**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Программа ДПО Intaro

ЛАБОРАТОРНАЯ РАБОТА №4

“Создание и использование сценариев (скриптов) в Linux”

Студент \_\_\_\_\_\_\_\_\_\_ Целищев А.Е.

ПМ-21-2

Руководитель \_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Доцент каф. АСУ

Липецк 2023 г

Оглавление

[Цели работы 3](#_Toc9114)

[1 часть 4](#_Toc27089)

[2 часть 6](#_Toc4632)

[Выводы 1](#_Toc32722)7

[Ответы на контрольные вопросы 18](#_Toc5737)

**Цели работы**

- Изучить основные возможности языка программирования высокого уровня Shell;

- Получить навыки написания и использования скриптов.

**1 часть**

1. Выведем информационные сообщения с помощью echo и printf



Рис 1. Вывод сообщения с помощью echo.



Рис 2. Вывод сообщения с помощью printf.

1. Присвоим переменной *А* целочисленное значение и выведем её в терминал:

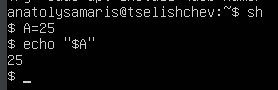


Рис 3. Присваивание значения переменной и вывод ее значения.

1. Присвоить переменной *В* значение переменной *А*. Посмотреть значение переменной *В*.

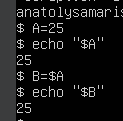


Рис 4. Присваивание переменной значения другой переменной.

1. Присвоить переменной *С* значение «ПутьДоСвоегоКаталога». Перейти в этот каталог с использованием переменной.

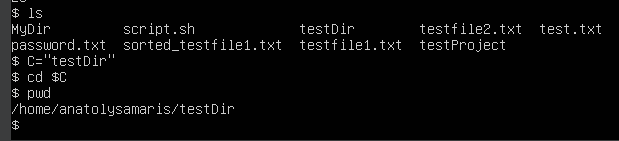


Рис 5. Использование переменной для перехода в другой каталог.

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



Рис 6. Использование переменной для выполнения команды date.

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



Рис 7. Использование переменной для чтения содержимого файла.

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

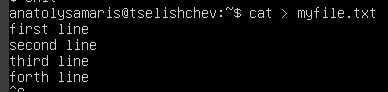


Рис 8. Создание тестового файла.

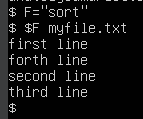


Рис 9. Использование переменной для сортировки содержимого файла.

**2 часть**

1. Для удобства будем выполнять работу в отдельном файле script.sh, в котором будем писать скрипты и исполнять его в командной строке. Используя Vim, создадим файл script.sh. В первой строке напишем #!/bin/bash, таким образом обозначив местоположение интерпретатора, а дальше напишем код скрипта. В данном случае это программа для присвоения целочисленного значения переменной *А* и просмотра значения переменной:

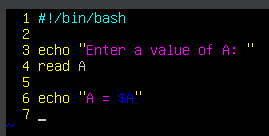


Рис 10. Код программы script.sh.

Сделаем файл исполняемым, присвоив всей системе право на исполнение файла, и затем командой bash script.sh вызовем файл на исполнение.

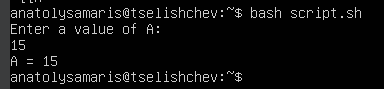


Рис 11. Вызов скрипта с помощью bash.

1. Программа запрашивает имя пользователя, затем здоровается с ним, используя значение переменной.

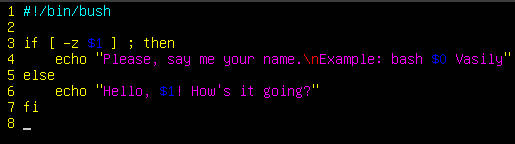


Рис 12. Код задачи 2.

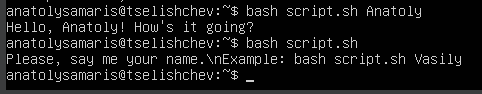


Рис 13. Работа скрипта задачи 2.

1. Программа запрашивает значения двух переменных, вычисляет произведение этих переменных, используя команды «expr» и «bc».

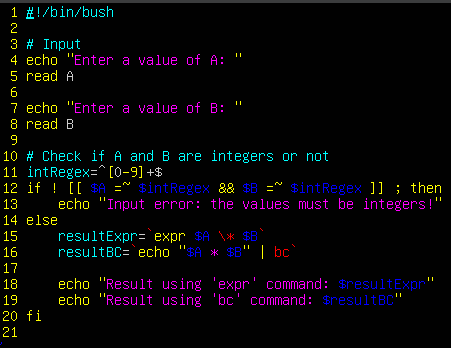


Рис 14. Код задачи 3.

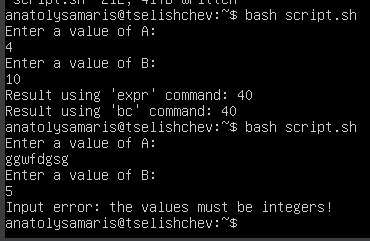


Рис 15. Работа скрипта задачи 3.

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

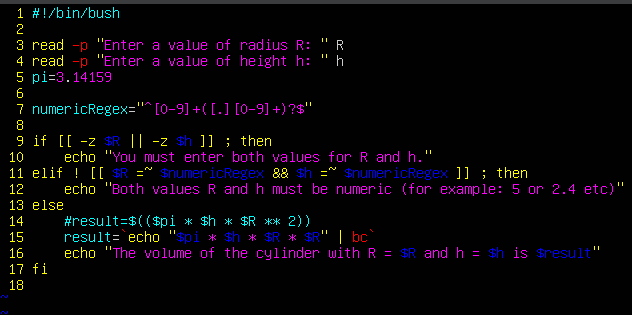


Рис 16. Код задачи 4.

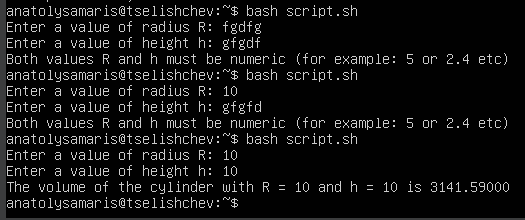


Рис 17. Работа скрипта задачи 4.

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

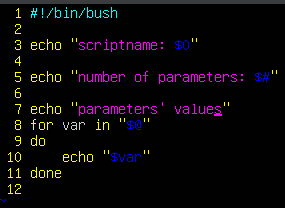


Рис 18. Код задачи 5.

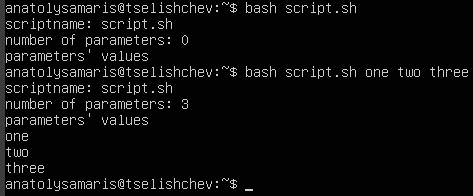


Рис 19. Работа скрипта задачи 5.

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

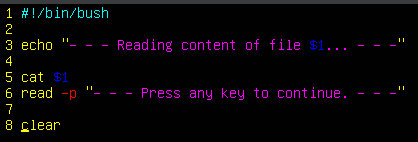


Рис 20. Код задачи 6.

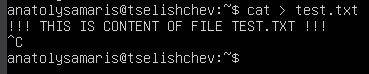


Рис 21. Создание тестового файла для задачи 6.

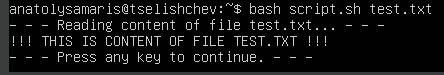


Рис 22. Работа скрипта задачи 6.

1. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно.

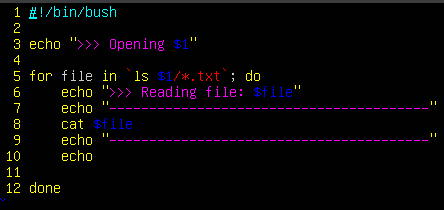


Рис 23. Код задачи 7.

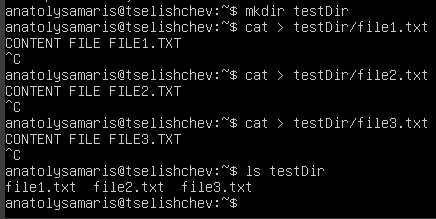


Рис 24. Создание каталога и файлов для задачи 7.

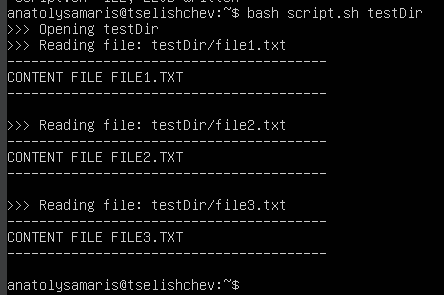


Рис 25. Работа скрипта задачи 7.

1. Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения.

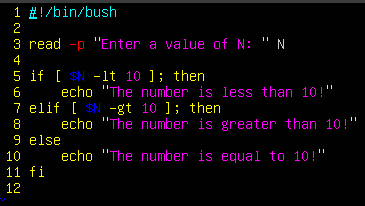


Рис 26. Код задачи 8.

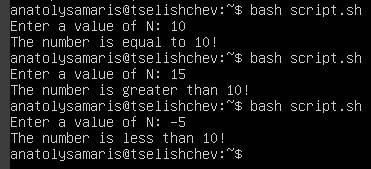


Рис 27. Работа скрипта задачи 8.

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

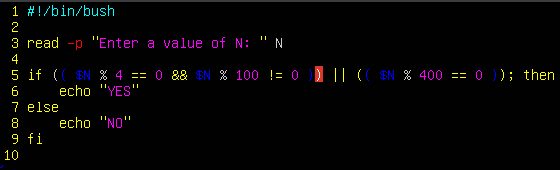


Рис 28. Код задачи 9.

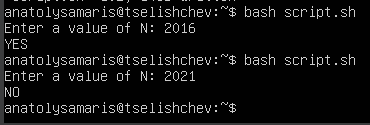


Рис 29. Работа скрипта задачи 9.

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

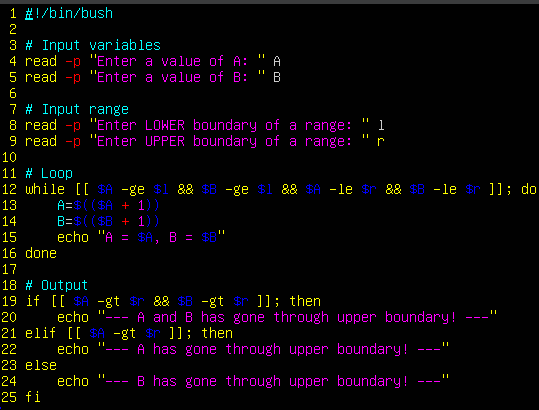


Рис 30. Код задачи 10.

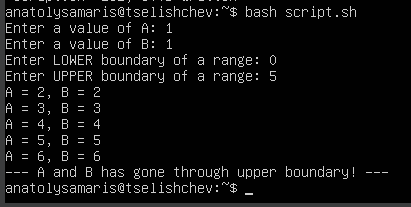


Рис 31. Работа скрипта задачи 10.

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

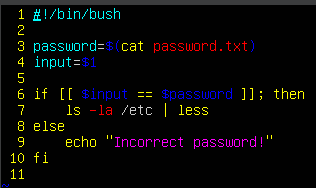


Рис 32. Код задачи 11.

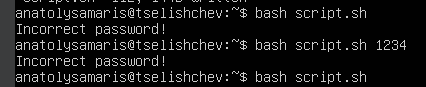


Рис 33. Работа скрипта задачи 11.

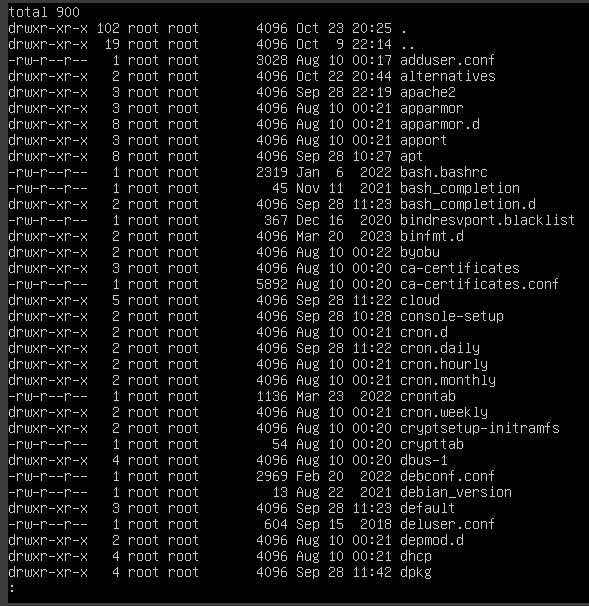


Рис 34. Вывод при правильном пароле в задаче 11.

1. Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выводится соответствующее сообщение.

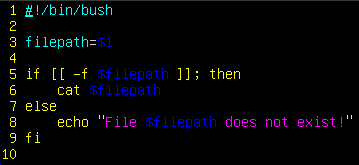


Рис 35. Код задачи 12.

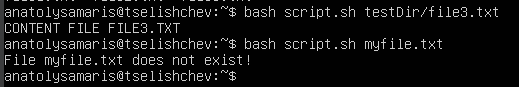


Рис 36. Работа скрипта задачи 12.

1. 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.

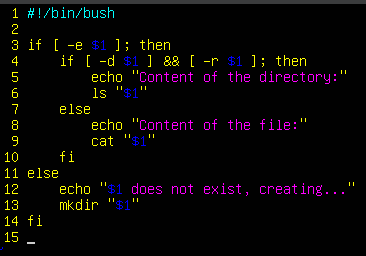


Рис 37. Код задачи 13.

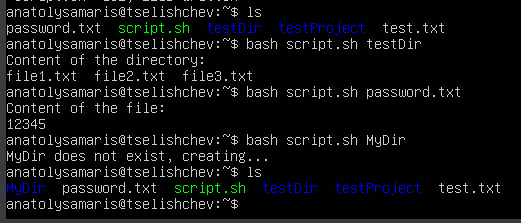


Рис 38. Работа скрипта задачи 13.

1. Анализируются атрибуты файла. Если первый файл существует и ис-

пользуется 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).

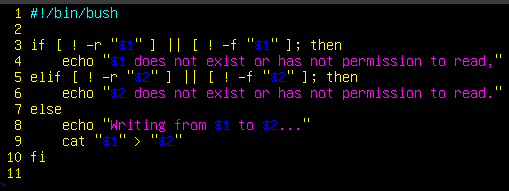


Рис 39. Код задачи 14.

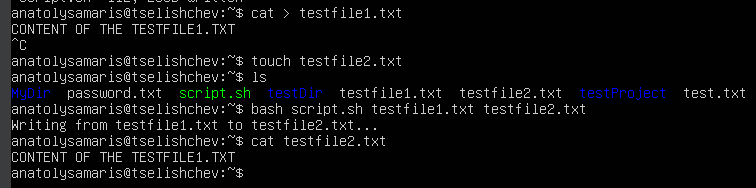


Рис 40. Работа скрипта задачи 14.

1. Если файл запуска программы найден, программа запускается (по выбору).

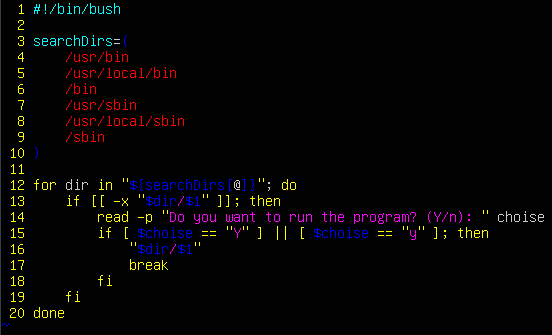


Рис 41. Код задачи 15.



Рис 42. Работа скрипта задачи 15.



Рис 43. Результат работы скрипта.

1. 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.

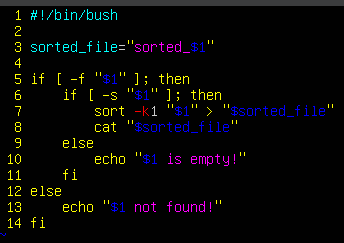


Рис 44. Код задачи 16.



Рис 45. Работа скрипта задачи 16.

**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 «-а» - 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/фукционал 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 вводные 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.