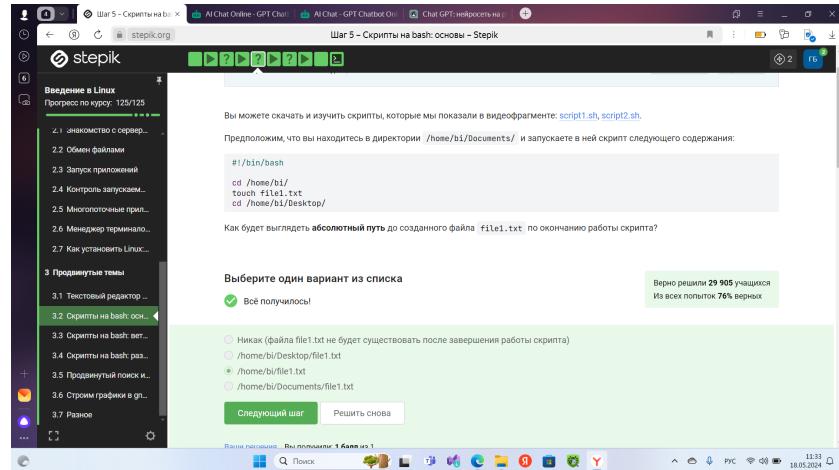


Рис. 4.7: Задание 7

`/home/bi/file1.txt` — потому что именно в этой директории мы создаем новый файл, а уже после его создания мы переходим в другую папку.

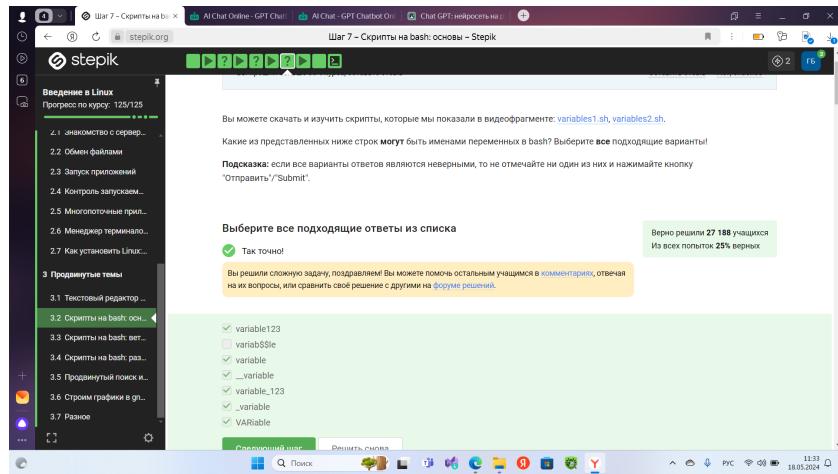


Рис. 4.8: Задание 8

Имя не может начинаться с цифры, содержать специальные символы или пробелы.

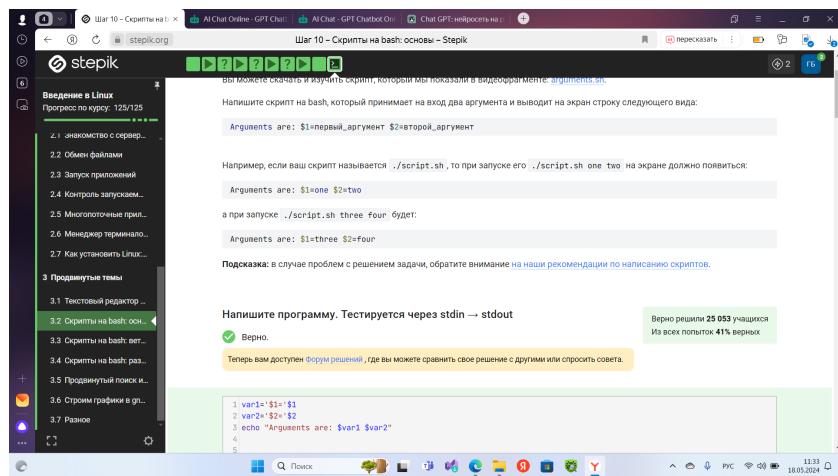


Рис. 4.9: Задание 9

\$ echo опции строки Эта команда печатает строки, которые передаются в качестве аргументов в стандартный вывод и обычно используется в сценариях оболочки для отображения сообщения или вывода результатов других команд.

`var1=$1` - обозначение переменных

`var2=$2`

```
echo "Arguments are: \$1=$var1 \$2=$var2" - строка печати.
```

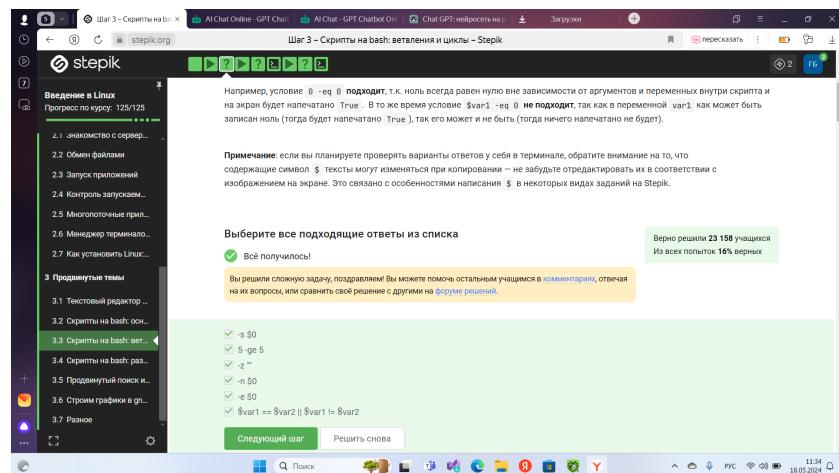


Рис. 4.10: Задание 10

- \$0 - имя скрипта
- \$# - вернет количество аргументов
- -ge - больше или равно
- -n - не пустая строка.

Имя скрипта - это не пустая строка.

#\$ Это число аргументов без учета имени скрипта, который всегда \$0. И число аргументов всегда будет или равно нулю, или больше него, тк просто не может скатиться в отрицательную сторону.

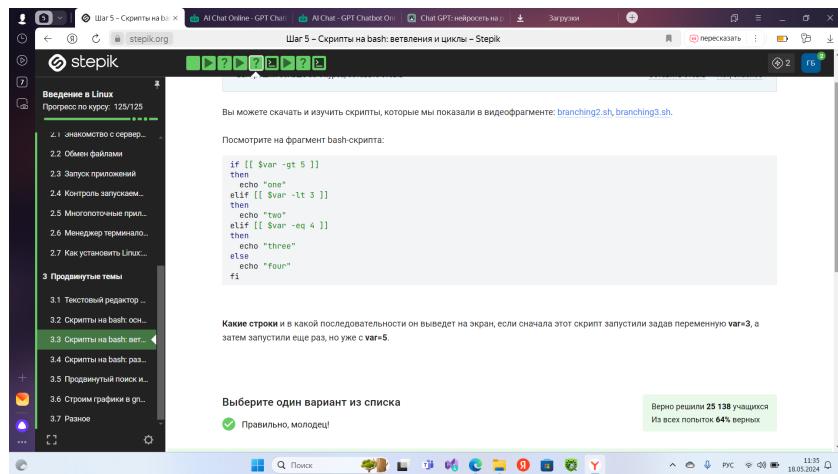


Рис. 4.11: Задание 11

- -lt, (<) - меньше
- -gt - больше
- -eq - равно

3 не больше 5, 3 не меньше 3, 3 не равно 4.

5 не больше 5, 5 не меньше 3, 5 не равно 4.

Оба раза выведет four.

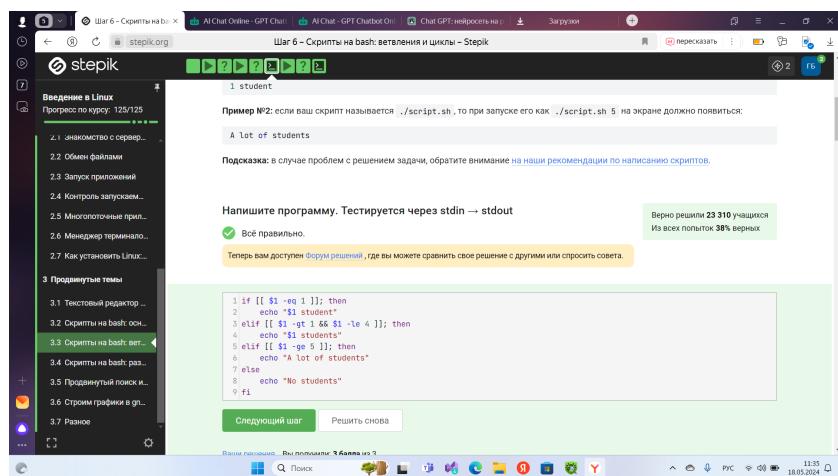


Рис. 4.12: Задание 12

1. Задаю общую часть в каждом выводе - слово “student”: v=student

2. Выполняем команды для разных аргументов.

3. res - это результат для вывода

4. echo “\$res” - вывести результат

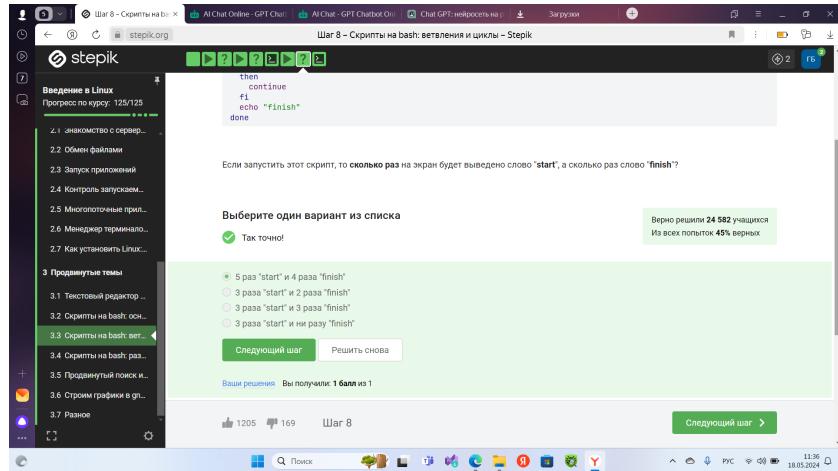


Рис. 4.13: Задание 13

- (Start)
- a > c нет (Finish)
- (Start)
- , > c нет (Finish)
- (Start)
- b > c нет (Finish)
- (Start)
- , > c нет (Finish)
- (Start)
- c_d > c да

```
#!/bin/bash
...
3 while true; do
4     echo "enter your name:"
5     read name
6     if [[ -z "$name" ]]; then
7         echo "bye"
8         break
9     fi
10    echo "enter your age:"
11    read age
12    if [ "$age" -eq 0 ]; then
13        echo "bye"
14        break
15    fi
16    if [[ "$age" -le 16 ]]; then
17        group="child"
18    elif [[ "$age" -ge 17 && "$age" -le 25 ]]; then
19        group="youth"
20    else
21        group="adult"
22    fi
23
24
25
26    echo "$name, your group is $group"
```

Рис. 4.14: Задание 14

```
#!/bin/bash
...
3 while true; do
4     echo "enter your name:"
5     read name
6     if [[ -z "$name" ]]; then
7         echo "bye"
8         break
9     fi
10    echo "enter your age:"
11    read age
12    if [ "$age" -eq 0 ]; then
13        echo "bye"
14        break
15    fi
16    if [[ "$age" -le 16 ]]; then
17        group="child"
18    elif [[ "$age" -ge 17 && "$age" -le 25 ]]; then
19        group="youth"
20    else
21        group="adult"
22    fi
23
24
25
26    echo "$name, your group is $group"
```

Рис. 4.15: Задание 14

child=16

adult=25

stdout=0

```
while [[ $stdout != 1 ]] #конструкция типа while-True
do
    echo "enter your name: " #Пользователь вводит имя
```

```

read name

if [[ (-z $name) || ($name = 0) ]] ;then #Если имя не по параметрам, простишь
    echo "bye"
    stdout=1

elif [[ -n $name ]]; then #А вот если имя нормальное
    while [[ $stdout != 1 ]] ;do
        echo "enter your age: " #То пусть вводит возраст
        read age #Считываем возраст
        if [[ ($age -eq 0) || (-z $age) ]] ;then #Если возраст 0 или строка пустая
            echo "bye"
            stdout=1

        elif [[ $age -le $child ]] ;then #Если меньше или равен ребенку, то ребенок
            echo "$name, your group is child"
        elif [[ $age -gt $adult ]] ; then #Больше взрослого - то взрослый
            echo "$name, your group is adult" ;else
                if [[ ($age -ge 17) && ($age -le 25) ]] ;then #Если от 17 до 25,
                    echo "$name, your group is youth" ;fi
                fi ;break
        done ;fi
    done
done

```

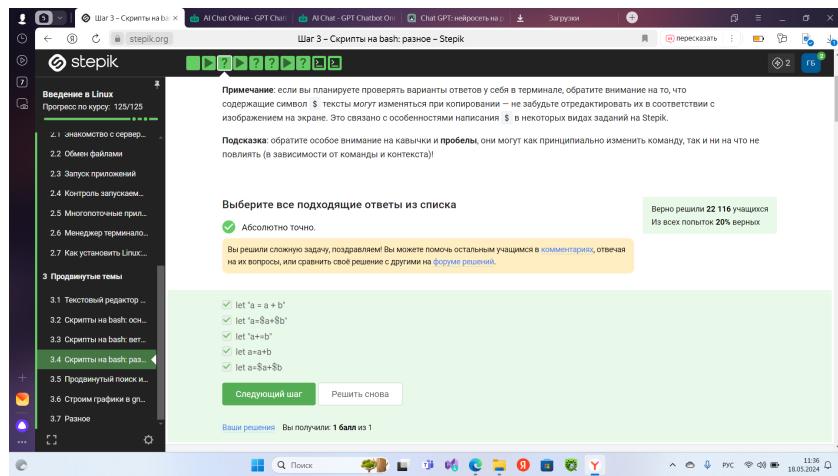


Рис. 4.16: Задание 15

1. $a = \$a$
2. $a += b$ это то же самое, что и $a = a + b$, но с символами “ $+=$ ” != “ $=+$ ”
3. если выражение не в скобках, но с пробелами - работать не будет. ($let a=a+b$ - сработает; $let a = a + b$ - нет)

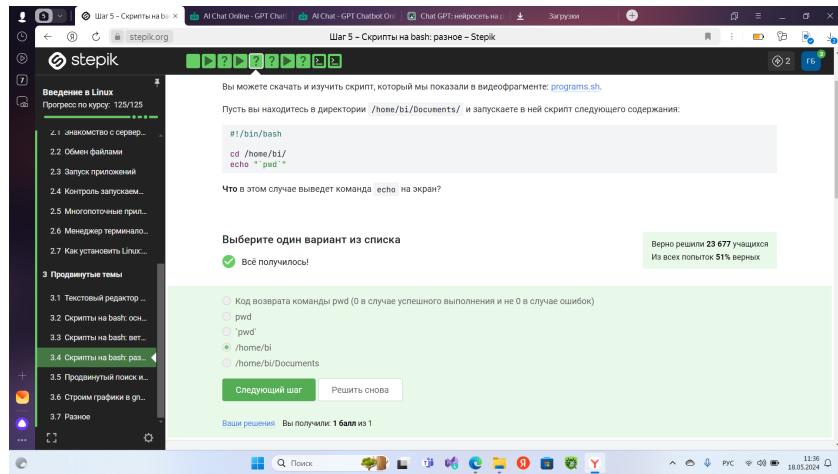


Рис. 4.17: Задание 16

Выведет путь до директории, в которую мы перешли, так как “`pwd`” - это команда

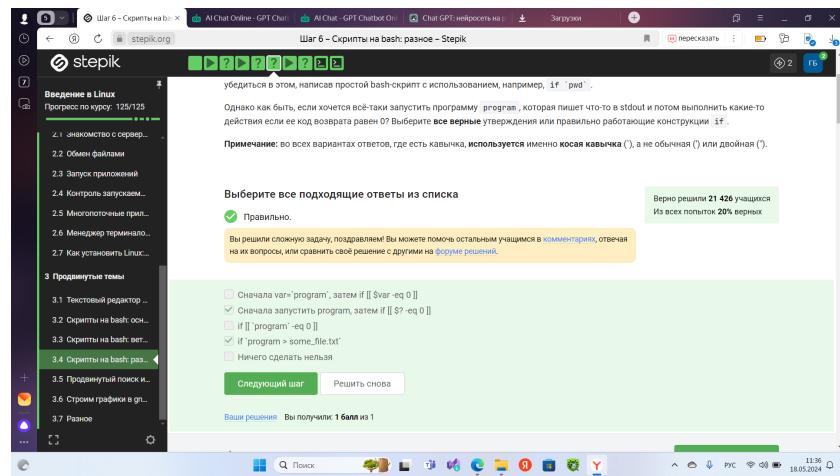


Рис. 4.18: Задание 16_2

`programm` выполняет стандартный вывод в терминал (если это принцип работы программы). И нам нужно настроить вывод в файл.

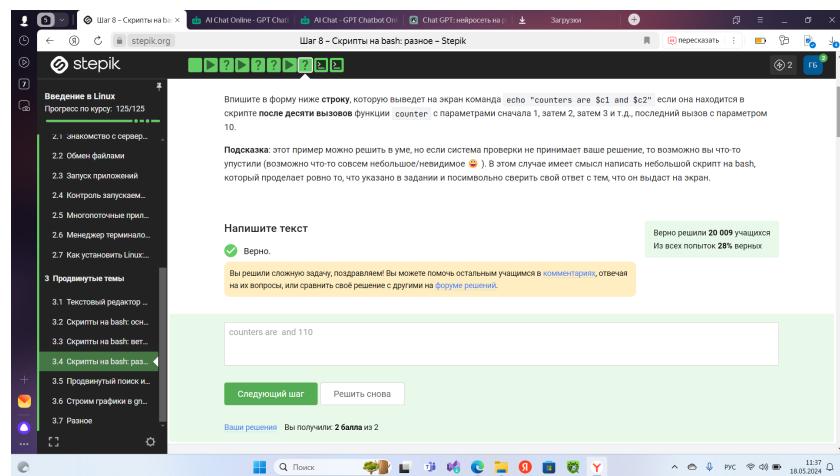


Рис. 4.19: Задание 17

Первая переменная локальная, и это просто пустая строка, вторая переменная - это сумма арифметической прогрессии от 1 до 10, равна 55, но при умножении на 2 даст 110.

The screenshot shows a computer screen with a browser window open to Stepik.org. The page title is "Шаг 9 – Скрипты на bash: разное – Stepik". On the left, there's a sidebar with a tree view of course topics under "Введение в Linux". The main area contains a code editor with the following Bash script:

```
1 #!/bin/bash
2
3 gcd () {
4     do
5         let "c=$((a%b))"
6         let "a=$b"
7         let "b=$c"
8     done
9 }
10
11 }
12
13
14 read a
15
16 while [ ! -z "$a" ]
17 do
18     gcd
19     echo "GCD is $a"
20     read a
21 done
22
23 echo "bye"
```

Below the code editor, there are two buttons: "Следующий шаг" and "Решить снова". At the bottom of the page, it says "Ваше решение: Вы получили 4 балла из 4".

Рис. 4.20: Задание 18

The screenshot shows a computer screen with a browser window open to Stepik.org. The page title is "Шаг 10 – Скрипты на bash: разное – Stepik". On the left, there's a sidebar with a tree view of course topics under "Введение в Linux". The main area contains a code editor with the following Bash script:

```
1 # put your shell (bash) code here
2 #!/bin/bash
3 while [[ $True ]]
4 do
5     read birinchil anal ikkinchi
6     if [[ $birinchil == "exit" ]]
7     then
8         echo "bye"
9         break
10    elif [[ $birinchil =~ ^[0-9]+\$ && ${#ikkinchi} =~ ^[0-9]+\$" ]]
11    then
12        echo "error"
13        break
14    else
15        case $anal in
16            "+") let "result = birinchil + ikkinchi";;
17            "-" let "result = birinchil - ikkinchi";;
18            "/") let "result = birinchil / ikkinchi";;
19            "*") let "result = birinchil * ikkinchi";;
20            "%") let "result = birinchil % ikkinchi";;
21            "**") let "result = birinchil ** ikkinchi";;
22        *) echo "error"; break ;;
23    esac
24    echo "$result"
25 fi
26 done
27
28
29
30
```

At the bottom of the page, it says "11:37 18.05.2024".

Рис. 4.21: Задание 18

Алгоритм нахождения НОД делением

1. Большее число делим на меньшее.
2. Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).
3. Если есть остаток, то большее число заменяем на остаток от деления.
4. Переходим к пункту 1.

```
1 # put your shell (bash) code here
2 #!/bin/bash
3 while [[ $True ]]
4 do
5     read birinchi anal ikkinchi
6     if [[ $birinchi == "exit" ]]
7         then
8             echo "bye"
9             break
10    elif [[ "$birinchi" =~ ^[0-9]+\$ && "$ikkinchi" =~ ^[0-9]+\$ ]]
11        then
12            echo "error"
13            break
14        else
15            case $anal in
16                "+") let "result = birinchi + ikkinchi";;
17                "-") let "result = birinchi - ikkinchi";;
18                "/") let "result = birinchi / ikkinchi";;
19                "*") let "result = birinchi * ikkinchi";;
20                "%") let "result = birinchi % ikkinchi";;
21                "**") let "result = birinchi ** ikkinchi";;
22            *) echo "error"; break ;;
23        esac
24        echo "$result"
25    fi
26 done
27
28
29
30
```

Рис. 4.22: Задание 19

```
1 # put your shell (bash) code here
2 #!/bin/bash
3 while [[ $True ]]
4 do
5     read birinchi anal ikkinchi
6     if [[ $birinchi == "exit" ]]
7         then
8             echo "bye"
9             break
10    elif [[ "$birinchi" =~ ^[0-9]+\$ && "$ikkinchi" =~ ^[0-9]+\$ ]]
11        then
12            echo "error"
13            break
14        else
15            case $anal in
16                "+") let "result = birinchi + ikkinchi";;
17                "-") let "result = birinchi - ikkinchi";;
18                "/") let "result = birinchi / ikkinchi";;
19                "*") let "result = birinchi * ikkinchi";;
20                "%") let "result = birinchi % ikkinchi";;
21                "**") let "result = birinchi ** ikkinchi";;
22            *) echo "error"; break ;;
23        esac
24        echo "$result"
25    fi
26 done
27
28
29
30
```

Рис. 4.23: Задание 19

Калькулятор выглядит обычно - мы вводим два числа, пишем, что с ними надо сделать, и потом, учитывая случаи ошибок, выводим результат.

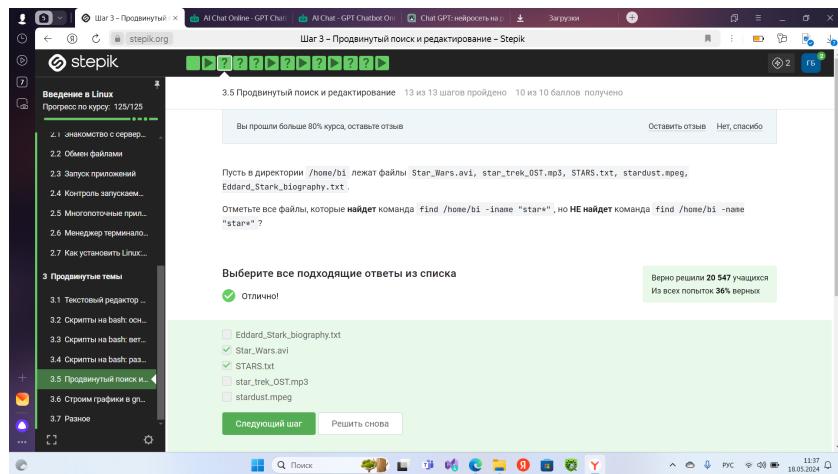


Рис. 4.24: Задание 20

`-iname` ищет без учета регистра, а `-name` в точности как в запросе. Звездочка стоит после слова – это значит после слова может быть сколько угодно символов.

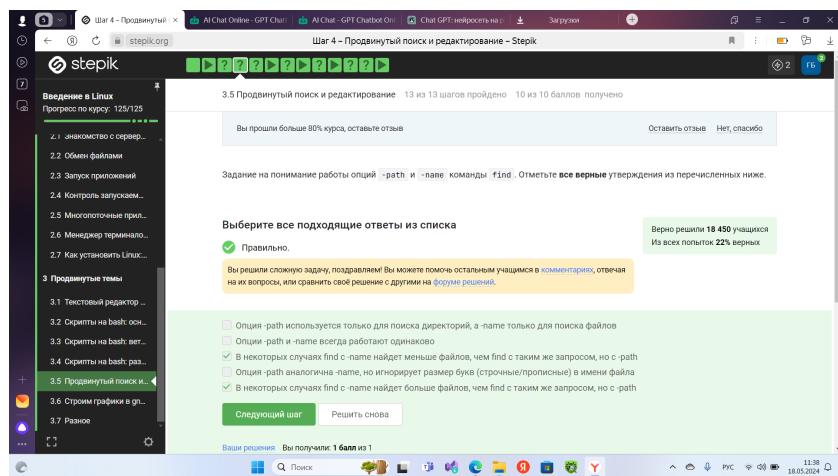


Рис. 4.25: Задание 21

`find [path] [expression]`

где: `path` – это путь к директории, в которой нужно выполнить поиск файлов (по умолчанию, поиск производится в текущей директории и всех ее поддиректориях);

`expression` – это выражение, которое определяет критерии поиска файлов.

-name: поиск файлов по имени. Например: find /home/user -name myfile.txt

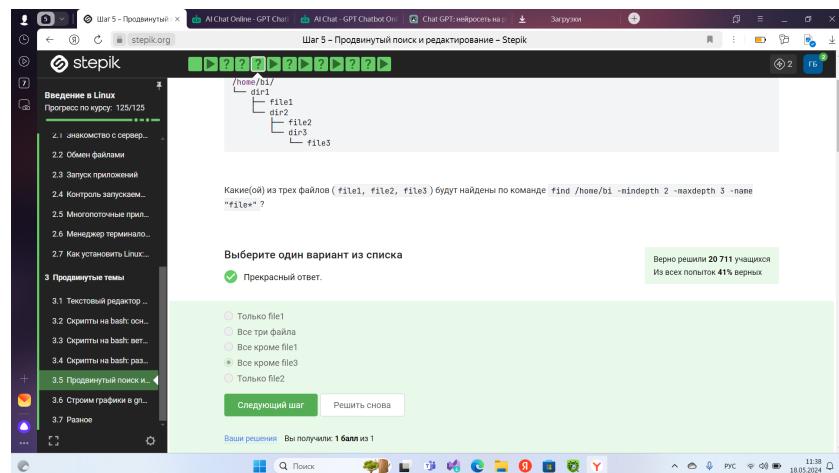


Рис. 4.26: Задание 22

Текущий каталог - это depth=1, а остальное считается просто:

/home/bi -> depth=1

/home/bi/dir1 -> depth=2

/home/bi/dir1/dir2 -> depth=3

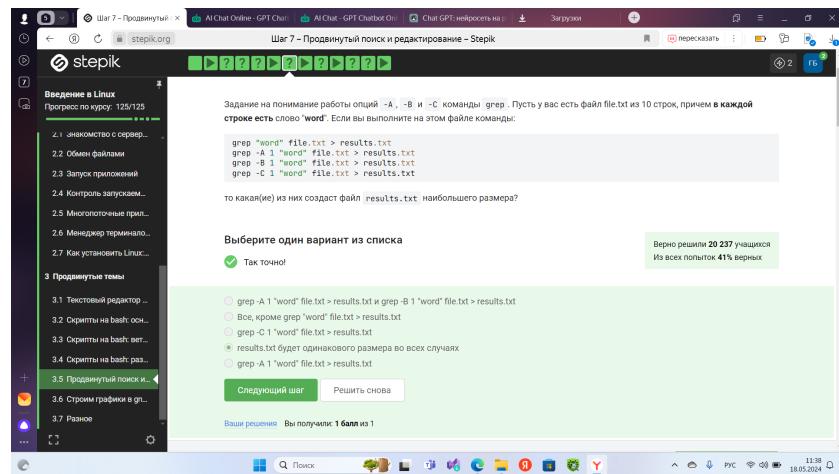


Рис. 4.27: Задание 23

Из описания man: Print NUM lines of trailing context after/before matching lines
“matching lines” - множественное число, строки в которых нашлось совпадение

Т.е. если идут 2...10...100 строк подряд, в которых обнаружилось совпадение, контекст будет выведен до и после этой ГРУППЫ строк, а не до и после каждой строки в этой группе

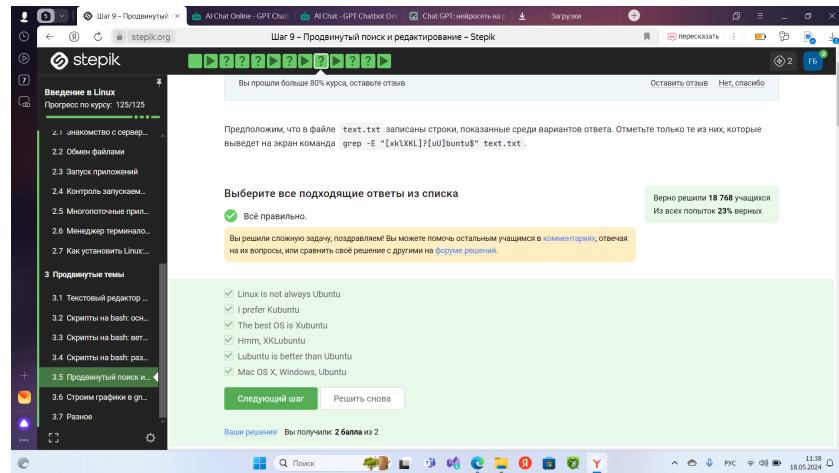


Рис. 4.28: Задание 24

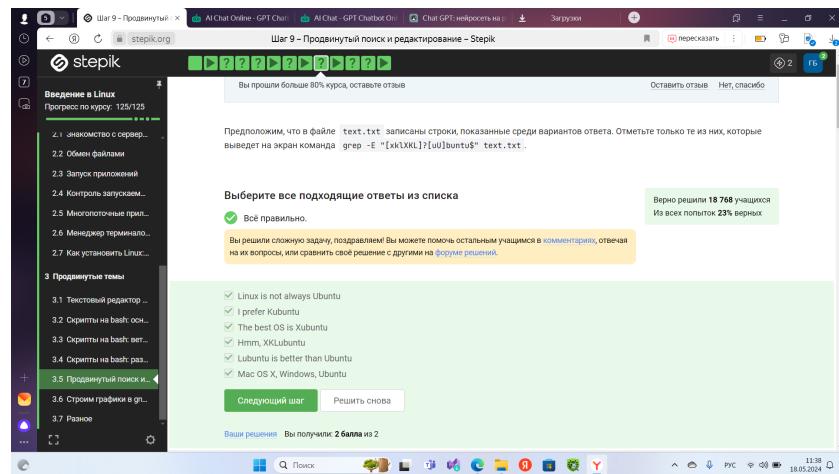


Рис. 4.29: Задание 24

Объяснение на втором скриншоте.

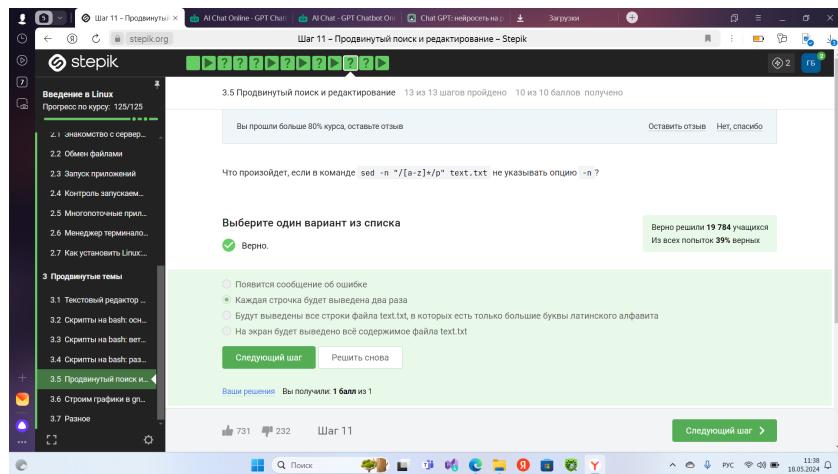


Рис. 4.30: Задание 25

The `-n` option disables the automatic printing, which means the lines you don't specifically tell it to print do not get printed, and lines you do explicitly tell it to print (e.g. with `p`) get printed only once.

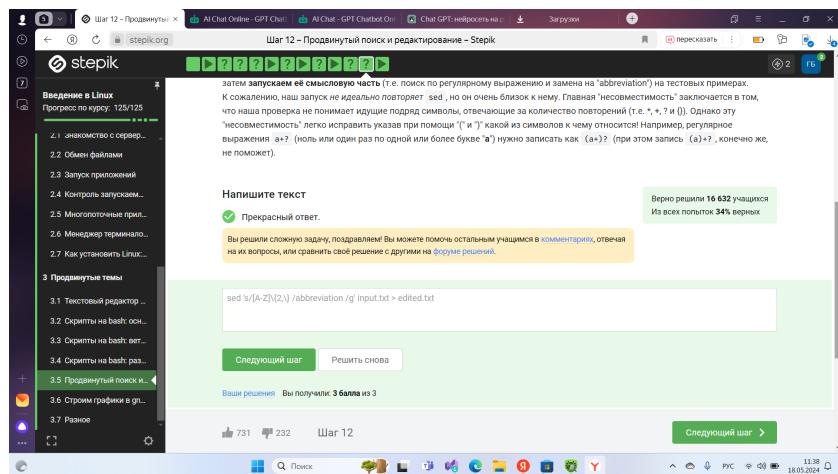


Рис. 4.31: Задание 26

аббревиатура АВВА отличается от двух других аббревиатур тем, что справа он неё стоит запятая без пробела: “АВВА,”.

При этом по условию аббревиатура должна выглядеть как [XX] или [XXX] (и ещё больше X). Следовательно, для этой проверки надо добавить пробел

квадратными скобками [] слева и, соответственно, с права.

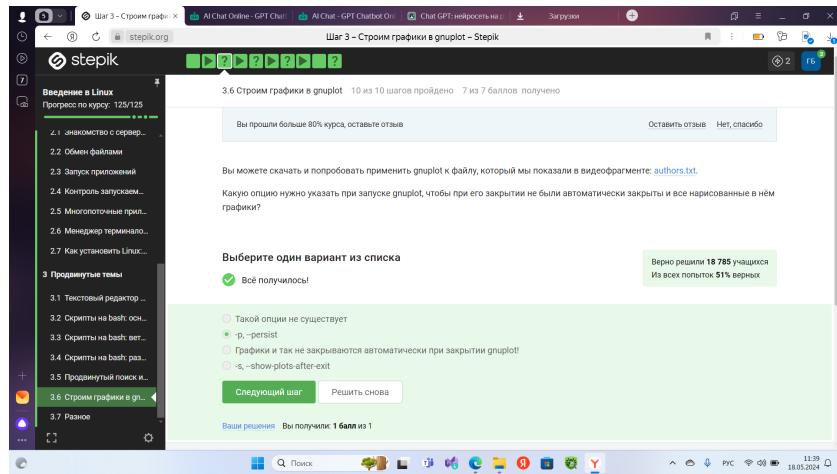


Рис. 4.32: Задание 27

-persist lets plot windows survive after main gnuplot program exits.

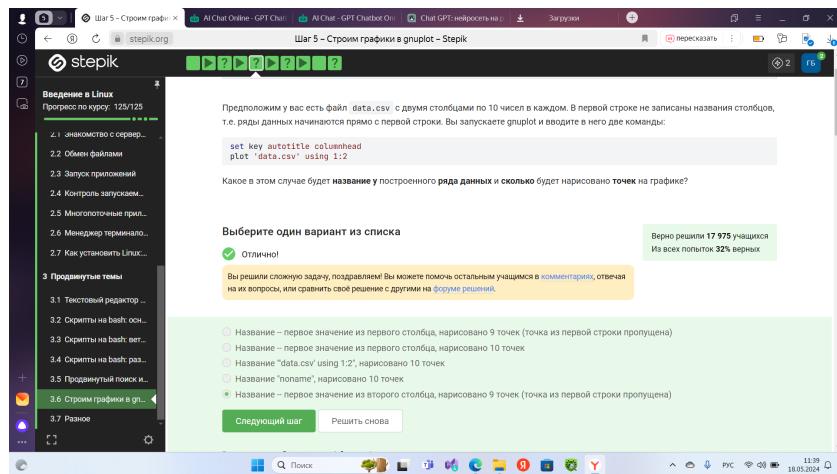


Рис. 4.33: Задание 28

plot 'data.csv' using 1:2 даст ошибку:

```
warning: Skipping data file with no valid points ^ x range is  
invalid
```

Скорее всего причиной такого поведения является тот факт, что формат CSV содержит строки, где столбцы разделены запятой? Содержимое файла:

1, 21
2, 22
3, 23
4, 24
5, 25
6, 26
7, 27
8, 28
9, 29
10, 30

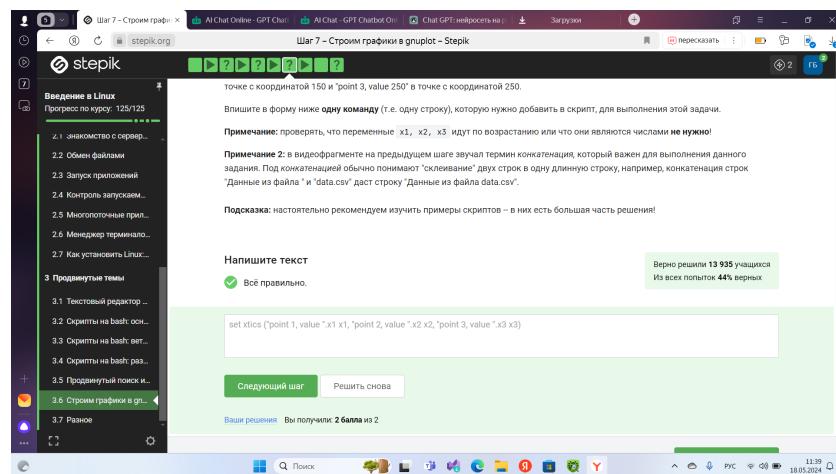


Рис. 4.34: Задание 29

Сначала идет команда установки подписей, а потом в скобках:
подпись - пробел - переменная с координатой - запятая
Повторяется это количество раз соответствующее числу переменных, и без запятой (в случае с последней переменной)

А подпись в свою очередь получается конкатенацией текста из задания и переменной с координатой.

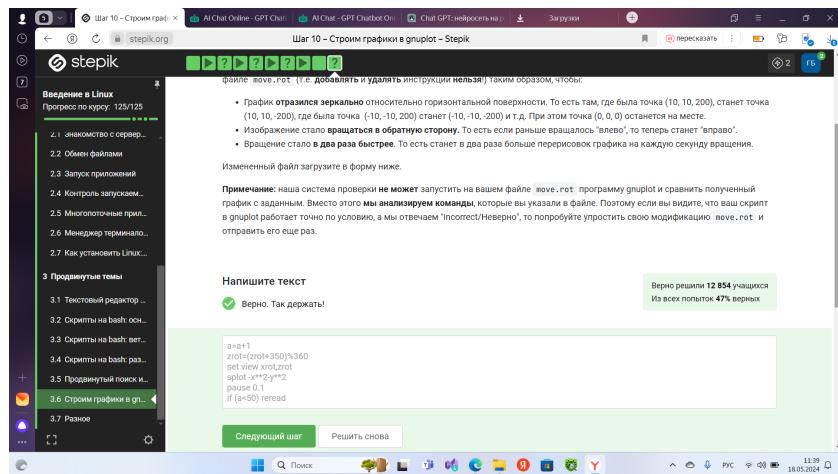


Рис. 4.35: Задание 30

- График строится строкой “`splot x2+y2`”.
- Вращение задается строкой “`zrot=(zrot+10)%360`”. Значит, смещение вперед (которое было изначально) можно также задать строкой “`zrot=(zrot+360+10)%360`” или иначе говоря “`zrot=(zrot+370)%360`”. А теперь посмотрим на наше требование - чтоб вращалось в другую сторону, значит, по аналогии, необходимо вместо перебора на 10 сделать недобор.

“`zrot=(zrot+350)%360`”

- Строка “`pause 0.2`” ставит выполнение на паузу на определенный промежуток времени. В задании сказали перерисовывать чаще, значит пауза должна быть меньше.

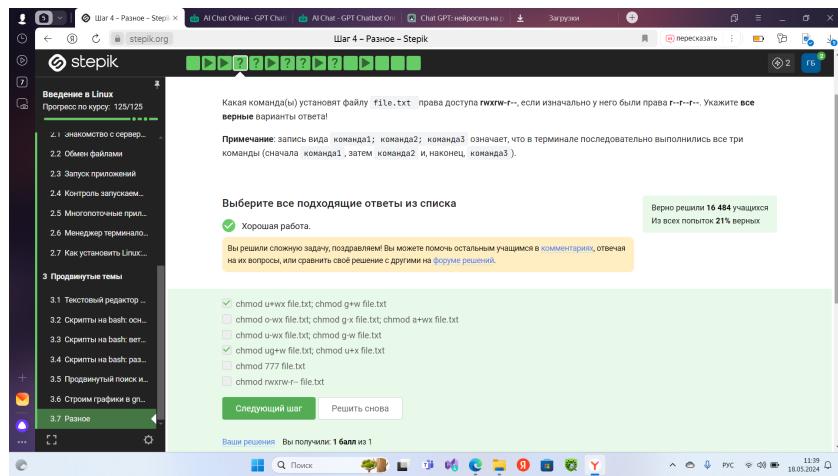


Рис. 4.36: Задание 31

- r - чтение;
- w - запись;
- x - выполнение;
- s - выполнение от имени суперпользователя (дополнительный);
- u - владелец файла;
- g - группа файла;
- o - все остальные пользователи;
- 0 - никаких прав;
- 1 - только выполнение;
- 2 - только запись;
- 3 - выполнение и запись;
- 4 - только чтение;
- 5 - чтение и выполнение;

- 6 - чтение и запись;
- 7 - чтение запись и выполнение.

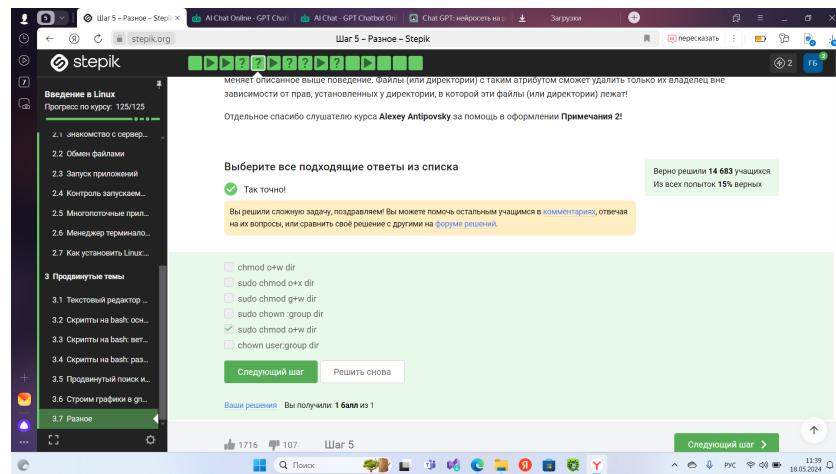


Рис. 4.37: Задание 32

Решений два типа:

- Сменить права гостей, добавив W
- Сделать владельцем нужную группу или пользователя, в зависимости от того, у кого из них уже есть права на W
- Помнить, что root - владелец и остальные для него - others.

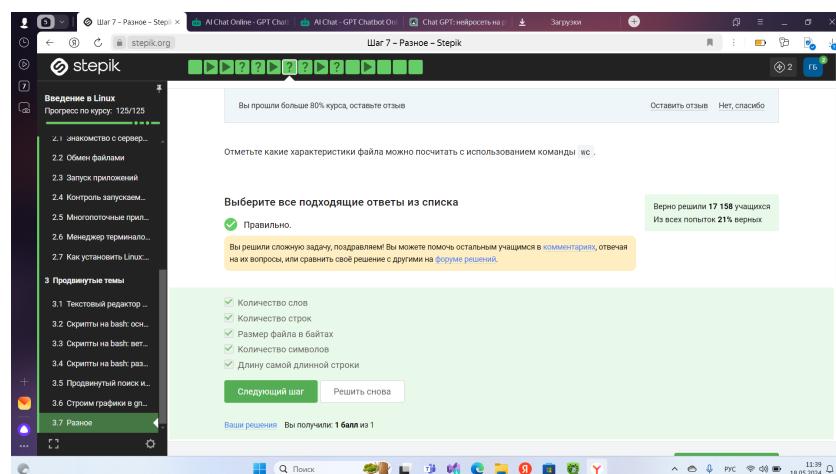


Рис. 4.38: Задание 33

- **wc -l** вывести количество строк
- **wc -c** вывести количество байт
- **wc -m** вывести количество символов
- **wc -L** вывести длину самой длинной строки
- **wc -w** вывести количество слов

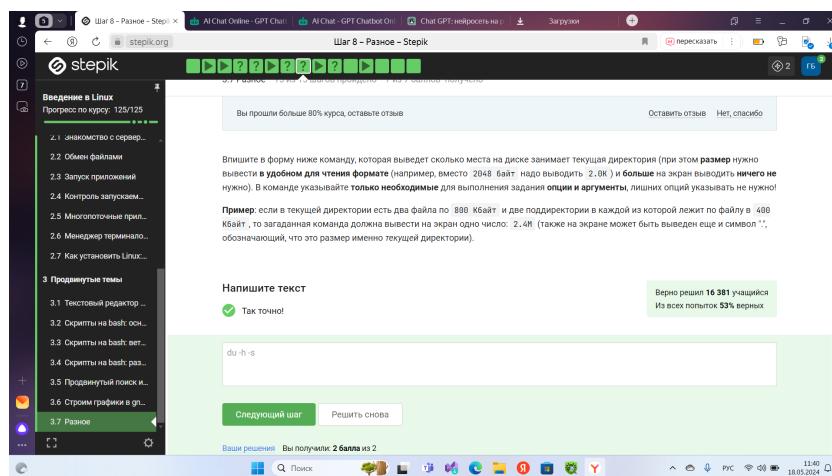


Рис. 4.39: Задание 34

-h, --human-readable print sizes in human readable format (e.g., 1K 234M 2G)
-s, --summarize display only a total for each argument

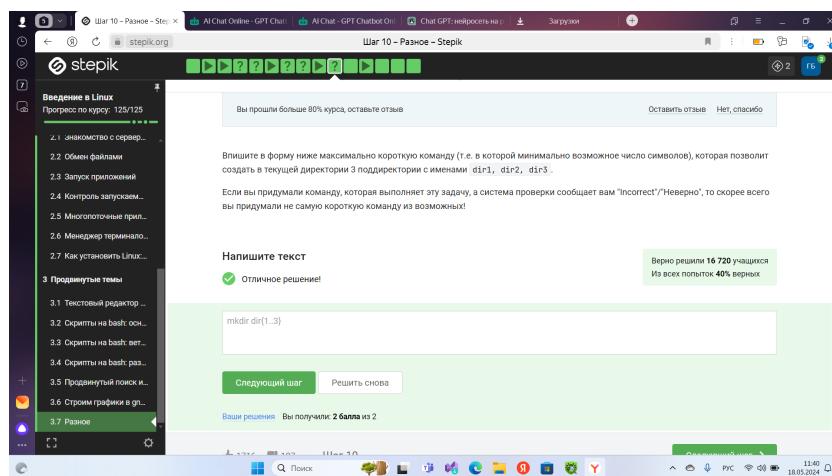


Рис. 4.40: Задание 35

Команда создаст три директории от dir1 до dir3.

5 Сертификат

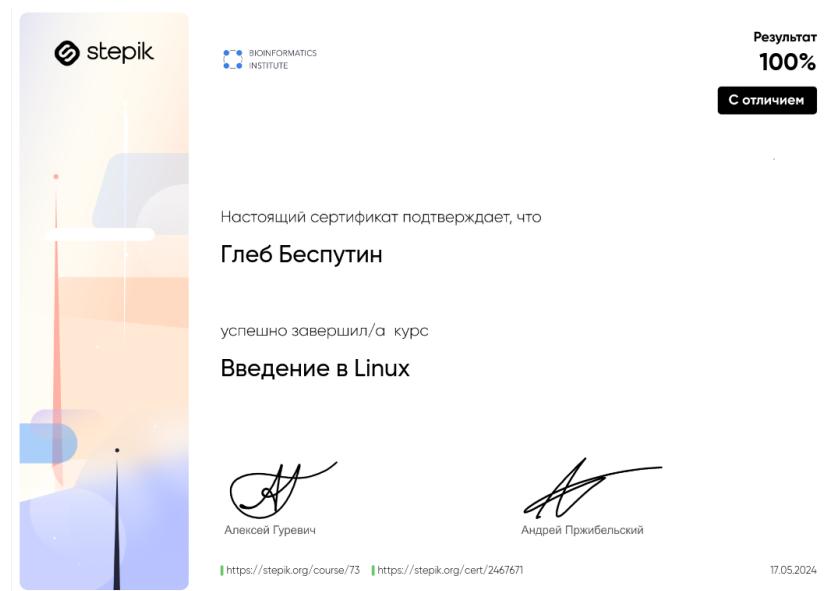


Рис. 5.1: Сертификат

6 Выводы

Я просмотрел курс и освежила в памяти навыки работы с более сложными командами в Линукс.

Список литературы

1. Введение в Linux