МИНИСТЕСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Отчет по лабораторной работе № 1.3

на тему «Скрипты, диалоги, создание проекта»

по дисциплине «Системное программирование»

Кафедра программного обеспечения информационных систем и технологий

Выполнили

Студенты группы 10702122 Н.В. Тимощенко

Д.Е. Кудрявец

Проверил доцент Галай В.В

Минск 2024

**Цель работы:**

Закрепить на практике принципы создания проектов с помощью скриптов SHELL, освоить средства примитивного графического интерфейса в Linux-скриптах..

**Выполнение работы:**

**Задание 1:**

Пункт 1:

Сделать пример с двумя вложенными диалогами типа YesNo.

Код:

#!/bin/bash

# Первый диалог Yes/No

dialog --yesno "Вы хотите продолжить?" 7 60

response1=$?

if [ $response1 -eq 0 ]; then

# Если ответ "Yes", запускаем второй диалог Yes/No

dialog --yesno "Вы уверены, что хотите продолжить?" 7 60

response2=$?

if [ $response2 -eq 0 ]; then

# Если ответ "Yes" на второй диалог

dialog --msgbox "Вы выбрали продолжить." 7 60

else

# Если ответ "No" на второй диалог

dialog --msgbox "Вы выбрали не продолжать." 7 60

fi

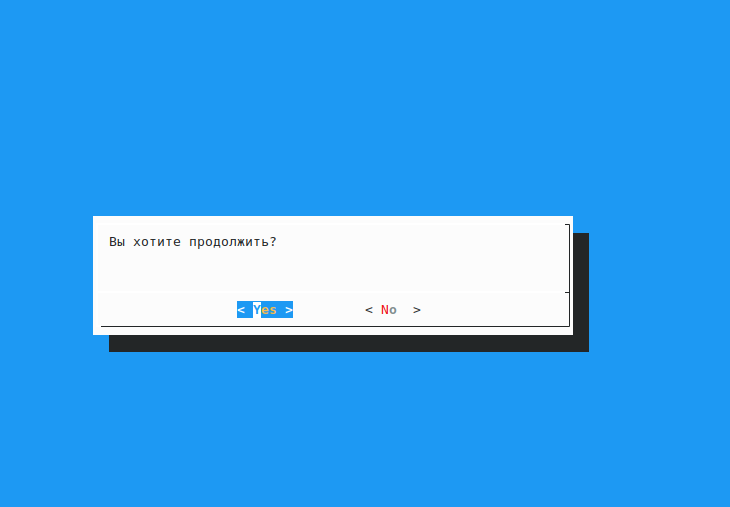
else

# Если ответ "No" на первый диалог

dialog --msgbox "Вы выбрали не продолжать." 7 60

fi

Результат:



Пункт 2:

Сделать пример, запрашивающий сначала имя человека, а потом

профессию. Вывести имя + профессию, прочитанные в диалоге.

Код:

#!/bin/bash

# Запрос имени

name=$(dialog --inputbox "Введите ваше имя:" 8 40 3>&1 1>&2 2>&3 3>&-)

# Проверка, что имя введено

if [ -z "$name" ]; then

dialog --msgbox "Вы не ввели имя. Скрипт завершен." 7 40

exit 1

fi

# Запрос профессии

profession=$(dialog --inputbox "Введите вашу профессию:" 8 40 3>&1 1>&2 2>&3 3>&-)

# Проверка, что профессия введена

if [ -z "$profession" ]; then

dialog --msgbox "Вы не ввели профессию. Скрипт завершен." 7 40

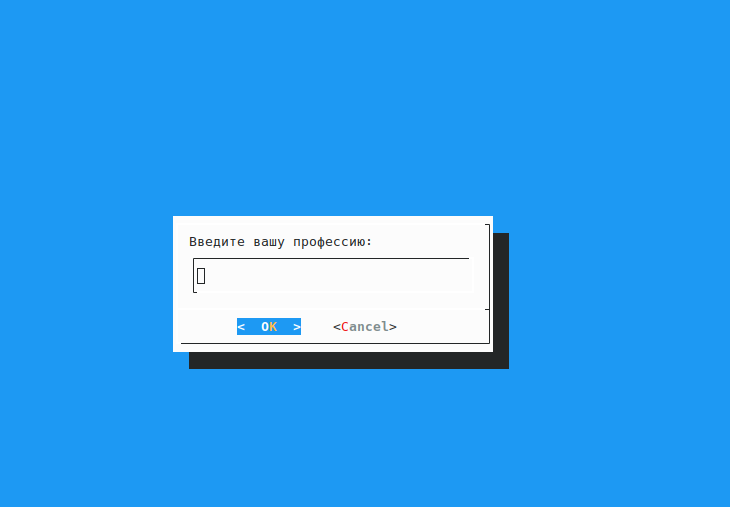
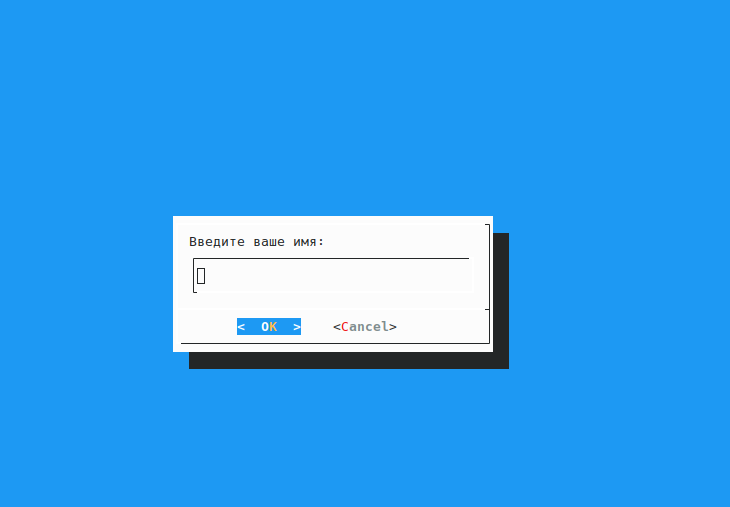
exit 1

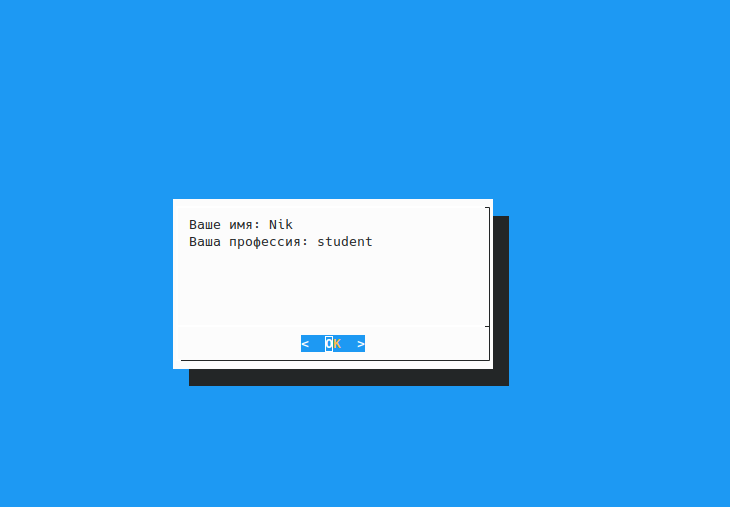
fi

# Вывод имени и профессии

dialog --msgbox "Ваше имя: $name\nВаша профессия: $profession" 10 40

Результат:





Пункт 3:

Вывести список с названиями валют. После выбора валюты система

должна вывести ее котировку.

Код:

#!/bin/bash

# Функция для получения списка валют

get\_currency\_list() {

echo "1. USD"

echo "2. EUR"

echo "3. GBP"

echo "4. JPY"

echo "5. AUD"

}

# Функция для получения котировки выбранной валюты

get\_currency\_rate() {

case $1 in

1) currency="USD";;

2) currency="EUR";;

3) currency="GBP";;

4) currency="JPY";;

5) currency="AUD";;

\*) echo "Неверный выбор"; exit 1;;

esac

# Используем API для получения курса валют

rate=$(curl -s "https://api.exchangerate-api.com/v4/latest/$currency" | jq -r ".rates.BYN")

echo "Курс $currency к BYN: $rate"

}

# Основная программа

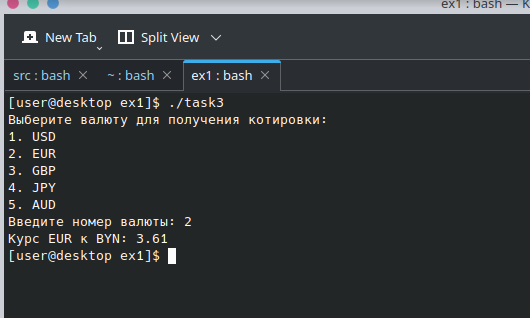
echo "Выберите валюту для получения котировки:"

get\_currency\_list

read -p "Введите номер валюты: " choice

get\_currency\_rate $choice

Результат:



Пункт 4:

Измените предыдущую программу так, вместо списка валют

предлагалось оконное меню валют. Чтобы программа работала в цикле.

Для выхода из цикла нужно вместо названия валюты вводить exit.

Код:

a#!/bin/bash

# Функция для получения котировки выбранной валюты

get\_currency\_rate() {

case $1 in

1) currency="USD";;

2) currency="EUR";;

3) currency="GBP";;

4) currency="JPY";;

5) currency="AUD";;

exit) clear; exit 0;;

\*) dialog --msgbox "Неверный выбор" 6 20; return;;

esac

# Используем API для получения курса валют

rate=$(curl -s "https://api.exchangerate-api.com/v4/latest/$currency" | jq -r ".rates.BYN")

dialog --msgbox "Курс $currency к BYN: $rate" 6 40

}

# Основная программа

while true; do

choice=$(dialog --menu "Выберите валюту для получения котировки (или введите 'exit' для выхода):" 15 50 6 \

1 "USD" \

2 "EUR" \

3 "GBP" \

4 "JPY" \

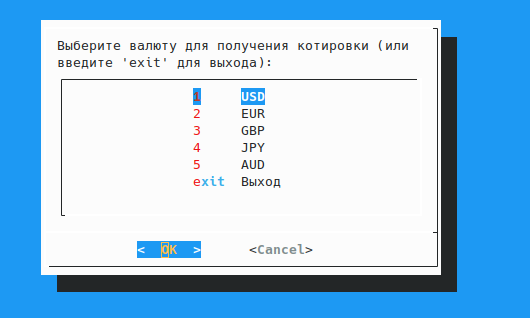
5 "AUD" \

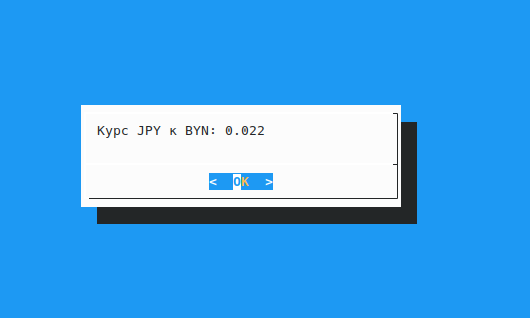
exit "Выход" 3>&1 1>&2 2>&3)

get\_currency\_rate $choice

done

Результат:





**Задание 2:**

Создайте копию проекта представелнного в упражнении No3. Внесем изменеия

в новом проекте, добавив вывод информации о домашнем каталоге каждого

пользователя и включив в вывод общее число файлов и подкаталогов в каждом

из них.

В этой новой версии проверяется наличие привилегий суперпользователя, но

вместо того, чтобы выполнить полный набор операций в каждой из ветвей if,

здесь устанавливаются некоторые переменные, которые затем используются в

цикле for. В функции использованы несколько локальных переменных и

команда printf для форматирования части вывода.

Код:

#!/bin/bash

# sys\_info\_page: программа вывода страницы с информацией о системе

PROGNAME=$(basename $0)

TITLE="System Information Report For $HOSTNAME"

CURRENT\_TIME=$(date +"%x %r %Z")

TIME\_STAMP="Generated $CURRENT\_TIME, by $USER"

report\_uptime () {

cat <<- \_EOF\_

<H2>System Uptime</H2>

<PRE>$(uptime)</PRE>

\_EOF\_

return

}

report\_disk\_space () {

cat <<- \_EOF\_

<H2>Disk Space Utilization</H2>

<PRE>$(df -h)</PRE>

\_EOF\_

return

}

report\_home\_space () {

if [[ $(id -u) -eq 0 ]]; then

cat <<- \_EOF\_

<H2>Home Space Utilization (All Users)</H2>

<PRE>$(du -sh /home/\*)</PRE>

\_EOF\_

else

cat <<- \_EOF\_

<H2>Home Space Utilization ($USER)</H2>

<PRE>$(du -sh $HOME)</PRE>

\_EOF\_

fi

return

}

usage () {

echo "$PROGNAME: usage: $PROGNAME [-f file | -i]"

return

}

write\_html\_page () {

cat <<- \_EOF\_

<HTML>

<HEAD>

<TITLE>$TITLE</TITLE>

</HEAD>

<BODY>

<H1>$TITLE</H1>

<P>$TIME\_STAMP</P>

$(report\_uptime)

$(report\_disk\_space)

$(report\_home\_space)

</BODY>

</HTML>

\_EOF\_

return

}

# обработка параметров командной строки

interactive=

filename=

while [[ -n $1 ]]; do

case $1 in

-f | --file) shift

filename=$1

;;

-i | --interactive)

interactive=1

;;

-h | --help)

usage

exit

;;

\*) usage >&2

exit 1

;;

esac

shift

done

# интерактивный режим

if [[ -n $interactive ]]; then

while true; do

read -p "Enter name of output file: " filename

if [[ -e $filename ]]; then

read -p "'$filename' exists. Overwrite? [y/n/q] > "

case $REPLY in

Y|y) break

;;

Q|q) echo "Program terminated."

exit

;;

\*) continue

;;

esac

fi

done

fi

# вывод страницы html

if [[ -n $filename ]]; then

if touch $filename && [[ -f $filename ]]; then

write\_html\_page > $filename

else

echo "$PROGNAME: Cannot write file '$filename'" >&2

exit 1

fi

else

write\_html\_page

fi

report\_home\_space () {

local format="%8s%10s%10s\n"

local i dir\_list total\_files total\_dirs total\_size user\_name

if [[ $(id -u) -eq 0 ]]; then

dir\_list=/home/\*

user\_name="All Users"

else

dir\_list=$HOME

user\_name=$USER

fi

echo "<H2>Home Space Utilization ($user\_name)</H2>"

for i in $dir\_list; do

total\_files=$(find $i -type f | wc -l)

total\_dirs=$(find $i -type d | wc -l)

total\_size=$(du -sh $i | cut -f 1)

echo "<H3>$i</H3>"

echo "<PRE>"

printf "$format" "Dirs" "Files" "Size"

printf "$format" "----" "-----" "----"

printf "$format" $total\_dirs $total\_files $total\_size

echo "</PRE>"

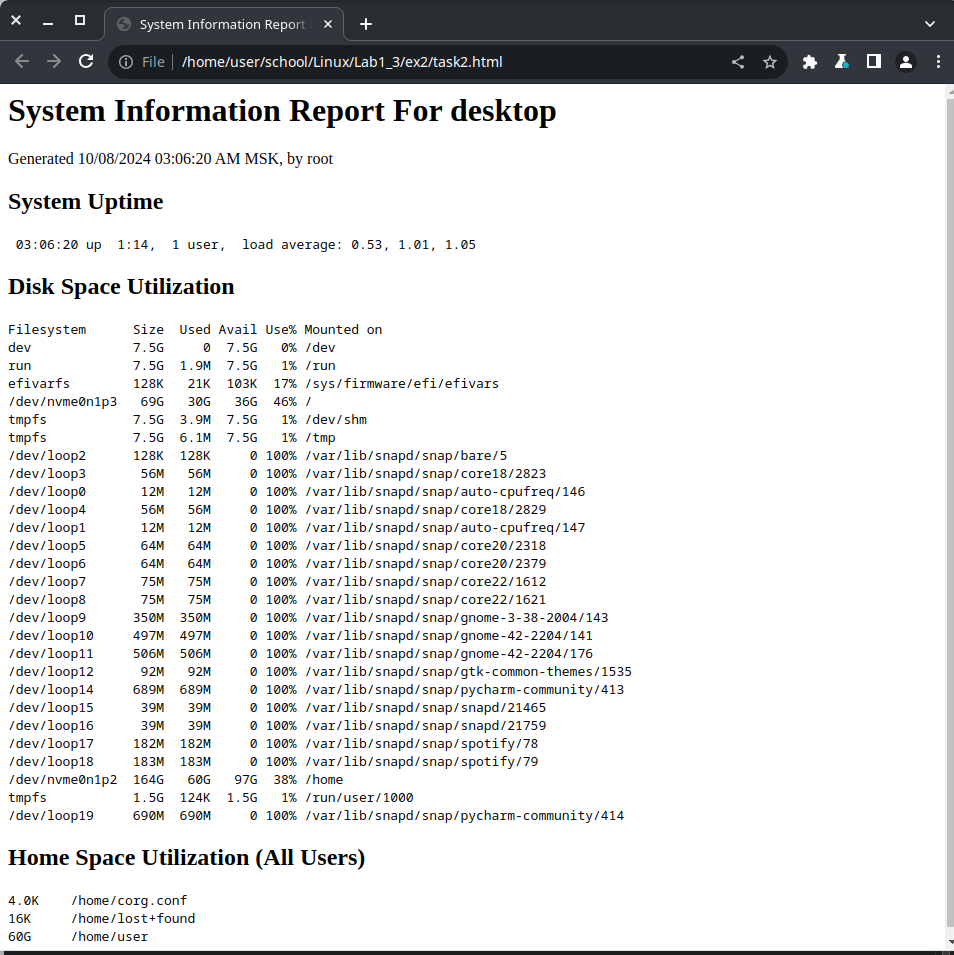
done

return

}

Результат**:**





**Задание 3.**

Доведите проект до совершенства. Для сценария выполненного в задании No2

создайте графическое диалоговое окно, с интерактивным меню выбора

функций. Информация должна автоматически передаваться в HTML-файл и

открываться в браузере.

Код:

#!/bin/bash

# sys\_info\_page: программа вывода страницы с информацией о системе

PROGNAME=$(basename $0)

TITLE="System Information Report For $HOSTNAME"

CURRENT\_TIME=$(date +"%x %r %Z")

TIME\_STAMP="Generated $CURRENT\_TIME, by $USER"

report\_uptime () {

cat <<- \_EOF\_

<H2>System Uptime</H2>

<PRE>$(uptime)</PRE>

\_EOF\_

return

}

report\_disk\_space () {

cat <<- \_EOF\_

<H2>Disk Space Utilization</H2>

<PRE>$(df -h)</PRE>

\_EOF\_

return

}

report\_home\_space () {

local format="%8s%10s%10s\n"

local i dir\_list total\_files total\_dirs total\_size user\_name

if [[ $(id -u) -eq 0 ]]; then

dir\_list=/home/\*

user\_name="All Users"

else

dir\_list=$HOME

user\_name=$USER

fi

echo "<H2>Home Space Utilization ($user\_name)</H2>"

for i in $dir\_list; do

total\_files=$(find $i -type f | wc -l)

total\_dirs=$(find $i -type d | wc -l)

total\_size=$(du -sh $i | cut -f 1)

echo "<H3>$i</H3>"

echo "<PRE>"

printf "$format" "Dirs" "Files" "Size"

printf "$format" "----" "-----" "----"

printf "$format" $total\_dirs $total\_files $total\_size

echo "</PRE>"

done

return

}

usage () {

echo "$PROGNAME: usage: $PROGNAME [-f file | -i]"

return

}

write\_html\_page () {

cat <<- \_EOF\_

<HTML>

<HEAD>

<TITLE>$TITLE</TITLE>

</HEAD>

<BODY>

<H1>$TITLE</H1>

<P>$TIME\_STAMP</P>

\_EOF\_

if [ "$1" == "1" ]; then

cat <<- \_EOF\_

$(report\_uptime)

\_EOF\_

fi

if [ "$2" == "1" ]; then

cat <<- \_EOF\_

$(report\_disk\_space)

\_EOF\_

fi

if [ "$3" == "1" ]; then

cat <<- \_EOF\_

$(report\_home\_space)

\_EOF\_

fi

cat <<- \_EOF\_

</BODY>

</HTML>

\_EOF\_

return

}

# обработка параметров командной строки

filename="taks3.html"

# вывод страницы html

if [[ -n $filename ]]; then

if touch $filename && [[ -f $filename ]]; then

choice=$(dialog --menu "Выберите элементы из списка" 15 50 6 \

1 "report uptime" \

2 "report disk space" \

3 "report home space" \

4 "usage" \

5 "View all the information" \

6 "Выход" 3>&1 1>&2 2>&3)

case $choice in

1)

write\_html\_page "1" "0" "0" > $filename

;;

2)

write\_html\_page "0" "1" "0" > $filename

;;

3)

write\_html\_page "0" "0" "1" > $filename

;;

4)

usage

exit 1

;;

5)

write\_html\_page "1" "1" "1" > $filename

;;

6)

exit 1

;;

esac

xdg-open "$filename"

else

echo "$PROGNAME: Cannot write file '$filename'" >&2

exit 1

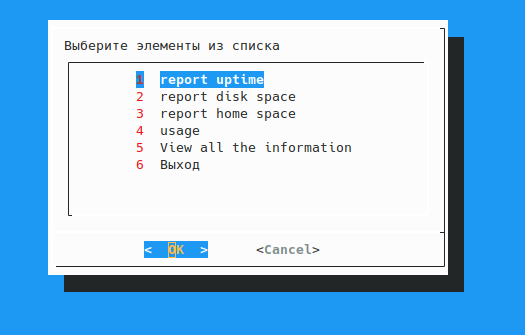
fi

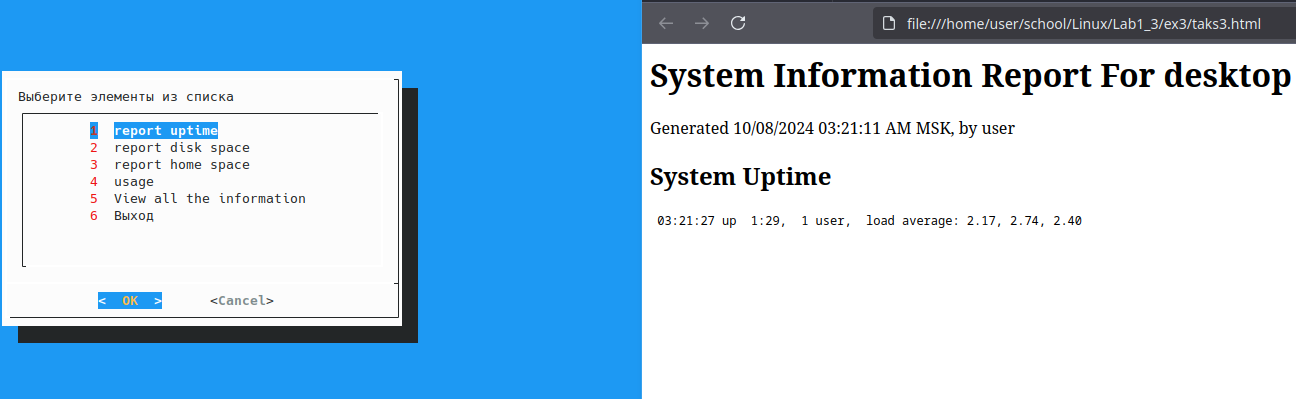
else

write\_html\_page

fi

Результат**:**





**Ответы на контрольные вопросы:**

**Возможности командной оболочки и скриптов Shell**

**1) Что такое переменный и константы, чем они отличаются? Как их**

**различает командная оболочка?**

 Переменные - это именованные области памяти, которые могут изменять свои значения в процессе выполнения программы. Они используются для хранения данных, которые могут изменяться. Константы, напротив, имеют фиксированные значения, которые не изменяются на протяжении выполнения программы. В командной оболочке переменные создаются с помощью команды = (например, VAR=value), а константы обычно задаются с помощью readonly или declare -r. Командная оболочка различает их по способу объявления и использованию. Например, переменная может быть изменена в любой момент, тогда как константа останется неизменной. Это различие важно для предотвращения случайных изменений данных, которые должны оставаться постоянными.

**2) Как командная оболочка определяет тип значения переменных?**

В командной оболочке тип значения переменной определяется автоматически на основе присвоенного значения. Например, если переменной присвоено числовое значение, она будет интерпретироваться как число. В Bash можно использовать команду declare для явного указания типа переменной (например, declare -i для целых чисел). Это позволяет избежать ошибок, связанных с неправильным типом данных. Командная оболочка также поддерживает строки и массивы, что делает её гибкой для различных задач. Автоматическое определение типа упрощает работу с переменными, так как не требует явного указания типа при каждом присвоении значения.

**3) Какое типичное соглашение о написании имен переменных и**

**констант принято программистами? Для чего это делается?**

Программисты часто используют соглашения для именования переменных и констант, чтобы улучшить читаемость кода. Например, имена переменных пишутся в нижнем регистре с подчеркиваниями (my\_variable), а константы - в верхнем регистре (MY\_CONSTANT). Это помогает быстро различать переменные и константы в коде. Соглашения также могут включать использование префиксов для обозначения типа данных или области видимости переменной. Например, префикс g\_ может указывать на глобальную переменную. Такие соглашения делают код более понятным и облегчают его сопровождение.

**4) Что такое переменные окружения? Назовите команды с помощью**

**которых можно получить значения переменных окружения?**

 Переменные окружения - это переменные, которые определяют поведение операционной системы и приложений. Они могут быть получены с помощью команд printenv, env или echo $VARIABLE\_NAME. Примеры переменных окружения включают PATH, HOME, USER. Эти переменные используются для настройки среды выполнения программ и скриптов. Например, переменная PATH определяет пути к исполняемым файлам, что позволяет запускать программы без указания полного пути. Переменные окружения могут быть изменены пользователем для настройки системы под свои нужды.

**5) Что такое пользовательские переменные и для чего они**

**используются?**

Пользовательские переменные создаются пользователем для хранения данных, которые могут быть использованы в скриптах и командах. Они помогают автоматизировать задачи и сохранять временные данные. Например, MY\_VAR="Hello" создаст переменную MY\_VAR с значением “Hello”. Пользовательские переменные могут быть использованы для передачи данных между различными частями скрипта или для хранения результатов промежуточных вычислений. Они также могут быть использованы для настройки параметров выполнения скрипта, что делает их важным инструментом для автоматизации.

**6) Что такое область видимости переменных? Какие типы области**

**видимости переменных существуют?**

Область видимости переменных определяет, где переменная может быть использована. Существуют глобальные переменные, доступные во всей программе, и локальные переменные, доступные только в пределах функции или блока кода. В Bash локальные переменные создаются с помощью команды local. Это позволяет избежать конфликтов имен и случайных изменений переменных в разных частях программы. Глобальные переменные могут быть полезны для хранения данных, которые должны быть доступны везде, но их использование должно быть ограничено для предотвращения ошибок.

**7) Что такое конфликт имен переменных и как решается эта проблема?**

Конфликт имен переменных возникает, когда две переменные имеют одинаковые имена, но разные значения или области видимости. Эта проблема решается использованием уникальных имен или локальных переменных для ограничения области видимости. Например, можно использовать префиксы или суффиксы для создания уникальных имен переменных. Также можно использовать локальные переменные внутри функций, чтобы ограничить их область видимости и избежать конфликтов с глобальными переменными. Это помогает поддерживать чистоту и читаемость кода.

**8) Что такое функции командной оболочки?**

Функции в командной оболочке - это блоки кода, которые могут быть вызваны по имени для выполнения определенных задач. Они помогают структурировать код и повторно использовать его. Функции объявляются с помощью ключевого слова function или просто имя\_функции(). Внутри функции можно использовать локальные переменные, что позволяет избежать конфликтов имен. Функции могут принимать аргументы и возвращать значения, что делает их мощным инструментом для создания сложных скриптов. Использование функций улучшает читаемость и сопровождение кода.

**9) Что такое «код завершения программы»?**

Код завершения программы - это числовое значение, возвращаемое программой после завершения работы. Он указывает на успешное или неуспешное выполнение программы. В командной оболочке код завершения можно получить с помощью переменной $?. Значение 0 обычно указывает на успешное выполнение, а любое другое значение - на ошибку. Код завершения может быть использован для управления потоком выполнения скрипта, например, для выполнения определенных действий в случае ошибки. Это позволяет создавать более надежные и устойчивые к ошибкам скрипты.

**10) Какой командой и каким средством в скриптах обеспечивается**

**интерактивность программы?**

Интерактивность в скриптах обеспечивается с помощью команд read для получения ввода от пользователя и echo для вывода сообщений. Это позволяет создавать диалоговые окна и запрашивать данные у пользователя. Например, команда read -p "Введите ваше имя: " name запрашивает у пользователя ввод имени и сохраняет его в переменной name. Интерактивные скрипты могут использоваться для настройки параметров выполнения или для получения данных, необходимых для выполнения задач. Это делает скрипты более гибкими и удобными для пользователя.

**Графический интерфейс диалога**

**11) С помощью каких утилит реализуется средства примитивного**

**графического интерфейса в Linux-скриптах?**

Для создания примитивного графического интерфейса в Linux-скриптах используются утилиты dialog, whiptail и zenity. Они позволяют создавать диалоговые окна, меню и формы для взаимодействия с пользователем. Например, утилита dialog может использоваться для создания меню выбора, окон ввода и сообщений. Утилита zenity позволяет создавать графические окна с использованием GTK. Эти инструменты делают скрипты более удобными и интуитивно понятными для пользователя, что улучшает взаимодействие с программой.

**12) Укажите какими интернет-источниками вы пользовались при**

**изучении и разработке графического диалога.**

При изучении и разработке графического диалога можно использовать официальную документацию утилит, такие как man страницы, а также онлайн-ресурсы, такие как Stack Overflow, GitHub и специализированные форумы. Эти ресурсы предоставляют примеры кода, советы и решения распространенных проблем. Также можно найти учебные материалы и руководства, которые помогут освоить создание графических интерфейсов в скриптах. Использование различных источников информации позволяет получить более полное представление о возможностях и особенностях утилит.

**13) Какие существуют типы окон примитивного графического**

**интерфейса Linux-скриптов?**

Существуют различные типы окон для примитивного графического интерфейса, включая диалоговые окна с сообщениями, окна ввода, меню, флажки и радиокнопки. Эти окна позволяют создавать простые и удобные интерфейсы для взаимодействия с пользователем. Например, диалоговые окна с сообщениями могут использоваться для отображения информации или предупреждений, а окна ввода - для получения данных от пользователя. Меню позволяют пользователю выбирать из нескольких вариантов, а флажки и радиокнопки - для выбора одного или нескольких параметров. Эти элементы делают взаимодействие с программой более интуитивным и удобным.