

Отчет о прохождении внешних курсов

3 этап

Кижваткина Анна Юрьевна

Содержание

1	Цель работы	6
2	Задание	7
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
5	Сертификат	27
6	Выводы	28

Список иллюстраций

4.1 Задание 1	9
4.2 Задание 2	10
4.3 Задание 3	10
4.4 Задание 4	11
4.5 Задание 5	11
4.6 Задание 6	12
4.7 Задание 7	12
4.8 Задание 9	13
4.9 Задание 10	13
4.10 Задание 11	14
4.11 Задание 12	15
4.12 Задание 13	15
4.13 Задание 14	16
4.14 Задание 14	16
4.15 Задание 15	16
4.16 Задание 16	17
4.17 Задание 17	17
4.18 Задание 18	18
4.19 Задание 18	18
4.20 Задание 19	19
4.21 Задание 19	19
4.22 Задание 20	19
4.23 Задание 21	20
4.24 Задание 22	20
4.25 Задание 23	21
4.26 Задание 24	21
4.27 Задание 24	21
4.28 Задание 25	22
4.29 Задание 26	22
4.30 Задание 27	22
4.31 Задание 28	22
4.32 Задание 29	22
4.33 Задание 30	23
4.34 Задание 31	23
4.35 Задание 33	25
4.36 Задание 34	25
4.37 Задание 35	26

5.1 Сертификат	27
--------------------------	----

Список таблиц

1 Цель работы

Ознакомиться с функционалом операционной системы Linux.

2 Задание

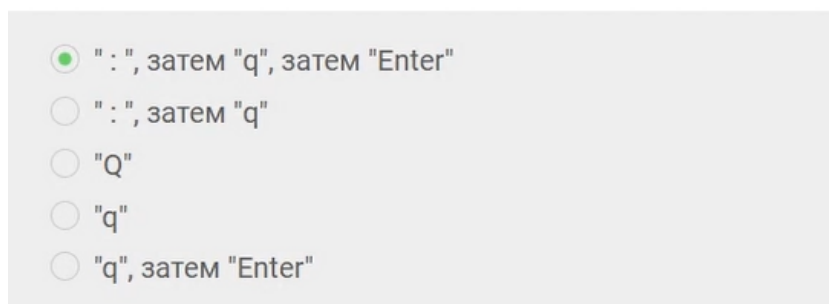
Просмотреть видео и на основе полученной информации пройти тестовые задания.

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. fig. 4.1, fig. 4.2, fig. 4.3, fig. 4.4, fig. 4.5, fig. 4.6, fig. 4.7, fig. ??, fig. 4.8, fig. 4.9, fig. 4.10, fig. 4.11, fig. 4.12, fig. 4.13, fig. 4.14, fig. 4.15, fig. 4.16, fig. ??, fig. 4.17, fig. 4.18, fig. 4.19, fig. 4.20, fig. 4.21, fig. 4.22, fig. 4.23, fig. 4.24, fig. 4.25, fig. 4.26, fig. 4.27, fig. 4.28, fig. 4.29, fig. 4.30, fig. 4.31, fig. 4.32, fig. 4.33, fig. 4.34, fig. ??, fig. 4.35, fig. 4.36, fig. 4.37, fig. 5.1).



☒ ": ", затем "q", затем "Enter"

☐ ": ", затем "q"

☐ "Q"

☐ "q"

☐ "q", затем "Enter"

Рис. 4.1: Задание 1

Стоит упомянуть, что у редактора vim есть туториал, 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 подойдет под конкретную ситуацию.

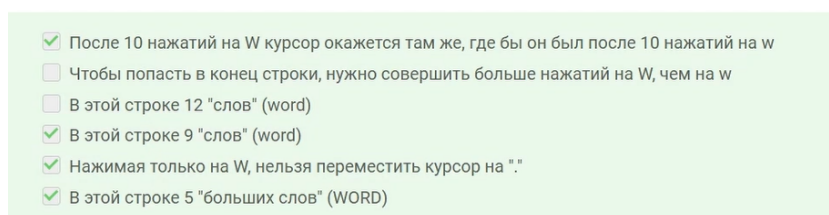


Рис. 4.2: Задание 2

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,

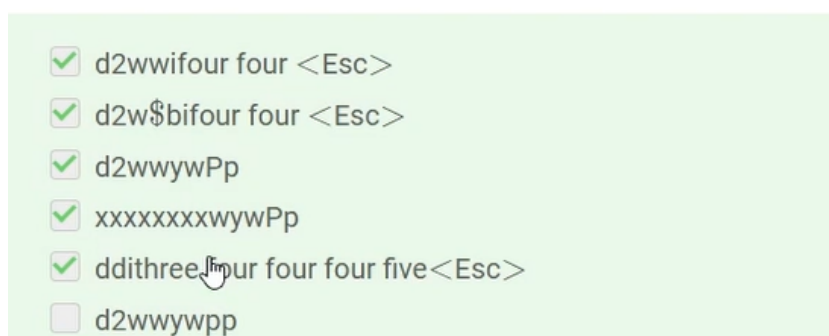


Рис. 4.3: Задание 3

- \$ — 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;
- уу (также Y) — копирование текущей строки в неименованный буфер;
- уу — 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 %.

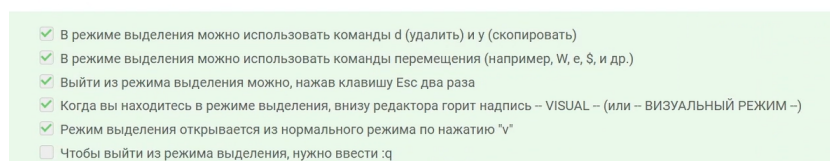


Рис. 4.5: Задание 5

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

Нажать Esc достаточно один раз, но да ладно.

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;

☐ Никакие команды появляться не будут
☐ Только из набора А
☐ Из наборов В и С
☒ Только из набора С
☐ Только из набора В

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.

☒ /home/bi/file1.txt
☐ Никак (файла file1.txt не будет существовать после завершения работы скрипта)
☐ /home/bi/Desktop/file1.txt
☐ /home/bi/Documents/file1.txt

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.

☒ _variable
☒ VARiable
☒ variable_123
☒ __variable
☒ variable123
☒ variable
☐ var.i.able

Следующий шаг Решить снова

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

```
1 #!/bin/bash
2 var1=$1
3 var2=$2
4
5 echo "Arguments are: \$1=$var1 \$2=$var2"
6
7
8
9
10
```

Рис. 4.8: Задание 9

\$ echo опции строка 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.

☒ -s \$0

☒ \$# -ge 0

☒ -e \$0

☒ !(4 -le 3)

☒ -n \$0

☒ \$var1 == \$var2 || \$var1 != \$var2

Следующий шаг

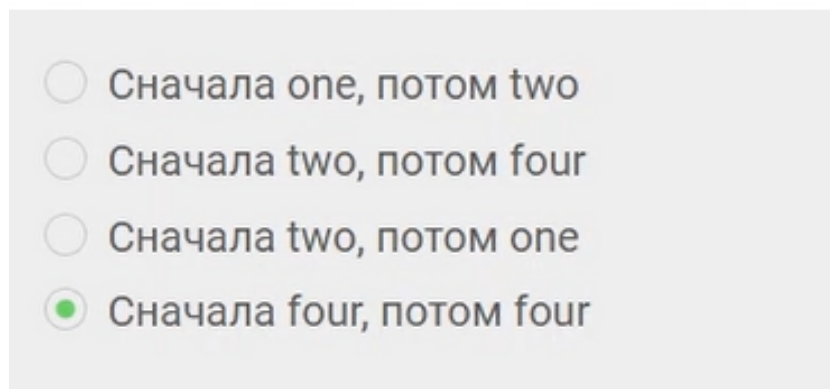
Решить снова

Рис. 4.9: Задание 10

- \$0 - имя скрипта

- \$# - вернет количество аргументов
- -ge - больше или равно
- -n - 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.



☐ Сначала one, потом two

☐ Сначала two, потом four

☐ Сначала two, потом one

☒ Сначала four, потом four

Рис. 4.10: Задание 11

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

Оба раза выведет four.

```

1 #!/bin/bush
2 v=student
3 case $1 in
4 0) res="No ${v}s";;
5 1) res="1 ${v}";;
6 [2-4]) res="$1 ${v}s";;
7 *) res="A lot of ${v}s";;
8 esac
9 echo "$res"
10
11

```

Рис. 4.11: 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

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

Рис. 4.12: Задание 13

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).

Рис. 4.13: Задание 14

```

1 child=16
2 adult=25
3 stdout=8
4
5 while [[ $stdout != 1 ]]
6 do
7     echo "enter your name: "
8     read name
9     if [[ (-z $name) || ($name = 0) ]];then
10        echo "bye"
11        stdout=1
12    elif [[ -n $name ]]; then
13        while [[ $stdout != 1 ]];do
14            echo "enter your age: "
15            read age
16            if [[ ($age -eq 0) || (-z $age) ]];then
17                echo "bye"
18                stdout=1
19            elif [[ $age -le $child ]];then
20                echo "$name, your group is child"
21            elif [[ $age -gt $adult ]];then
22                echo "$name, your group is adult" ;else
23                if [[ ($age -ge 17) && ($age -le 25) ]];then
24                    echo "$name, your group is youth" ;fi
25            fi ;break
26        done ;fi
27 done

```

Рис. 4.14: Задание 14

Пишем код исходя из условий задания

☒ let a=\$a+\$b
 ☒ let a=a+b
 ☒ let "a+=b"
 ☐ let a = a + b
 ☐ a=\$a+\$b

Рис. 4.15: Задание 15

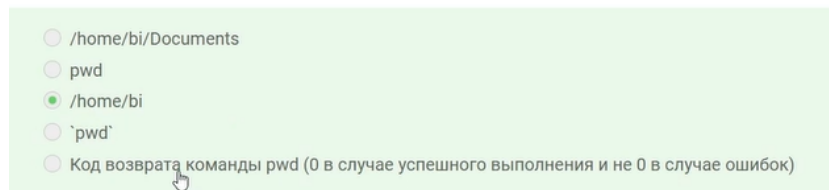
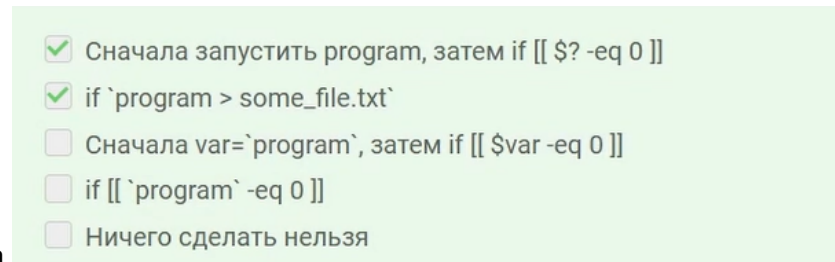


Рис. 4.16: Задание 16

Выведет путь до директории, в которую мы перешли, так как ”
pwd



” - это команда

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

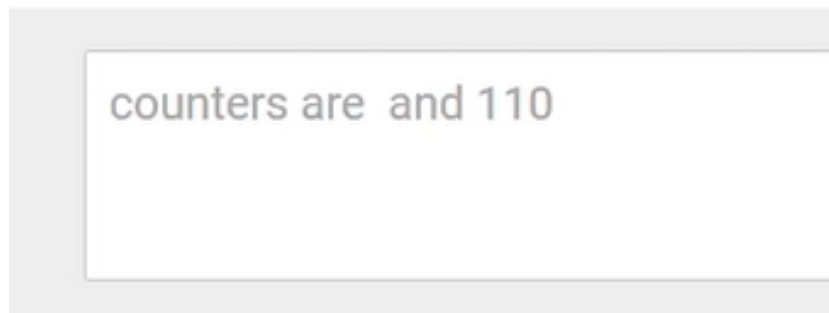


Рис. 4.17: Задание 17

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.

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)**.

Рис. 4.18: Задание 18

```
1 while ( true )
2 do
3     read n1 n2
4     if [ -z $n1 ]; then
5         echo "bye"
6         break
7     else
8         gcd () {
9             remainder=1
10            if [ $2 -eq 0 ]
11            then
12                echo "bye"
13            fi
14            while [ $remainder -ne 0 ]
15            do
16                remainder=$((n1%n2))
17                n1=$n2
18                n2=$remainder
19            done
20        }
21        gcd $1 $2
22        echo "GCD is $n1"
23    fi done
```

Рис. 4.19: Задание 18

Алгоритм нахождения НОД делением

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.

Рис. 4.20: Задание 19

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

Рис. 4.21: Задание 19

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

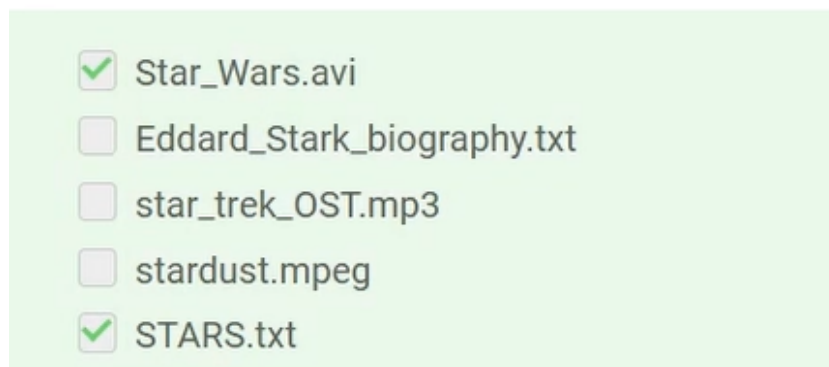


Рис. 4.22: Задание 20

-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.

- ☐ Опция -path используется только для поиска директорий, а -name только для поиска файлов
- ☒ Если заменить в команде поиска -name, на -path, то результат поиска иногда может остаться таким же
- ☐ Опция -path аналогична -name, но игнорирует размер букв (строчные/прописные) в имени файла
- ☒ В некоторых случаях find с -name найдет больше файлов, чем find с таким же запросом, но с -path
- ☒ В некоторых случаях find с -name найдет меньше файлов, чем find с таким же запросом, но с -path

Рис. 4.23: 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`

- ☐ Только file1
- ☒ Все кроме file3
- ☐ Все кроме file2
- ☐ Все три файла
- ☐ Ни один файл найден не будет

Рис. 4.24: 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`

- ☐ grep -A 1 "word" file.txt > results.txt и grep -B 1 "word" file.txt > results.txt
- ☐ grep -A 1 "word" file.txt > results.txt
- ☐ Все, кроме grep "word" file.txt > results.txt
- ☐ grep -C 1 "word" file.txt > results.txt
- ☒ results.txt будет одинакового размера во всех случаях

Рис. 4.25: 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

- ☒ I prefer Kubuntu
- ☒ Hmm, XKLubuntu
- ☒ Mac OS X, Windows, Ubuntu
- ☒ Lubuntu is better than Ubuntu
- ☒ Linux is not always Ubuntu
- ☐ Uuuubuntu!

Рис. 4.26: Задание 24

```
Linux is not always Ubuntu
[tsganina@fedora ~]$ grep -E "[xklXKL]?[uU]buntu$" text.txt
I prefer Kubuntu
The best OS is Xubuntu
Lubuntu is better than Ubuntu
Mac OS X, Windows, Ubuntu
[tsganina@fedora ~]$ gedit text.txt
[tsganina@fedora ~]$ grep -E "[xklXKL]?[uU]buntu$" text.txt
I prefer Kubuntu
Linux is not always Ubuntu
```

Рис. 4.27: Задание 24

Объяснение на втором скриншоте.

- ☐ Появится сообщение об ошибке
- ☒ Каждая строчка будет выведена два раза
- ☐ Будут выведены все строки файла text.txt, в которых есть только большие буквы латинского алфавита
- ☐ На экран ничего не напечатается

Рис. 4.28: Задание 25

```
sed 's/[A-Z]\{2,\} /abbreviation /g' input.txt > edited.txt
```

Рис. 4.29: Задание 26

- ☐ Такой опции не существует
- ☐ Графики и так не закрываются автоматически при закрытии gnuplot!
- ☐ -raise
- ☒ -p, -persist

Рис. 4.30: Задание 27

- ☐ Название – первое значение из первого столбца, нарисовано 9 точек (точка из первой строки пропущена)
- ☒ Название – первое значение из второго столбца, нарисовано 9 точек (точка из первой строки пропущена)
- ☐ Название – первое значение из второго столбца, нарисовано 10 точек
- ☐ Название "data.csv" using 1:2, нарисовано 10 точек
- ☐ Название "no name", нарисовано 10 точек

Рис. 4.31: Задание 28

```
set xtics ("point 1, value ".x1 x1, "point 2, value ".x2 x2, "point 3, value ".x3 x3)
```

Рис. 4.32: 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.

```
a=a+1
zrot=(zrot+350)%360
set view xrot,zrot
splot -x**2-y**2
pause 0.1
if (a<50) reread
```

Рис. 4.33: Задание 30

1. График строится строкой “splot x2+y2”.
2. 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”

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

```
☐ chmod o-wx file.txt; chmod g-x file.txt; chmod a+wx file.txt
☐ chmod 777 file.txt
☒ chmod a+wx file.txt; chmod o-wx file.txt; chmod g-x file.txt
☐ chmod 467 file.txt
☐ chmod u-wx file.txt; chmod g-w file.txt
☒ chmod 764 file.txt
```

Рис. 4.34: 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.

- ☒ sudo chmod o+w dir
- ☒ sudo chown user dir
- ☐ sudo chown :group dir
- ☒ sudo chmod a+w dir
- ☒ sudo chown user:group dir
- ☐ chown user:group dir

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.

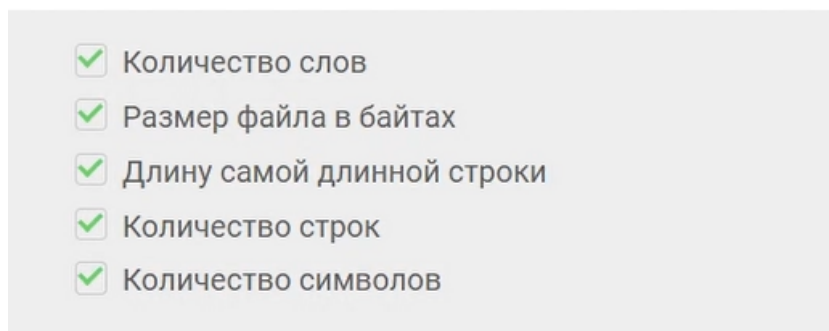


Рис. 4.35: Задание 33

- `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

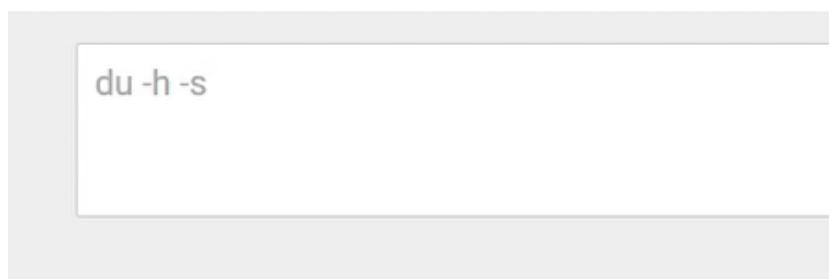


Рис. 4.36: Задание 34

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

```
mkdir dir{1..2}
```

I

Рис. 4.37: Задание 35

Команда создаст три директории от dir1 до dir3.

5 Сертификат

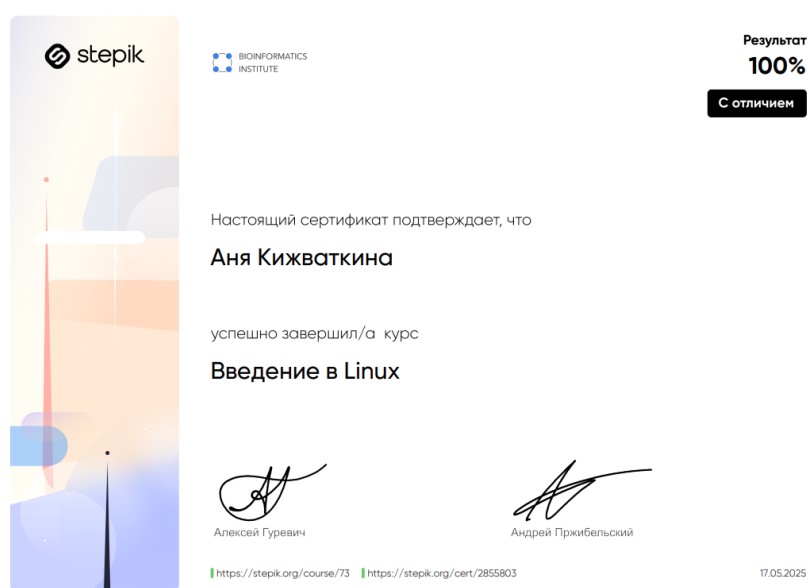


Рис. 5.1: Сертификат

6 Выводы

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