

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Лабораторная работа № 4

по дисциплине “Linux”

“Создание и использование сценариев (скриптов) в Linux”

Студент

Болдырев М.Р

(подпись, дата)

Группа АС-21-1

Руководитель

доцент

Кургасов В.В

(подпись, дата)

Липецк 2023

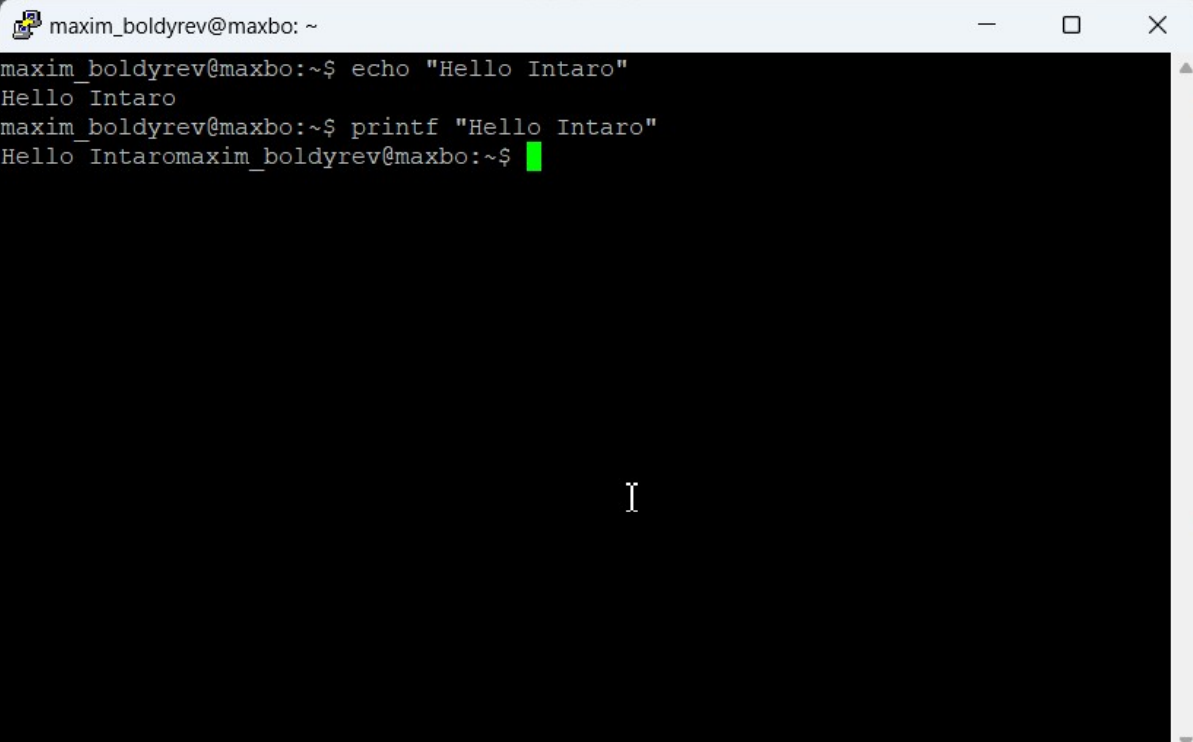
Цели работы

- Изучить основные возможности языка программирования высокого уровня Shell;
- Получить навыки написания и использования скриптов.

Порядок выполнения работы

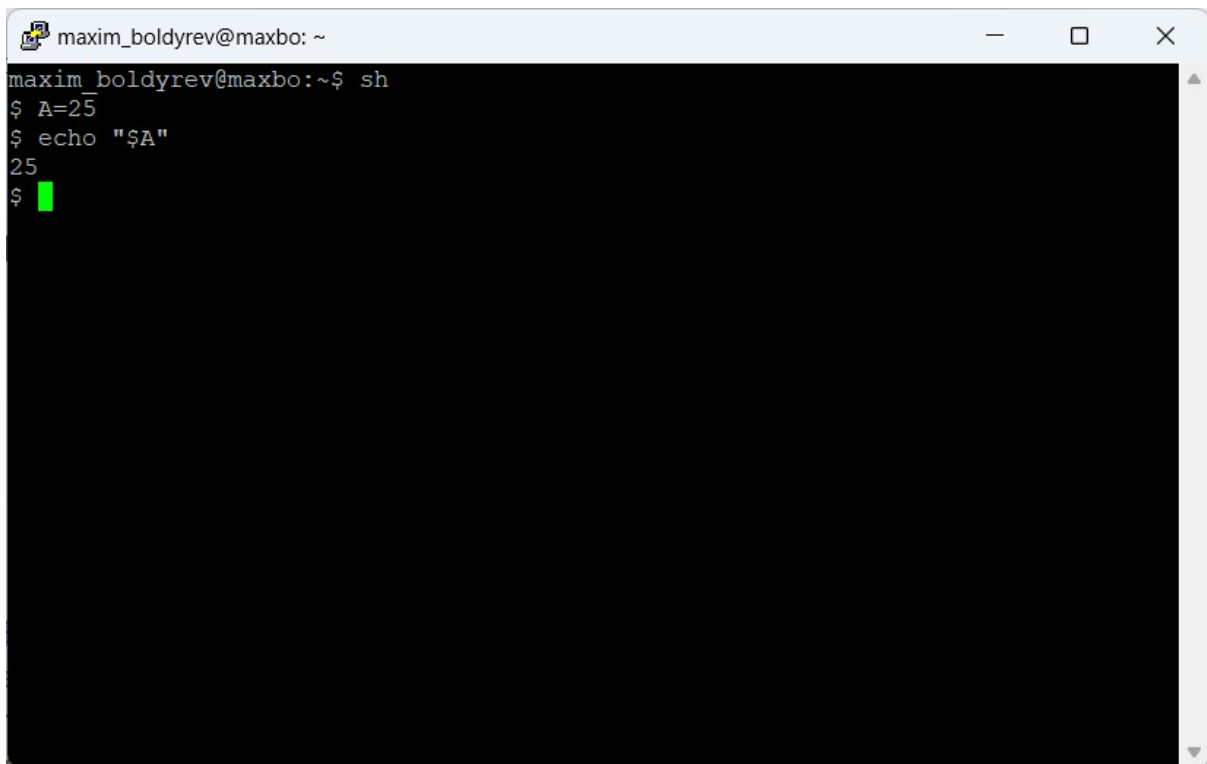
1 часть

1. Выведем информационные сообщения с помощью echo и printf



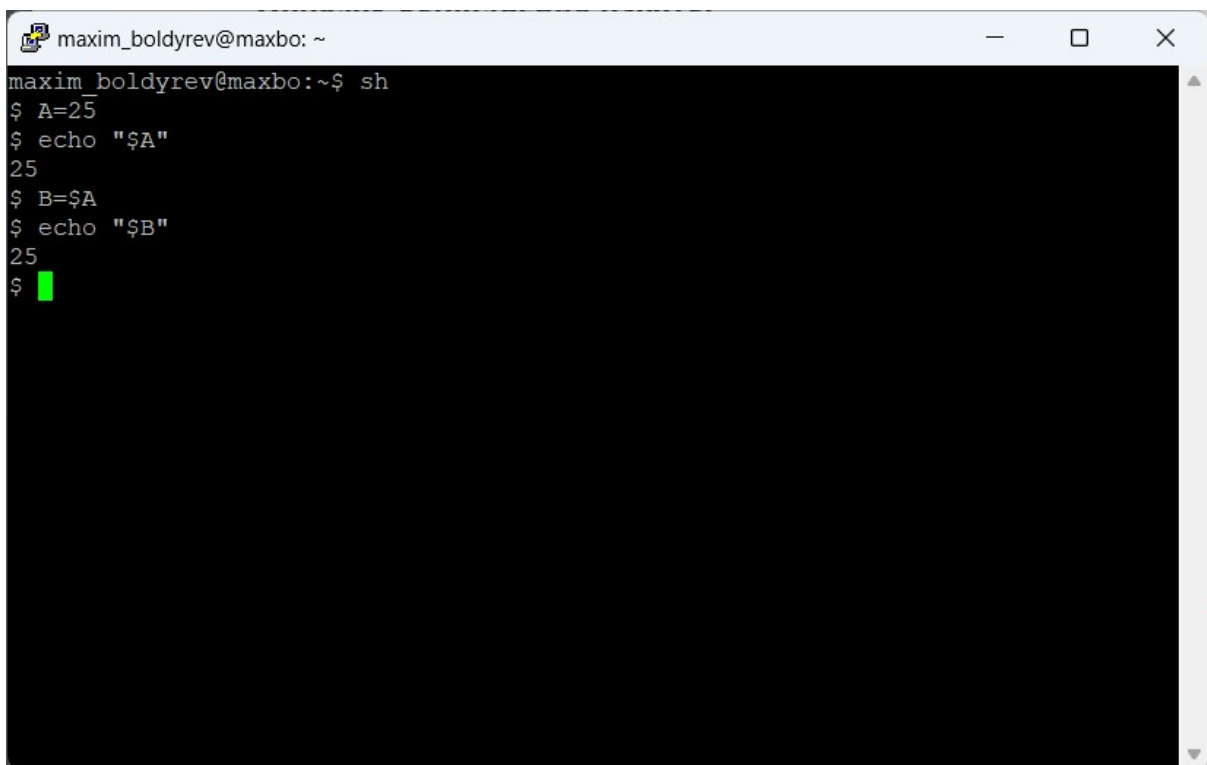
```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ echo "Hello Intaro"  
Hello Intaro  
maxim_boldyrev@maxbo:~$ printf "Hello Intaro"  
Hello Intaromaxim_boldyrev@maxbo:~$
```

2. Присвоим переменной *A* целочисленное значение и выведем её в терминал:



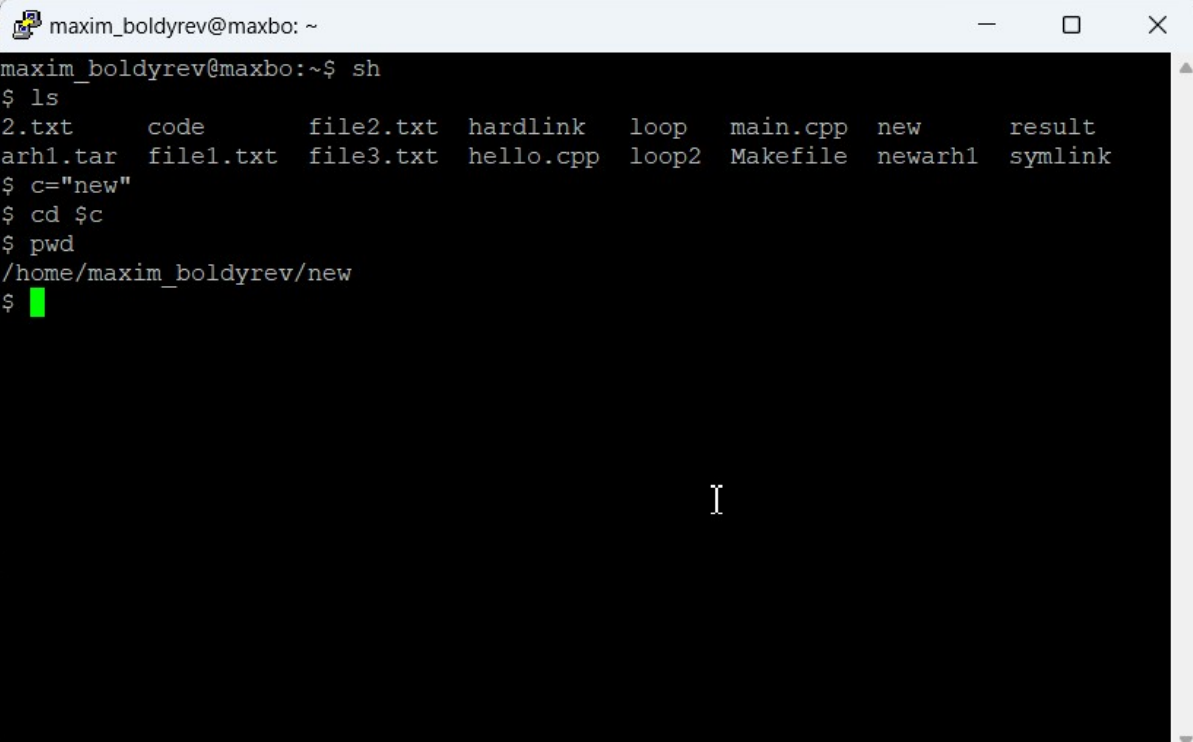
```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ sh  
$ A=25  
$ echo "$A"  
25  
$
```

3. Присвоить переменной *B* значение переменной *A*. Посмотреть значение переменной *B*.



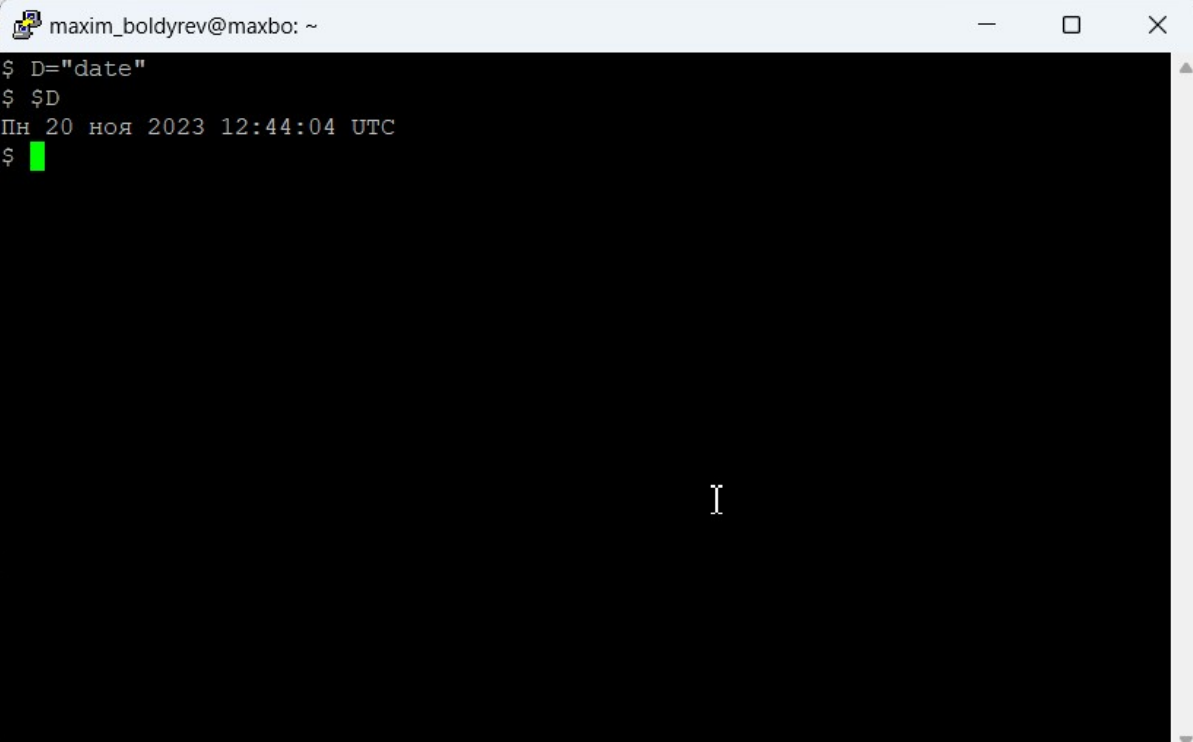
```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ sh  
$ A=25  
$ echo "$A"  
25  
$ B=$A  
$ echo "$B"  
25  
$
```

4. Присвоить переменной *C* значение «ПутьДоСвоегоКаталога». Перейти в этот каталог с использованием переменной.

A terminal window titled 'maxim_boldyrev@maxbo: ~' with standard window controls. The terminal shows a shell prompt '\$ sh' followed by a series of commands: '\$ ls' which lists files in two rows, '\$ c="new"', '\$ cd \$c', and '\$ pwd' which outputs '/home/maxim_boldyrev/new'. The prompt '\$' is followed by a green cursor.

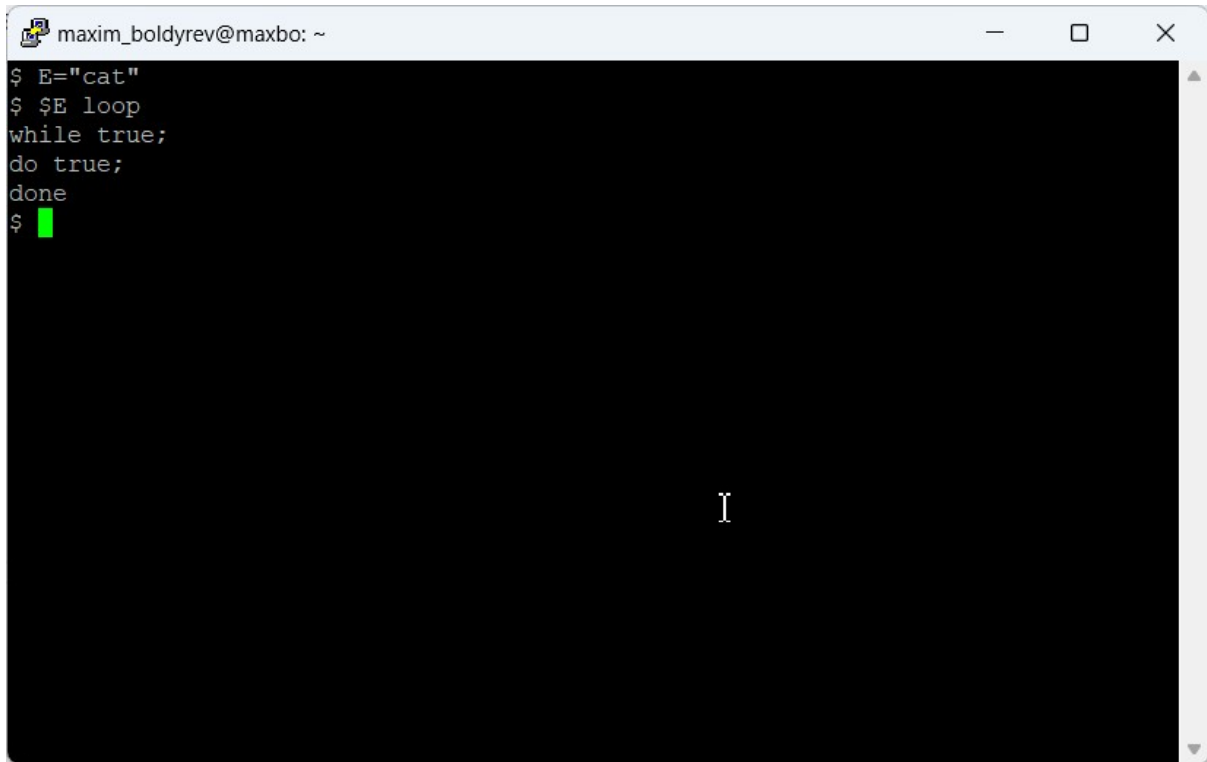
```
maxim_boldyrev@maxbo: ~$ sh
$ ls
2.txt      code      file2.txt  hardlink  loop     main.cpp  new      result
arh1.tar   file1.txt file3.txt  hello.cpp loop2    Makefile  newarh1  symlink
$ c="new"
$ cd $c
$ pwd
/home/maxim_boldyrev/new
$
```

5. Присвоить переменной *D* значение «ИмяКоманды», а именно, команды DATE. Выполнить эту команду, используя значение переменной.

A terminal window titled 'maxim_boldyrev@maxbo: ~' with standard window controls. The terminal shows a shell prompt '\$' followed by the commands '\$ D="date"', '\$ \$D', and the output 'Пн 20 ноя 2023 12:44:04 UTC'. The prompt '\$' is followed by a green cursor.

```
maxim_boldyrev@maxbo: ~$
$ D="date"
$ $D
Пн 20 ноя 2023 12:44:04 UTC
$
```

6. Присвоить переменной *E* значение «ИмяКоманды», а именно, команды просмотра содержимого текстового файла. Выполнить эту команду, используя значение переменной.

A terminal window with a light blue title bar containing the text 'maxim_boldyrev@maxbo: ~' and standard window controls. The terminal has a black background with white text. It shows a shell script snippet: '\$ E="cat"', '\$ \$E loop', 'while true;', 'do true;', 'done', and '\$' followed by a green cursor. A vertical scrollbar is on the right side.

```
maxim_boldyrev@maxbo: ~  
$ E="cat"  
$ $E loop  
while true;  
do true;  
done  
$
```

7. Присвоить переменной *F* значение «ИмяКоманды», а именно, сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.

```
maxim_boldyrev@maxbo: ~  
$ cat > file.txt  
abrikos  
sos  
pobeda  
dima  
molodec  
^C  
$ F="sort"  
$ $F file.txt  
abrikos  
dima  
molodec  
pobeda  
sos  
$ █
```

Далее нам предстоит написать скрипты, при запуске которых будут происходить действия, указанные в заданиях. Для удобства я буду писать и выполнять скрипты из файла script.sh. Результат выполнения каждого пункта будет в формате код - результат выполнения.

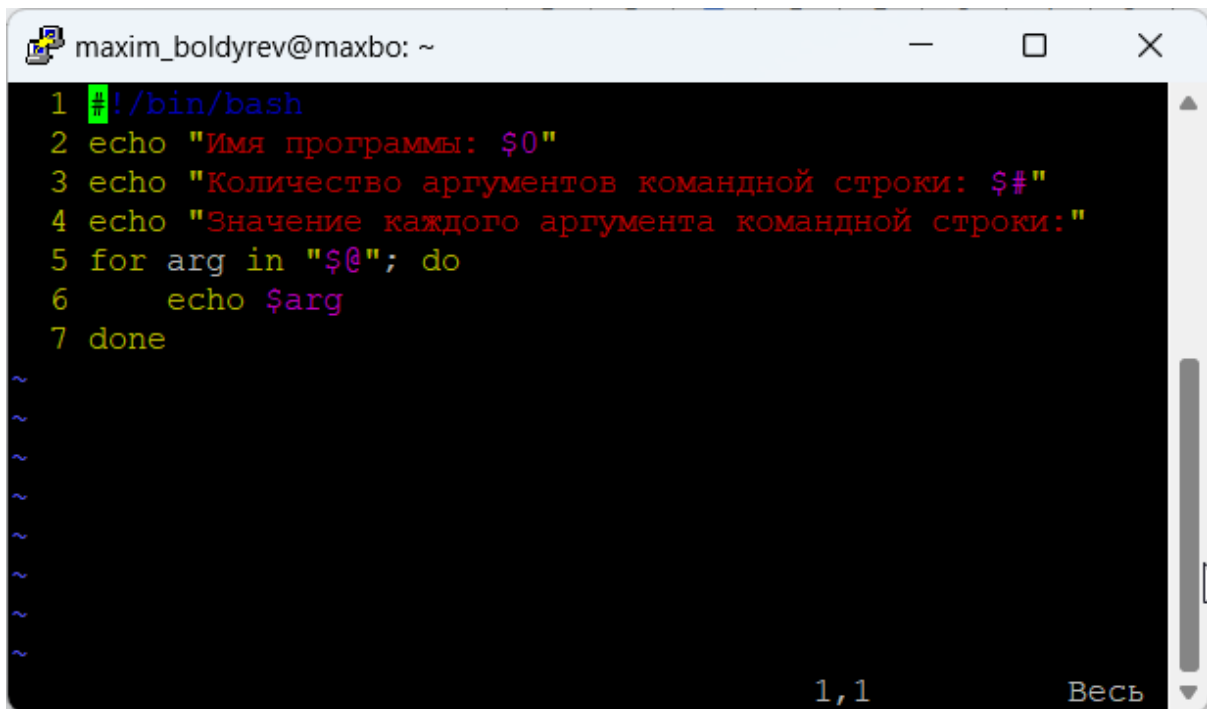
1. Программа запрашивает значение переменной, а затем выводит значение этой переменной


```
maxim_boldyrev@maxbo: ~  
1 #!/bin/bash  
2 # Запрос значений двух переменных  
3 echo "Введите значение первой переменной:"  
4 read var1  
5 echo "Введите значение второй переменной:"  
6 read var2  
7  
8 # Вычисление произведения с использованием expr  
9 product_expr=$(expr $var1 \* $var2)  
10 echo "Произведение с использованием expr: $product_expr"  
11  
12 # Вычисление произведения с использованием bc  
13 product_bc=$(echo "$var1 * $var2" | bc)  
14 echo "Произведение с использованием bc: $product_bc"  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
"script.sh" 14L, 652B 1,1 Весь
```

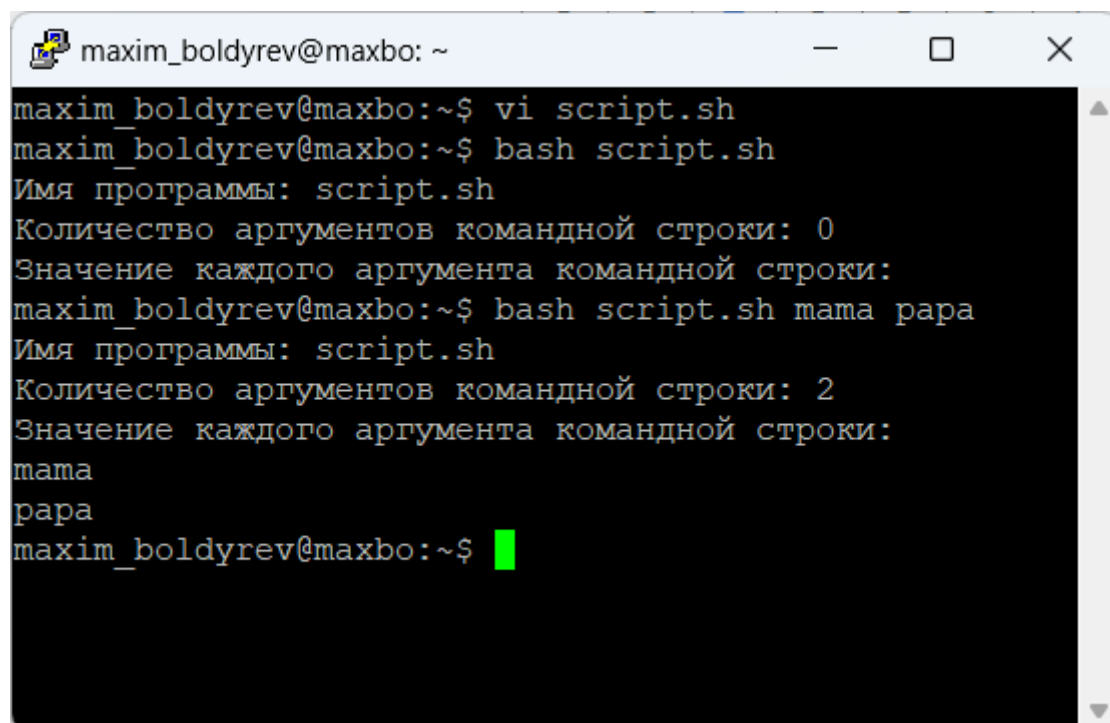
```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ bash script.sh  
Введите значение первой переменной:  
2  
Введите значение второй переменной:  
3  
Произведение с использованием expr: 6  
Произведение с использованием bc: 6  
maxim_boldyrev@maxbo:~$
```

4. Вычисление объема цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран.

5. Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки.

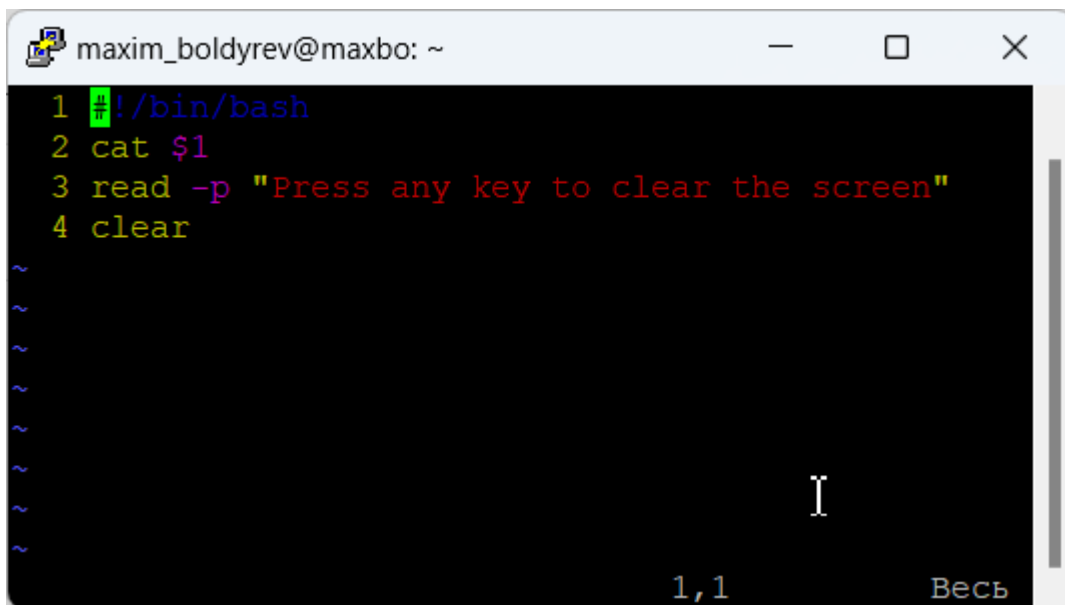


```
maxim_boldyrev@maxbo: ~  
1 #!/bin/bash  
2 echo "Имя программы: $0"  
3 echo "Количество аргументов командной строки: $#"  
4 echo "Значение каждого аргумента командной строки:"  
5 for arg in "$@"; do  
6     echo $arg  
7 done  
~  
~  
~  
~  
~  
~  
1,1  Весь
```



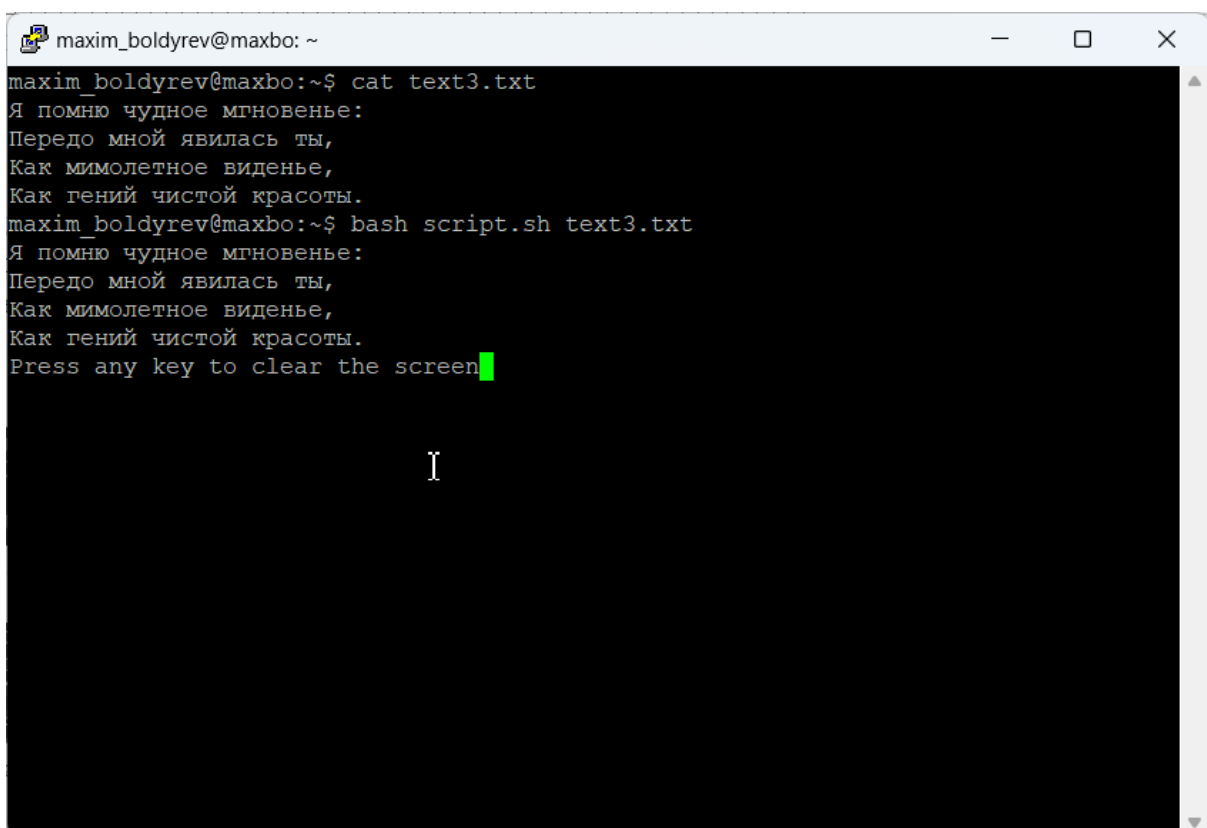
```
maxim_boldyrev@maxbo: ~$ vi script.sh  
maxim_boldyrev@maxbo:~$ bash script.sh  
Имя программы: script.sh  
Количество аргументов командной строки: 0  
Значение каждого аргумента командной строки:  
maxim_boldyrev@maxbo:~$ bash script.sh мама папа  
Имя программы: script.sh  
Количество аргументов командной строки: 2  
Значение каждого аргумента командной строки:  
мама  
папа  
maxim_boldyrev@maxbo:~$
```

6. Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.



```
maxim_boldyrev@maxbo: ~  
1 #!/bin/bash  
2 cat $1  
3 read -p "Press any key to clear the screen"  
4 clear  
~  
~  
~  
~  
~  
~  
~  
~  
1,1      Весь
```

The image shows a terminal window titled 'maxim_boldyrev@maxbo: ~'. It contains a script with four lines: a shebang, a command to cat a file, a prompt for a key press, and a clear command. The terminal shows several tilde characters (~) representing previous command history. At the bottom, it shows '1,1' and 'Весь'.



```
maxim_boldyrev@maxbo: ~$ cat text3.txt  
Я помню чудное мгновенье:  
Передо мной явилась ты,  
Как мимолетное виденье,  
Как гений чистой красоты.  
maxim_boldyrev@maxbo: ~$ bash script.sh text3.txt  
Я помню чудное мгновенье:  
Передо мной явилась ты,  
Как мимолетное виденье,  
Как гений чистой красоты.  
Press any key to clear the screen  
I
```

The image shows a terminal window titled 'maxim_boldyrev@maxbo: ~'. It shows the execution of a script. First, 'cat text3.txt' is run, displaying a poem. Then, 'bash script.sh text3.txt' is run, which displays the same poem and then a prompt 'Press any key to clear the screen'. A cursor is visible on the line below the prompt.

7. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога построчно.

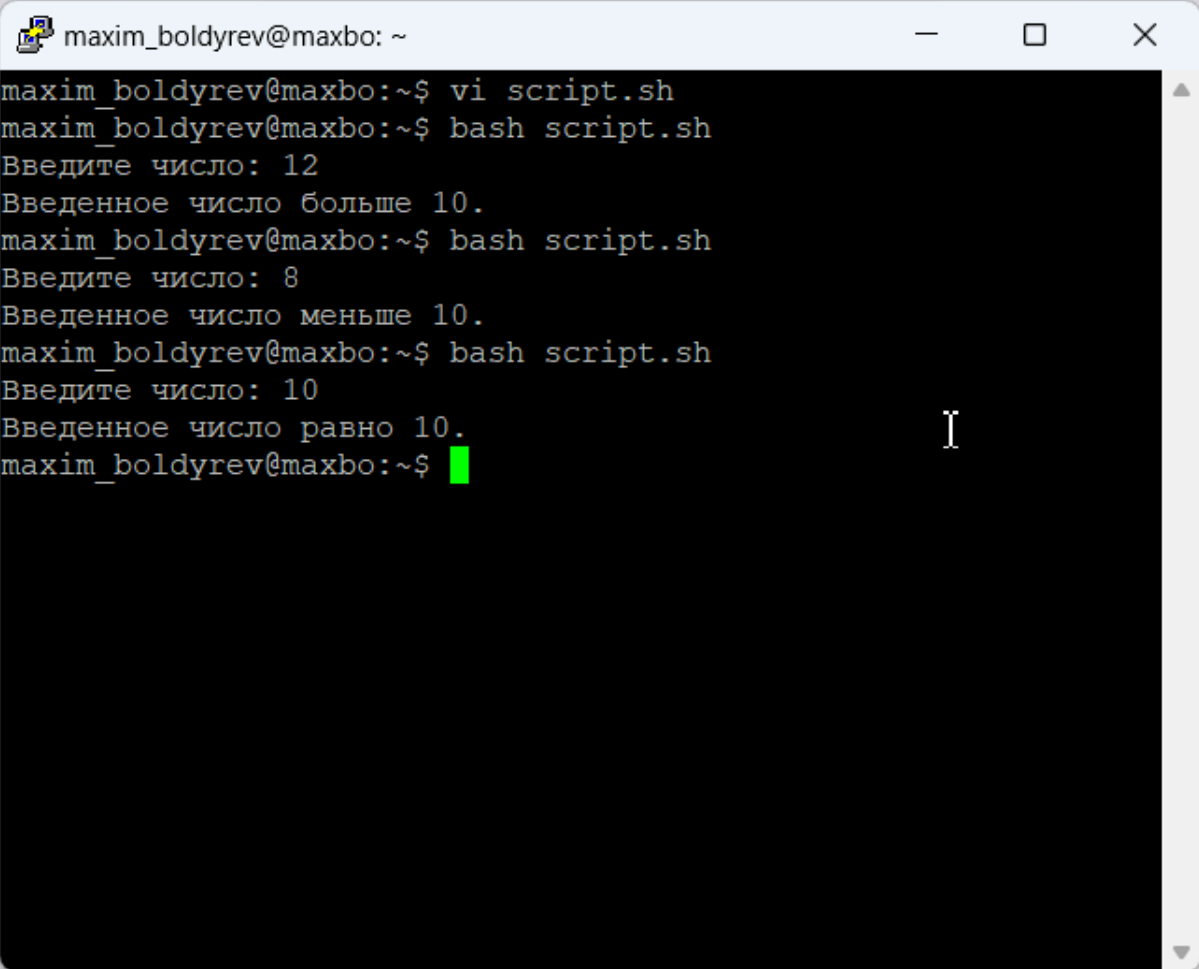
A screenshot of a terminal window titled "maxim_boldyrev@maxbo: ~". The terminal displays a shell script named "script.sh" with seven lines of code:

```
1 for file in *; do  
2     echo "File: $file"  
3     echo "-----"  
4     head -n 20 "$file"  
5     echo "-----"  
6     sleep 1  
7 done
```

The cursor is positioned at the end of the seventh line. Below the script, there are several tilde (~) symbols representing directory listings. At the bottom of the terminal, status information is shown: "\"script.sh\" 7L, 186B" on the left, "1,1" in the center, and "Весь" (Ves' - All) on the right.


```
maxim_boldyrev@maxbo: ~  
1 #!/bin/bash  
2  
3 # Запрос ввода числа  
4 read -p "Введите число: " number  
5  
6 # Проверка значения  
7 if [ $number -eq 10 ]; then  
8     echo "Введенное число равно 10."  
9 elif [ $number -lt 10 ]; then  
10    echo "Введенное число меньше 10."  
11 else  
12    echo "Введенное число больше 10."  
13 fi
```

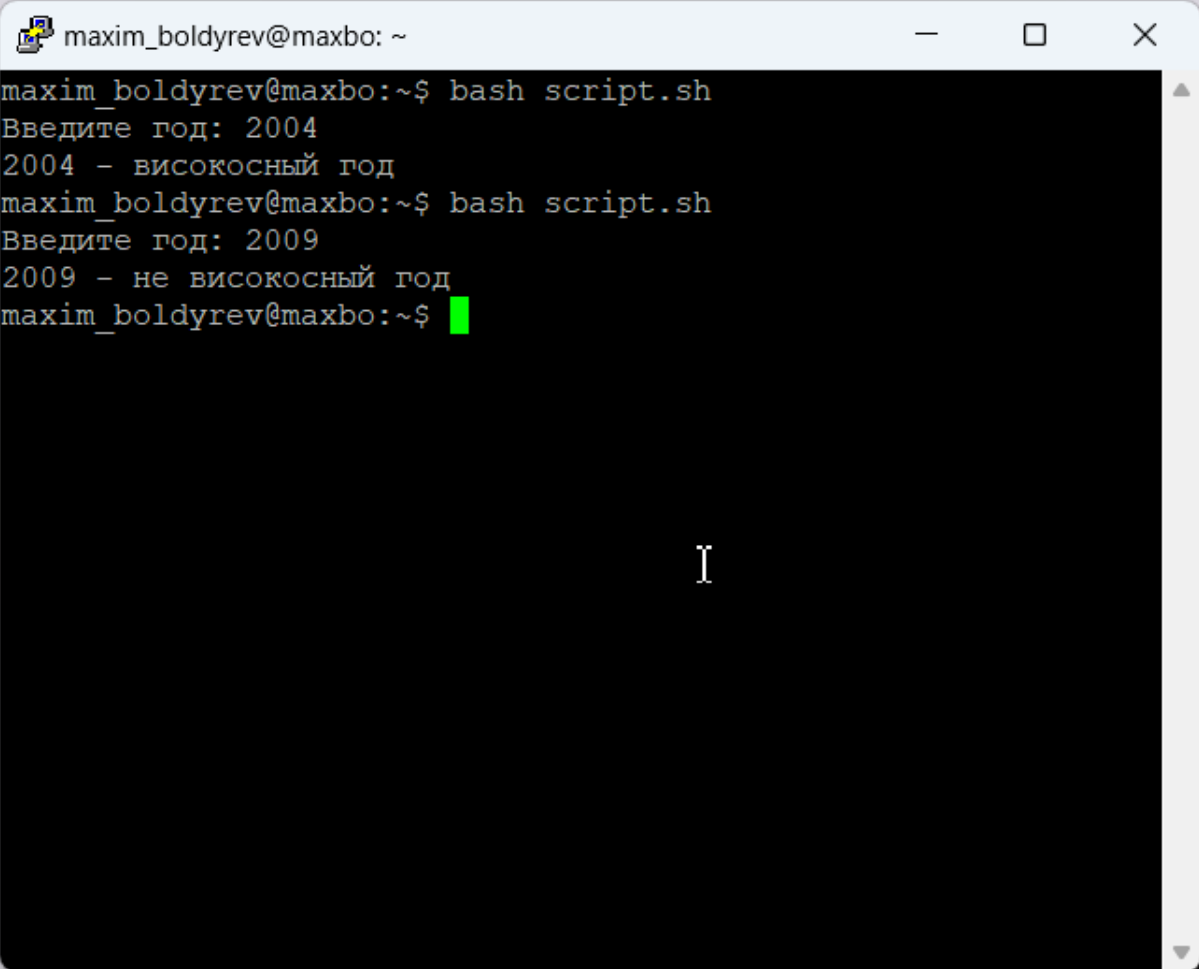
"script.sh" 13L, 370B 1,1 Весь

A terminal window titled 'maxim_boldyrev@maxbo: ~' with standard window controls. The terminal shows a script being edited and then executed three times. The first execution takes input 12 and outputs 'Введенное число больше 10.'. The second takes input 8 and outputs 'Введенное число меньше 10.'. The third takes input 10 and outputs 'Введенное число равно 10.'. A green cursor is visible at the end of the last command line.

```
maxim_boldyrev@maxbo:~$ vi script.sh
maxim_boldyrev@maxbo:~$ bash script.sh
Введите число: 12
Введенное число больше 10.
maxim_boldyrev@maxbo:~$ bash script.sh
Введите число: 8
Введенное число меньше 10.
maxim_boldyrev@maxbo:~$ bash script.sh
Введите число: 10
Введенное число равно 10.
maxim_boldyrev@maxbo:~$
```

9. Программой запрашивается год, определяется, високосный ли он. Результат выводится на экран.

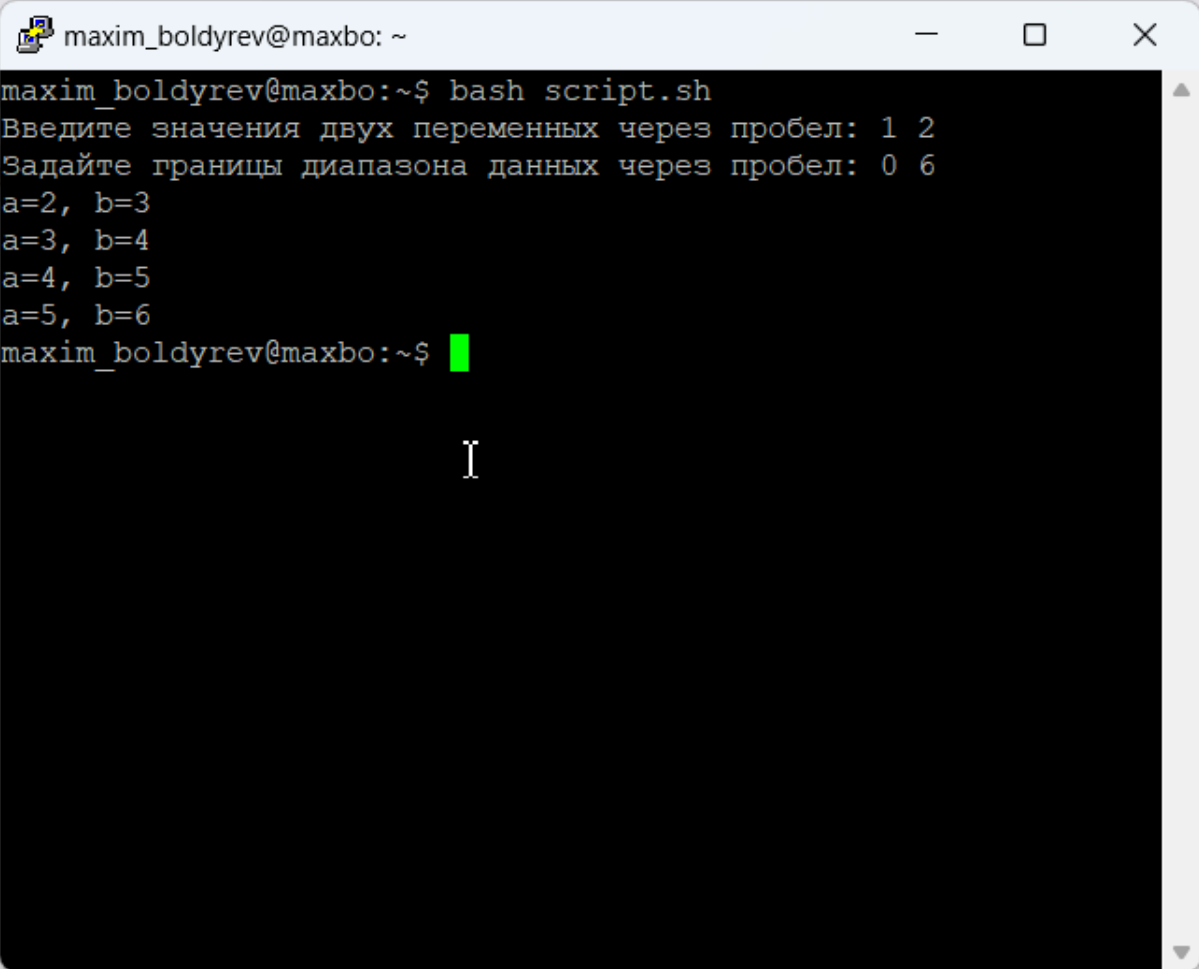
[illegible]

A terminal window titled 'maxim_boldyrev@maxbo: ~' with standard window controls. The terminal shows the execution of a script named 'script.sh'. The first run takes the input '2004' and outputs '2004 - високосный год'. The second run takes the input '2009' and outputs '2009 - не високосный год'. The prompt is a green cursor.

```
maxim_boldyrev@maxbo:~$ bash script.sh
Введите год: 2004
2004 - високосный год
maxim_boldyrev@maxbo:~$ bash script.sh
Введите год: 2009
2009 - не високосный год
maxim_boldyrev@maxbo:~$
```

10. Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются.

```
maxim_boldyrev@maxbo: ~  
1 #!/bin/sh  
2  
3 read -p "Введите значения двух переменных через пробел: "  
  a b  
4  
5 read -p "Задайте границы диапазона данных через пробел: "  
  c d  
6  
7 if [ $a -gt $c ] && [ $b -gt $c ]  
8  
9 then  
10  
11 while [ $a -lt $d ] && [ $b -lt $d ]  
12  
13 do  
14  
15 a=$((a+1))  
16  
17 b=$((b+1))  
18  
19 echo "a=$a, b=$b" "Вы не попали"  
20  
21 done  
"script.sh" 27L, 426B 1,1 Наверху
```

A terminal window titled 'maxim_boldyrev@maxbo: ~' with standard window controls. The terminal shows the execution of 'bash script.sh'. It prompts for two values separated by a space, which are '1' and '2'. Then it prompts for range boundaries, which are '0' and '6'. The script then prints four lines: 'a=2, b=3', 'a=3, b=4', 'a=4, b=5', and 'a=5, b=6'. The prompt returns to 'maxim_boldyrev@maxbo:~\$' with a green cursor. A large vertical cursor is also visible in the center of the terminal area.

```
maxim_boldyrev@maxbo:~$ bash script.sh
Введите значения двух переменных через пробел: 1 2
Задайте границы диапазона данных через пробел: 0 6
a=2, b=3
a=3, b=4
a=4, b=5
a=5, b=6
maxim_boldyrev@maxbo:~$
```

11. В качестве аргумента командной строки вводится пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc.

The screenshot shows a terminal window titled "maxim_boldyrev@maxbo: ~". The terminal displays the execution of a shell script named "script.sh". The script contains the following code:

```
#!/bin/bash  
read -s -p "Введите пароль: " password  
echo  
if [[ $password == "1234" ]]; then  
    ls -la /etc | less  
else  
    echo "Неправильный пароль."  
fi
```

The prompt character is "~". At the bottom of the terminal, there are status indicators: "`"script.sh"` 10L, 187B" on the left, "1,1" in the center, and "Весь" (All) on the right.


```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ bash script.sh  
Введите название файла: main.cpp  
#include <iostream>  
#include <algorithm>  
#include <iostream>  
  
using namespace std;  
  
int main() {  
    int n;  
    std::cout << "Введите размер массива: ";  
    std::cin >> n;  
  
    int* arr = new int[n];  
  
    std::cout << "Введите элементы массива: ";  
    for (int i = 0; i < n; i++) {  
        std::cin >> arr[i];  
    }  
  
    std::sort(arr, arr + n, std::greater<int>());  
  
    std::cout << "Отсортированный массив: ";  
    for (int i = 0; i < n; i++) {
```

```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ bash script.sh  
Введите название файла: rayan.txt  
Файл не существует  
maxim_boldyrev@maxbo:~$
```

13. Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог, просматривается содержимое этого файла.

[illegible]

```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ bash script.sh newarhl  
total 4  
-rw-rw-r-- 1 maxim_boldyrev maxim_boldyrev 0 сен 25 22:21 hello.cpp  
drwxrwxr-x 2 maxim_boldyrev maxim_boldyrev 4096 сен 25 22:19 new  
maxim_boldyrev@maxbo:~$ bash script.sh folder  
maxim_boldyrev@maxbo:~$ ll  
total 240  
drwxr-x--- 11 maxim_boldyrev maxim_boldyrev 4096 ноя 29 15:15 ./  
drwxr-xr-x 8 root root 4096 ноя 18 08:59 ../  
-rw-rw-r-- 2 maxim_boldyrev maxim_boldyrev 0 сен 25 23:36 2.txt  
-rw-rw-r-- 1 maxim_boldyrev maxim_boldyrev 10240 сен 25 22:23 arhl.tar  
-rw----- 1 maxim_boldyrev maxim_boldyrev 7372 ноя 29 14:22 .bash_history  
-rw-r--r-- 1 maxim_boldyrev maxim_boldyrev 220 янв 6 2022 .bash_logout  
-rw-r--r-- 1 maxim_boldyrev maxim_boldyrev 3771 янв 6 2022 .bashrc  
drwx----- 2 maxim_boldyrev maxim_boldyrev 4096 сен 24 16:09 .cache/  
-rw-rw-r-- 1 maxim_boldyrev maxim_boldyrev 0 ноя 29 14:13 code  
-rw----- 1 maxim_boldyrev maxim_boldyrev 12288 ноя 29 14:22 .file1.txt.swo  
-rw----- 1 maxim_boldyrev maxim_boldyrev 12288 ноя 29 14:12 .file1.txt.swp  
-rw-rw-r-- 1 maxim_boldyrev maxim_boldyrev 0 ноя 29 14:12 file.txt  
drwxrwxr-x 2 maxim_boldyrev maxim_boldyrev 4096 ноя 29 15:15 folder/  
-rw-rw-r-- 2 maxim_boldyrev maxim_boldyrev 0 сен 25 23:36 hardlink  
-rw-rw-r-- 1 maxim_boldyrev maxim_boldyrev 0 сен 25 22:21 hello.cpp  
-rw----- 1 maxim_boldyrev maxim_boldyrev 20 ноя 29 14:58 .lessht  
drwxrwxr-x 3 maxim_boldyrev maxim_boldyrev 4096 сен 24 20:53 .local/
```

14. Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются соответствующие сообщения (использовать имена файлов и/или позиционные параметры).

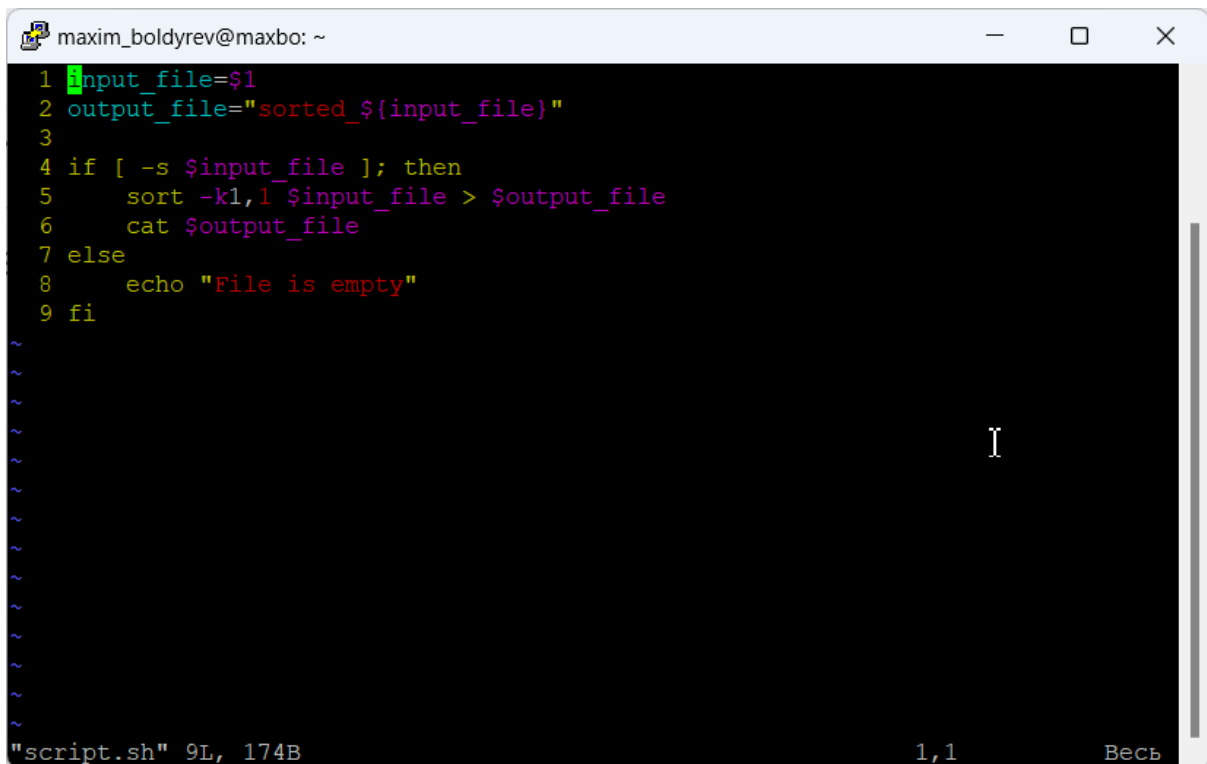
[illegible]

```
maxim_boldyrev@maxbo: ~  
1  
  
VIM - Vi IMproved (улучшенный Vi)  
  
    версия 8.2.213  
      Брам Мооленаар и другие  
С изменениями, внесёнными team+vim@tracker.debian.org  
Vim - свободно распространяемая программа с открытым кодом  
  
Станьте зарегистрированным пользователем Vim!  
наберите :help register<Enter>   для получения информации  
  
наберите :q<Enter>               чтобы выйти из программы  
наберите :help<Enter> или <F1>   для получения справки  
наберите :help version8<Enter>   для информации о версии
```

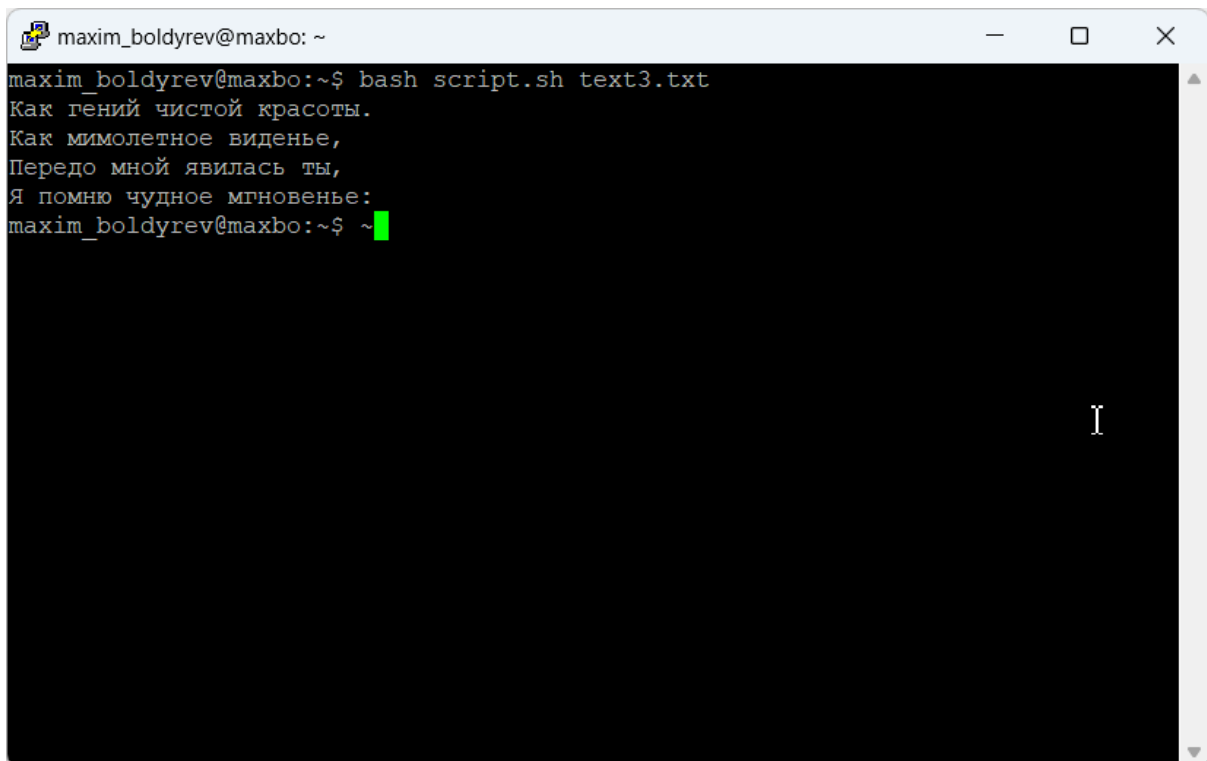
0,0-1 Весь

16. В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля, содержимое файла сортируется по первому столбцу по возрастанию, отсортированная

информация помещается в другой файл, содержимое которого затем отображается на экране.



```
maxim_boldyrev@maxbo: ~  
1 input_file=$1  
2 output_file="sorted_${input_file}"  
3  
4 if [ -s $input_file ]; then  
5     sort -k1,1 $input_file > $output_file  
6     cat $output_file  
7 else  
8     echo "File is empty"  
9 fi  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
"script.sh" 9L, 174B 1,1 Весь
```



```
maxim_boldyrev@maxbo: ~  
maxim_boldyrev@maxbo:~$ bash script.sh text3.txt  
Как гений чистой красоты.  
Как мимолетное виденье,  
Передо мной явилась ты,  
Я помню чудное мгновенье:  
maxim_boldyrev@maxbo:~$ ~
```

Выводы

Изучил основные возможности языка программирования Shell, выполнил практические задания с использованием встроенного в терминал интерпретатора sh и посредством написания скриптов в исполняемых файлах и последующим их исполнением.

Ответы на контрольные вопросы

В чем отличие пользовательских переменных от переменных среды?

Пользовательские переменные - это переменные, определенные и использованные конкретным пользователем внутри его рабочей среды. Они могут быть созданы и изменены пользователем с целью использования в своих скриптах или приложениях. Обычно они имеют локальную область видимости и доступны только в пределах рабочей сессии пользователя.

Переменные среды - это переменные, которые определены на уровне операционной системы и доступны для всех пользователей и приложений, работающих на компьютере. Они используются для хранения информации, которая может быть передана между различными процессами или программами. Примеры таких переменных: PATH, HOME, USER. Переменные среды обычно имеют глобальную область видимости.

Математические операции в SHELL.

Математические операции в SHELL подобны тем, что используются в других языках программирования высокого уровня (например, Python): «+» - сложение ($a + b$), «-» - вычитание ($a - b$), «*» - умножение ($a * b$), «/» - деление (a / b), «%» - остаток от деления ($a \% b$), «**» - возведение в степень ($a ** b$).

В SHELL для математических операций мы можем использовать, например, следующий синтаксис: `result=$(($a + $b))`. Либо можно использовать `expr`: `result=$(expr $a + $b)`.

Условные операторы в SHELL.

Shell поддерживает 2 вида условных операторов: `if` и `case`, которые имеют следующий синтаксис:

```
if [ УСЛОВИЕ1 ] ; then
    # Блок кода, если условие1 выполнено
elif [ УСЛОВИЕ2 ] ; then
    # Блок кода, если условие2 выполнено
else
    # Блок кода, если ни одно из описанных условий не выполнено
```

```
fi
```

```
case ПЕРЕМЕННАЯ in
    значение1)
        команда1
        ;;
    значение2)
        команда2
        ;;
    *)
        команда3
        ;;
esac
```

Принципы построения простых и составных условий.

«&&» или «-a» - логическое И; «||» или «-o» - логическое ИЛИ; «=» или «==» - проверка на равенство; «!=» или «-ne» - проверка на НЕравенство; «-z» - проверяет, что значение переменной пусто; «-n» - проверяет, что значение переменной НЕпустое; также поддерживает все операции сравнения типа больше/меньше/равно: «>», «<», «>=», «<=» или «-gt» (greater than), «-lt» (less than), «-ge» (greater or equal), «-le» (less or equal).

Циклы в SHELL.

Поддерживаются циклы while, until, for. Цикл while выполняется до тех пор, пока условие истинно; цикл until, напротив, выполняется до тех пор, пока условие ложно; цикл for используется для итерации по элементам некоторого массива, множества элементов.

```
while [ УСЛОВИЕ ] do
    # Блок кода, если условие выполнено
done
```

```
until[ УСЛОВИЕ ] do
    # Блок кода, если условие не выполнено
done
```

```
for ПЕРЕМЕННАЯ in МНОЖЕСТВО_ЭЛЕМЕНТОВ
    # Блок кода, использующий элемент из множества
done
```

Причем каждый цикл поддерживает операторы `break` (прерывание выполнения цикла и переход к следующему за ним блоку кода) и `continue` (прерывание ИТЕРАЦИИ цикла и переход к следующей ИТЕРАЦИИ).

Массивы и модули в SHELL.

В Shell массивы могут быть использованы так же, как и в других языках программирования. Их синтаксис следующий:

- Определение массива:

```
myArray=("value1" "value2" "value3")
```

то есть элементы просто перечисляются через пробел (или символ новой строки `\n`). Например, таким образом можно сохранить в переменную результат вывода некоторой команды: `lsResult=$(ls)`.

- Доступ к элементу по индексу:

```
array=("a" "b" "c")
```

```
echo ${array[0]}
```

(Вывод: "a")

Причем следующая команда выведет все элементы массива:

```
echo ${array[@]}
```

(Вывод: "a" "b" "c")

А следующая команда выведет количество элементов в массиве:

```
echo ${#array[@]}
```

(Вывод: 3)

Также по индексу можно присвоить новое значение элемента:

```
${array[2]="d"}
```

- Удаление элемента массива:

```
unset array[2]
```

- Добавление элементов в конец массива:

```
${array+=("e" "f")}
```

Чтение параметров командной строки.

При запуске скрипта мы можем передать ему некоторые позиционные параметры следующим образом:

```
bash script.sh par1 par2 par3 ...
```

В коде работа с параметрами происходит следующим образом:

`$0` - всегда **имя файла скрипта** (в данном случае `script.sh`)

`$1` - первый позиционный параметр (в данном случае `par1`)

\$2 - второй позиционный параметр (par2) и тд

Если скрипт предполагает передачу неопределенного количества параметров, можно использовать цикл for для итерации по всем переданным параметрам:

```
for parameter in "$@" do
    #do smth
done
```

А узнать в скрипте количество переданных параметров можно так: \$#

Как различать ключи и параметры?

Ключи представляют из себя некоторые опции, указывающие на некоторые дополнительные настройки/функционал используемой команды, и имеют префикс «-» (состоящие из одного символа, например, «-h») либо «--» (обычно для ключей, представляющих собой целое слово или выражение: «--help»).

Параметры префиксов не имеют.

Сначала передаются ключи, затем параметры. Например:

```
ls -a myDir
```

-a здесь - ключ, указывающий вывести все файлы и подкаталоги в указанном каталоге myDir, в том числе скрытые.

myDir - параметр, в данном случае указывающий показать содержимое каталога myDir.

Чтение данных из файлов.

В shell содержимое файла можно считать теми же командами, что и в терминале, например, командой cat:

```
cat filetoread.txt
```

Считать построчно:

```
while read line; do
    # do smth
done < filetoread.txt
```

Или считать файл построчно, сохранив строки как элементы массива в переменную (ниже - в переменную lines):

```
readarray lines < filetoread.txt
```

Стандартные дескрипторы файлов.

- Дескриптор стандартного ввода (stdin) - используется для чтения данных из стандартного входного потока. По умолчанию это клавиатура, но его можно перенаправить на другой ввод, например, через файл или канал.
- Дескриптор стандартного вывода (stdout) - используется для вывода данных в стандартный поток вывода. По умолчанию результаты выводятся на экран, но его также можно перенаправить в файл или через канал.
- Дескриптор стандартной ошибки (stderr) - используется для вывода сообщений об ошибках и диагностической информации. По умолчанию результаты выводятся на экран, но его также можно перенаправить в файл или через канал.

Эти дескрипторы имеют определенные числовые значения:

- stdin: 0
- stdout: 1
- stderr: 2

Перенаправление вывода.

Результаты вывода команд можно перенаправить для вывода не на экран, например, в файл. Операторы перенаправления: «>», «>>», «|»

- ls > result.txt - Перенаправит вывод команды ls в файл result.txt. Если файл существует, его содержимое будет перезаписано.

- ls >> result.txt - Сделает то же, что и «>», но если файл уже существует, результат будет записан в конец файла.

- ls | grep "test" - Перенаправит результат вывода команды ls в команду grep, которая отфильтрует полученный результат и выведет только те результаты, которые содержат слово «test». Таким образом, оператор «|» перенаправляет вывод одной команды как входные данные в другую команду.

Подавление вывода.

Подавление вывода команды осуществляется перенаправлением вывода «в никуда», а именно в /dev/null - специальное устройство, игнорирующее все входящие данные:

```
ls > /dev/null
```

Следующая команда будет подавлять вывод ошибок и стандартного вывод команды ls:

```
ls > /dev/null 2>$1
```

Отправка сигналов скриптам.

Скриптам в процессе выполнения можно отправлять сигналы так же, как и другим процессам, например kill, killall и другие.

Использование функций.

Синтаксис функций в shell:

```
function myfunc {  
    # do smth  
}
```

Для использования команды в скрипте достаточно указать её название: myfunc.

Функции поддерживают также оператор return. Пример использования:

```
function myfunc {  
    read -p "Enter a value: " value  
    echo "adding value"  
    return $(( $value + 10 ))  
}  
myfunc  
echo "The new value is $?"
```

Здесь «\$?» используется для взятия значения из функции. Если мы выполним любую другую команду перед извлечением возвращенного значения из функции, результат будет утерян.

Отправка сообщений в терминал пользователя.

Это происходит с помощью команды echo:

```
echo "Hello, it is script file $0"
```

BASH и SHELL - синонимы?

SHELL (интерпретатор командной оболочки) - это общий термин, который описывает программу, предоставляющую пользователю интерфейс для взаимодействия с операционной системой. Она обрабатывает команды, вводимые пользователем, и выполняет их.

BASH (Bourne Again SHell) - это одна из наиболее распространенных командных оболочек в UNIX-подобных системах. BASH является

расширением исходного кода оригинальной командной оболочки Unix - Bourne shell (sh), и предоставляет дополнительные функции и улучшения. Таким образом, BASH - это конкретная реализация командной оболочки, в то время как SHELL - это более широкое понятие, описывающее класс программ, осуществляющих взаимодействие пользователя с операционной системой.

PowerShell в операционных системах семейства Windows: назначение и особенности.

PowerShell - это мощный интерактивный оболочечный язык и среда командной строки, разработанные компанией Microsoft для операционных систем Windows. Он был выпущен в 2006 году и является продолжением более старой командной оболочки cmd.exe.

Назначение PowerShell заключается в автоматизации задач администрирования и управления компьютером. Он предоставляет мощные средства для создания и выполнения сценариев, управления файлами и папками, настройки и управления службами, установки и удаления программных пакетов, редактирования реестра и многого другого. PowerShell также поддерживает управление удаленными компьютерами и Active Directory.

Особенности:

- **Объектно-ориентированность:** PowerShell представляет результаты команд в виде объектов, которые можно легко фильтровать, сортировать и передавать на вход других команд.
- **Расширяемость:** PowerShell поддерживает создание пользовательских модулей, которые добавляют новые команды и функциональность.
- **Интеграция с другими технологиями Microsoft:** PowerShell может использоваться для автоматизации работы с продуктами Microsoft, такими как SQL Server, Exchange, SharePoint и другими.
- **Кросс-платформенность:** начиная с версии PowerShell 6.0, Microsoft выпустила кросс-платформенную версию PowerShell, которая работает также на Linux и macOS.