Лабораторная работа №10. Программирование в командном процессоре ОС имх

# 1. Титульный лист

* Номер лабораторной работы: 10
* ФИО студента: Шилов Александр Ильич

# 2. Формулировка цели работы

Цель работы: Изучение принципов программирования в командном процессоре Unix, создание скриптов для автоматизации задач и обработки данных.

З. Описание результатов выполнения задания

Задание 1: Скрипт для резервного копирования

• Код скрипта:

## /bin/bash

 скрипт для резервного копирования файлов

 Проверка наличия аргументов if [ $# -1t 2 ] ; then echo ”Использование: $6 <ИСХОДНЫЙ каталог > «аталог\_назначения>” exit 1

# получение аргументов

### SOURCE DIR=$1



### ARCHIVE ГШМЕ=

DATE}.tar.gz”

# Проверка существования исходного каталога if ! -d ”$SOURCE DIR” ] ; then echo ”0шибка: исходный каталог '$SOURCE DIR' не существует. ' exit 1

4$ создание каталога назначения, если он не существует

if ! -d  ] ; then mkdir -р

. -пе е ] ; then echo ”0шибка: не удалось создать каталог назначения '$DEST О:

exit 1 echo ”Создан каталог назначения: $DEST DIR”

4$ создание архива echo ”Создание резервной копии из '$SOURCE DIR' в '$DEST\_DIR/$ARCHIVl tap -czf ”$DEST DIR/$ARCHIVE ГЫДМЕ” -С ”$SOURCE DIR” 

# Проверка успешности создания архива if -eq е ] ; then echo ”Резервное копирование успешно завершено. ” echo ”Архив сохранен как: $DEST\_DIR/$ARCHIVE ГШМЕ” echo ”Размер архива: $(du -h ”$DEST DIR/$ARCHIVE ГШМЕ” cut \_f1



echo ”0шибка: не удалось создать архив. ” exit 1

## exit

• Результат выполнения:

### $ ./backup.sh test\_source test\_backup

создание резервной копии из 'test\_source' в 'test ЬасКир/ЬасКир\_2Э2Ы

Резервное копирование успешно завершено.

Архив сохранен как: test ЬасКир/ЬасКир\_2625Э566 952146.tar.gz

Размер архива: 4. ек

Задание 2: Командный файл для обработки аргументов

• Код скрипта:

### /bin/bash

# скрипт для обработки аргументов командной строки

echo ”0бщее количество аргументов: $#” echo ”Имя скрипта: 9”

# вывод всех аргументов echo ” все аргументы: $@”

 вывод аргументов по отдельности echo ”Аргументы по отдельности: ” count=1 fop arg in ”$@"; do echo ”Аргумент $count: $apg” count=$((count + 1) ) done

4$ вывод аргументов в обратном порядке echo ”Аргументы в обратном порядке: ' fop (( i=$#; i>e; i-- ) ) ; do echo ”Аргумент $1: ${ ! i}” done

## exit

• Пример вывода:

$ ./args.sn опе two three four общее количество аргументов: 4 имя скрипта: ./apgs.sh все аргументы: опе two three four

Аргументы по отдельности:

Аргумент 1: опе

Аргумент 2: two

Аргумент З: thpee

Аргумент 4: four

Аргументы в обратном порядке:

Аргумент 4: four

Аргумент З: thpee

Аргумент 2: two

Аргумент 1: опе

Задание З: Аналог команды

• Код скрипта:

### /bin/bash

# скрипт-аналог команды 1s

# Функция для вывода справки show\_he1p() { echo ”Использование: $6 [опции] [каталог] ” echo ”ОПЦИИ : ”

|  |  |  |
| --- | --- | --- |
| echo |  | показать все файлы, включая скрытые“ |
| echo | -1 | Подробный формат вывода ” |
| echo | -h | показать эту справку“ |

## exit

# инициализация переменных show hidden=e



dipectopy=” . ”

# 06pa60TKa or-IUVIVI while getopts "alh" opt; do case $opt in

a) show\_hidden=l , 1) long\_format=l

### h) show\_help ,

\ ? ) echo "HeV13BecTHaq Olli-WIA : -$OPTARG" exit 1 esac done

# CABV1r 06Pa60TaHHblX onuhVl shift $ ( (OPTIND 1) )

# Ecm•l y.'Ka3aH KaTanor, ucn0J1b3yeM ero if [ $# -gt e ] ; then directory= " $1 "

# np0BepKa CyueCTBOBaHVIA KaTanora if ! -d "$directory" ] ; then echo "0UJh6Ka: KaTanor '$directory' He cyuecTByeT.' exit 1

## # nonyqeHL,le (ÞaVIJIOB if [ $show\_hidden -eq 1 ] ; then files=($(ls -a "$directory"))

### files=($(ls " $directory" ) )

# BblBOA pe3YJ1bTaT0B echo "Coaepxmqoe KaTanora: $directory" echo

## for file in "${files[@]}", do

# nponycl<aeM . . ecm•l He nol<a3blBaeM CKPblTble if [ $show\_hidden -eq e ] && [ [ "$file" I l "$file" continue

### $directory/$file"

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | if [ $long\_format -eq 1 ] ;  # Tnn 4)ahna | | then | | | | if [ -d "$full\_path" ] ; type= "d" | | then | | | | elif -L "$full\_path" | |  | then | | | type=  type='  # npaBa aocTyna permissions= |  |  |  | | if [ -r "$full\_path" |  | then | | permissions="${permissions}r" , | | if [ -w "$full\_path" |  | then | | permissions="${permissions}w" ; | | if [ -x "$full\_path" |  | then | | permissions="${permissions}x" , | |  |
|  |
|  |

#### # Pa3Mep VI aaTa MOAV14)VlKaUVIV1 size=$(du -h "$full\_path" 2>/dev/nu11 1 cut -fl) mod date=$(date -r "$full\_path" "+%Y-%m-%d %H : YOM" )

echo "$type$permissions $size $mod\_date $file"

echo "$file"

done

## exit

3aaaHne 4: noacuer •ainoB no •opMaTarv1

• KOA cKpnnTa:

### / bin/ bash

# скрипт для подсчета файлов по форматам

# Проверка наличия аргумента if [ $# -1t 1 ] ; then echo ”Использование: $6 «аталог>” exit 1

4$ получение аргумента



4$ Проверка существования каталога if ! -d ”$dipectopy” ] ; then echo ”0шибка: каталог '$directory' не существует. ' exit 1

echo ”Подсчет файлов по форматам в каталоге: $dipectopy” echo

# получение списка всех файлов рекурсивно fi1es=$(find ”$directory” -type .f sopt)

# инициализация ассоциативного массива для подсчета dec1are -д formats count

4$ подсчет файлов по расширениям fop fi1e in $fi1es; do

# получение расширения файла extension=”  . } ”

# Если файл не имеет расширения, считаем его как ” без расширения if [ ”$extension” ”$fi1e” ] ; then extension=”6e3 расширения ”

extension=” .$extension'

# Увеличиваем счетчик для данного расширения

### if [ -z "${formats count[$extension]}" then formats count[$extension]=l

formats count[$extension]=$((formats count[$extension] + 1))

done

# BblBOA pe3YJ1bTaT0B echo "(DopMaT I KOJIV1qeCTBO (þaVlJIOB" echo for format in !  ; do count=${formats count[$format]}

# npaBL,1J1bHoe CKJIOHeHV1e CJIOBa "(þaün" if [ $count -eq 1 ] ; then word='þahn " el if [ $count -ge 2 ] && [ $count -le 4 ] ; then word='þahna"



echo " $format : ${formats count[$format]} $word" done

# 06uee KOJIV1qeCTBO 4)aVIJIOB total=e for count in "${formats ; do

### total=$((total + count)) done

echo echo " Bcero: $total #VIJIOB"

## exit

• npv•wep BblBOAa:

$ ./countfiles.sh подсчет файлов по форматам в каталоге: .

Формат Количество файлов

. txt : З файла

4 файла

.md: 1 файл . gz : 1 файл

всего: 9 файлов

# 4. Выводы

1. Я освоил основы создания скриптов на Bash, включая работу с переменными, условными операторами и циклами.
2. Научился работать с аргументами командной строки и обрабатывать их в скриптах.

З. Изучил механизм архивации файлов в Unix с помощью команды tar.

1. Освоил работу с файловой системой: проверку существования файлов и каталогов, получение информации о файлах.
2. Научился использовать ассоциативные массивы для подсчета и группировки данных.

б. Понял принципы создания пользовательских аналогов стандартных команд Unix.

5. Ответы на контрольные вопросы

1 . Что такое командная оболочка?

Командная оболочка (shell) - это программа, которая предоставляет интерфейс между пользователем и операционной системой. Она интерпретирует команды пользователя и передает их ядру ОС для выполнения. Оболочка также предоставляет язык программирования для написания скриптов.

2. что такое роях?

POSIX (Portable 0perating System Interface) - это набор стандартов, определяющих интерфейс между операционной системой и прикладными программами. Эти стандарты обеспечивают совместимость между различными ИГЧЖ-подобными операционными системами, позволяя программам работать на разных платформах без изменений.

З. Каковы основные возможности командных оболочек? Основные возможности командных оболочек включают:

* Выполнение команд и программ
* Перенаправление ввода/вывода
* Конвейерная обработка данных (pipes)
* Управление заданиями (jobs)
* Подстановка имен файлов (wildcards)
* Программирование с использованием переменных, условий, циклов
* Обработка аргументов командной строки
* Автодополнение команд и имен файлов
* История команд

1. Что такое переменная окружения?

Переменная окружения - это динамическая переменная в операционной системе, которая содержит информацию, используемую одной или несколькими программами. Переменные окружения определяют поведение системы и программ, например, РАТН определяет каталоги для поиска исполняемых файлов, НОМЕ указывает на домашний каталог пользователя.

1. Каково назначение команд set и unset?
   * set используется для установки или отображения переменных оболочки. Без аргументов выводит список всех переменных. С опциями может изменять поведение оболочки.
   * unset используется для удаления переменных или функций из окружения оболочки.

б. Как определяются функции в языке программирования bash? Функции в bash определяются следующим образом:

function\_name() { # команды peturn va1ue # необязательно

или

## function function пате {

# команды peturn va1ue # необязательно

1. Каково назначение команд let и read?
   * 1et выполняет арифметические операции над переменными, например:

1et ” а  read считывает строку из стандартного ввода и присваивает ее переменным, например: read пате - считывает строку и сохраняет в переменной пате.

1. Что такое позиционные параметры?

Позиционные параметры - это специальные переменные в скриптах, которые содержат аргументы командной строки. $0 содержит имя скрипта, $1 - первый аргумент, $2 - второй и т.д. $# содержит количество аргументов, $@ и $\* содержат все аргументы.

1. Как осуществляется возврат значений из функций? Функции в bash возвращают значения через:
   * Код возврата (0-255) с помощью команды peturn
   * Вывод результата в stdout с помощью echo или ppintf, который можно захватить при вызове функции
   * Изменение глобальных переменных внутри функции
2. Что такое локальные переменные?

Локальные переменные - это переменные, объявленные с ключевым словом

1оса1 внутри функции. Они видны только внутри функции, в которой объявлены, и не влияют на одноименные переменные вне функции.

1 1 . Что такое рекурсия и как она реализована в языке программирования bash?

Рекурсия - это процесс, при котором функция вызывает сама себя. В bash рекурсия реализуется так же, как и в других языках - функция содержит вызов самой себя с измененными параметрами и условие ВЫХОДа из рекурсии:

## factoria1() { then echo 1

1оса1  $(( $1 echo $(( $1 \* temp ) )

1. Что такое командная подстановка?

Командная подстановка - это механизм, позволяющий использовать вывод одной команды как аргумент для другой команды или для присваивания переменной. Реализуется с помощью обратных кавычек 'command' или синтаксиса $(command).

1. Как создаются и используются массивы в языке программирования bash?

Массивы в bash создаются и используются следующим образом:

# создание массива

”

” e1ement1 ”e1ement2” ”e1ement3” )

# доступ к элементам

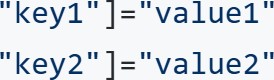
echo # первый элемент echo # все элементы

# добавление элемента аррау+= ( ”e1ement4” )

# количество элементов echo

Ассоциативные массивы (с версии Bash 4.0):

# объявление ассоциативного массива dec1are -д assoc аррау

4$ заполнение assoc аррау[ assoc аррау[

# доступ к элементам echo ${assoc

# все ключи echo ${ !assoc

1. Что такое регулярные выражения?

Регулярные выражения - это шаблоны, используемые для поиска и обработки текста. Они представляют собой последовательность символов, определяющую правила поиска. В Unix регулярные выражения используются во многих утилитах, таких как grep, sed, awk, и в языках программирования, включая bash.

1. Как осуществляется обработка ошибок в языке программирования bash?

Обработка ошибок в bash осуществляется следующими способами:

* + Проверка кода возврата команд с помощью
  + Использование условных операторов для проверки успешности выполнения команд
  + Использование конструкции set -е для автоматического завершения скрипта при ошибке
  + Использование конструкции trap для перехвата сигналов и выполнения действий при ошибках
  + Перенаправление вывода ошибок с помощью 2> или 2>&1