1. //Напишите скрипт на bash, который принимает на вход один аргумент (целое число от 0 до бесконечности), который будет обозначать число студентов в аудитории. В зависимости от значения числа нужно вывести разные сообщения.

if [[ $1 -eq 1 ]]; //если значение переменной $1 равно 1, то выполняется код внутри блока then

then

echo "$1 student"//вывод значения переменной $1 и слова «students»

elif [[ $1 -gt 1 && $1 -le 4 ]]; //если значение переменной $1 больше или равно 5, то выполняется код внутри блока else

then

echo "$1 students"

elif [[ $1 -ge 5 ]];

then

echo "A lot of students"

else

echo "No students"

fi//завершение условного оператора if

1. сколько раз на экран будет выведено слово "start", а сколько раз слово "finish"?

for str in a , b , c\_d

do

echo "start"

if [[ $str > "c" ]]//проверка, больше ли значение переменной str символа «c»

then

continue//переход к следующей итерации цикла, если значение переменной str больше символа «c»

fi

echo "finish"

done

#!/bin/bash

for str in a , b , c\_d #для значений str в <a>,<,>,<b>,<,>,<c\_d>

do

echo "$str start" #выводим в старт str с КАЖДЫМ подставленным значением

if [[ $str > "c" ]] #если значение str больше С

then #тогда

echo "$str больше с" #выводим сообщение, что str больше с

continue #и продолжаем подставлять значения

fi #если условие не сработало

echo "$str не больше с" #выводим сообщение, что str не больше с

echo "$str finish" #выводим в финиш str с ЭТИМ значением

done #завершаем цикл

#делаем вывод, что старт будет выводиться всегда, а финиш только

#в том случае, когда значение str будет не больше с

1. // Получение времени (Системный вызов)

SYSCALL\_DEFINE1(time, \_\_kernel\_old\_time\_t \_\_user \*, tloc)

{

\_\_kernel\_old\_time\_t i = (\_\_kernel\_old\_time\_t)ktime\_get\_real\_seconds();

if (tloc) {

if (put\_user(i,tloc))

return -EFAULT;

}

force\_successful\_syscall\_return();

return i;

}

1. // Например, возьмем приложение, которое будет ждать примерно 5 секунд:

.globl \_start

.data

curtime: .quad 0 # для хранения времени

.text

\_start:

# получаем начальное время

movq $0xc9, %rax # номер системной функци

movq $curtime, %rdi # адрес переменной для получения времени

syscall # выполняем систеиный вызов

movq curtime, %rdx # сохраняем полученное время в %rdx

addq $5, %rdx # добавляем 5 секунд

timeloop:

movq $0xc9, %rax # проверяем время

movq $curtime, %rdi

syscall

# если мы не достигли времени в %rdx, переходим обратно к timeloop

cmpq %rdx, curtime

jb timeloop

exit:

movq $0, %rdi

movq $60, %rax

syscall

1. Системный вызов приостанавливает выполнение программы и передает контроль ядру операционной системы. Операционная система проверяет валидность вызова и права приложения, выполняет соответствующую системную функцию и затем передает управление обратно процессу приложения.

.globl \_start

.text

\_start:

movq $22, %rdi # в RDI код статуса результата

movq $60, %rax # в RAX номер системной функции

syscall # выполняем системную функцию

vim variables1.sh

#!/bin/bash// указывает на то, что данный скрипт должен выполняться с помощью bash.

dir\_path=~/Linux\_course/scripting/test\_dir/ //присваивает значение

file\_path=${dir\_path}test\_file.txt

echo "Creating file $file\_path"//выводит сообщение о том, что создаётся файл с именем $file\_path

touch $file\_path//создаёт пустой файл с именем

echo "Checking content of dir $dir\_path"

ls $dir\_path//выводит список файлов и каталогов в каталоге

Esc // выход из режима вставки текста в Vim.

:wq // сохранение изменений и выход из редактора Vim.

bash ./variables1.sh //запускает скрипт

vim variables2.sh

#!/bin/bash // указывает на то, что данный скрипт должен выполняться с помощью bash.

var1="Hello, world!"//присваивается значение Hello world

var2="var1 is $var1"//присваивается ей строка "var1 is Hello, world!".

echo '$var2'//Выводится строка '$var2'('' не позволяют интерпретировать переменные)

echo "$var2"//Выводится строка "$var2"("" позволяют интерпретировать переменные)

var2='var1 is $var1' //присваивается строка 'var1 is $var1'

echo '$var2'//выводится строка

echo "$var2"

Esc // выход из режима вставки текста в Vim.

:wq // сохранение изменений и выход из редактора Vim.

bash ./variables2.sh //запускает скрипт

vim variables3.sh

#!/bin/bash // указывает на то, что данный скрипт должен выполняться с помощью bash.

variable1=$1 // присваивает переменной variable1 значение первого аргумента, переданного скрипту при его запуске.

variable2=$2 //присваивает переменной variable2 значение второго аргумента, переданного скрипту при его запуске.

echo "Arguments are: \$1=$variable1 \$2=$variable2" //выводит сообщение на экран, в котором отображаются значения двух переданных аргументов скрипту.

Esc // выход из режима вставки текста в Vim.

:wq // сохранение изменений и выход из редактора Vim.

bash ./variables3.sh //запускает скрипт

bash// указывает на то, что данный скрипт должен выполняться с помощью bash

vim script1.sh

#!/bin/bash

echo "Going to home!"// выводит сообщение «Going to home!»

cd ~//команда для перехода в домашний каталог

pwd//команда для вывода текущего рабочего каталога

ls//команда для вывода списка файлов и каталогов в текущем рабочем каталоге

Esc // выход из режима вставки текста в Vim.

:wq // сохранение изменений и выход из редактора Vim.

bash ./script1.sh //запускает скрипт

gcc ./hello\_world.c //запуск кода

chmod +x a.out

//запуск файла на исполнение

grep [OPTIONS] PATTERNS [FILE]

//grep выполняет поиск в файле

grep "127.0." /etc/hosts

//найдем текст "127.0." в файле /etc/hosts

ls ~ | grep "Do"

//паки на Do

awk -F : '$2 > 30 {print $1}' users.txt // выведем пользователей, у которых возраст больше 30

awk '/Tom/' users.txt //получить все строки, где имя пользователя - "Tom"

touch hello //создание текстового файла

pwd //показывает текущую директорию

Для установки Man-страниц

Для Ubuntu:

sudo apt install man-db manpages

или:

sudo apt install manpages-dev manpages-posix-dev

Какую информацию можно почерпнуть из структуры dirent: man readdir

man 2 fctnl //Интерфейс управления:

Освободить выделенные ресурсы: man closedir

Команда free – это самая простая и понятная в использовании команда для проверки использования памяти в Linux: $ free -h

Команда top показывает потребление ресурсов в режиме реального времени.

Она может отображать сводную информацию о системе, а также список задач, которыми в данный момент управляет ядро Linux: $ top

Oсновные команды для linux

ls - список файлов и папок в текущем каталоге

cd - перейти в другую директорию

pwd - показать текущую директорию

mkdir - создать новую директорию

rm - удалить файл или папку

cp - скопировать файл или папку

mv - переместить или переименовать файл или папку

touch - создать новый файл

cat - показать содержимое файла

grep - найти строки, содержащие указанный текст

ps - показать список активных процессов

kill - завершить процесс

chmod - изменить права доступа к файлу

chown - изменить владельца файла

top - показать список процессов, использующих наибольшее количество ресурсов