**Лабораторная работа № 3 Программирование в Shell**

**Цель:** изучить основы программирования на языке Shell, приобрести практические навыки по созданию пакетных исполняемых файлов.

**Теория:**

**1. ОПЕРАТОРЫ – КОМАНДЫ**

Язык **shell** по своим возможностям приближается к высокоуровневым алгоритмическим языкам программирования. Операторы языка **shell** позволяют создавать собственные программы, не требуют компиляции, построения объектного файла и последующей компоновки, так как shell, обрабатывающий их, является транслятором интерпретирующего, а не компилирующего типа.

Текст процедуры набирается как обычный текстовый файл. Проверенный и отлаженный shell-файл может быть вызван на исполнение, например, следующим способом:

$ chmod u+x shfil

$ shfil

$

Такая форма предполагает, что файл процедуры новый и его надо сначала сделать выполняемым. Можно использовать также и следующий способ:

$ sh -c "shfil" или $ sh shfil

В этих случаях по команде sh вызывается вторичный интерпретатор shell, и в качестве аргумента ему передается командная строка, содержащая имя файла процедуры shfil, находящегося в текущем каталоге. Однако, этот способ накладывает ограничения на исполнение некоторых команд ОС управления процессами.

Процедуре при ее запуске могут быть переданы аргументы. В общем случае командная строка вызова процедуры имеет следующий вид:

$ имя\_процедуры $1 $2 ...$9

Каждому из девяти первых аргументов командной строки в тексте процедуры соответствует один из позиционных параметров: $1, $2, ..., $9 соответственно. Параметр $0 соответствует имени самой процедуры, т.е. первому полю командной строки. К каждому из 10 первых аргументов можно обратиться из процедуры, указав номер его позиции. Количество аргументов присваивается другой переменной: **$#**(диез).

Сам интерпретатор shell автоматически присваивает значения следующим переменным (параметрам):

*? значение, возвращенное последней командой;*

*$ номер процесса;*

*! номер фонового процесса;*

*# число позиционных параметров, передаваемых в shell;*

*\* перечень параметров, как одна строка;*

*@ перечень параметров, как совокупность слов;*

*- флаги, передаваемые в shell.*

При обращении к этим переменным (т.е при использовании их в командном файле - shell-программе) следует впереди ставить "$".

**Пример**.

Вызов фала – specific par1 par2 par3 имеющего вид:

echo $0 - имя расчета

echo $? - код завершения

echo $$ - идентификатор последнего процесса

echo $! - идентификатор последнего фонового процесса

echo

echo $\* - значения параметров, как строки

echo $@ - значения параметров, как слов

echo

set -au

echo $- - режимы работы интерпретатора

Выдаст на экран:

specific - имя расчета

0 - код завершения

499 - идентификатор последнего процесса

98 - идентификатор последнего фонового процесса

par1 par2 par3 - значения параметров, как строки

par1 par2 par3 - значения параметров, как слов

au - режимы работы интерпретатора

Некоторые вспомогательные операторы:

echo - вывод сообщений из текста процедуры на экран.

$ echo "начало строки

> продолжение строки"

**# - для обозначения строки комментария в процедуре.** (Строка не будет обрабатываться shell-ом)**.**

**banner - вывод сообщения на экран заглавными буквами (например для идентификации следующих за ним сообщений).**

$banner 'hello ira'

HELLO IRA

$

**Простейший пример**.

Здесь оператор **echo** выполняется в командном режиме.

$shfil p1 pp2 petr

$echo $3

petr

$

**Пример**

Передача большого числа параметров

echo "$0: Много параметров"

echo " Общее число параметров = $#

Исходное состояние: $1 $5 $9 "

shift

echo "1 сдвиг: первый=$1 пятый=$5 девятый=$9"

shift 2

echo "1 + 2 = 3 сдвига: первый=$1 пятый=$5 девятый=$9"

perem=`expr $1 + $2 + $3`

echo $perem

Значения параметрам, передаваемым процедуре, можно присваивать и в процессе работы процедуры с помощью оператора

**set - присвоить значения позиционным параметрам;**

$set a1 ab2. abc

$echo $1 $2

a1 ab2 - в этом примере параметры указываются в явном виде.

$

Запуск **set** без параметров выводит список установленных системных переменных:

HOME=/home/sae

PATH=/usr/local/bin:/usr/bin:/bin:.:/usr/bin/X11:

IFS=

LOGNAME=sae

MAIL=/var/spool/mail/sae

PWD=/home/sae/STUDY/SHELL

PS1=${PWD}:" "

PS2=>

SHELL=/bin/bash

TERM=linux

TERMCAP=console|con80x25|dumb|linux:li#25:co#80::

UID=501

Количество позиционных параметров может быть увеличено до необходимого значения путем "сдвига" их в командной строке влево на одну позицию с помощью команды shift без аргументов:

shift - сдвинуть позиционные параметры влево на одну позицию

После выполнения shift прежнее значение параметра $1 теряется, значение $1 приобретает значение $2, значение $2 - значение $3 и т.д..Продолжение предыдущего примера:

$shift

$echo $1 $2

ab2 abc

$

**В UNIX при написании операторов важное значение отводится кавычкам (апострофам):**

'...' - для блокирования специальных символов, которые могут быть интерпретированы как управляющие;

"..." - блокирование наиболее полного набора управляющих символов или указания того, что здесь будет обрабатываться не сам аргумент, а его значение;

`...` - (обратные кавычки или знак ударения) для указания того, что они обрамляют команду и здесь будет обрабатываться результат работы этой команды (подстановка результатов работы указанной команды).

**Пример 1.**

$ date

Apr 3 14:27:07 2005

$ set `date`

$ echo $3

14:30:25

$

**Пример 2.**

$echo `ls`

fil.1

fil.2

...

$echo '`ls`'

# одинарные кавычки блокируют действие обратных кавычек

# т.е. они распечатываются как обычные символы

`ls`

$

Для ввода строки текста со стандартного устройства ввода используется оператор:

read имя1 [имя2 имя3 .] - чтение строки слов со стандартного ввода

Команда вводит строку, состоящую из нескольких полей (слов), со стандартного ввода, заводит переменную для каждого поля и присваивает первой переменной имя1, второй переменной - имя2, и т.д. Если имен больше, чем полей в строке, то оставшиеся переменные будут инициализированы пустым значением. Если полей больше, чем имен переменных, то последней переменной будет присвоена подстрока введенной строки, содержащая все оставшиеся поля, включая разделители между ними. В частности, если имя указано только одно, то соответствующей ему переменной присваивается значение всей строки целиком.

**Пример:**

#Текст процедуры:

echo "Введите значения текущих: гг мм ччвв"

read 1v 2v 3v

echo "год 1v"

echo "месяц 2v"

echo "сегодня 3v"

# здесь кавычки используются для блокирования пробелов

#Результат выполнения процедуры:

Введите значения текущих: гг мм ччвв

2005 Maрт 21 9:30 <Enter>

год 2005

месяц Maрт

сегодня 21 9:30

**2 УПРАВЛЕНИЕ ЛОКАЛЬНЫМИ ПЕРЕМЕННЫМИ**

В отличии от рассмотренных в начале курса системных переменных среды, переменные языка shell называются **локальными переменными** и используются в теле процедур для решения обычных задач. Локальные переменные связаны только с породившим их процессом. Локальные переменные могут иметь имя, состоящее из одного или нескольких символов. Присваивание значений переменным осуществляется с помощью известного оператора:

**"=" - присвоить (установить) значение переменной.**

При этом если переменная существовала, то новое значение замещает старое. Если переменная не существовала, то она строится автоматически shell. Переменные хранятся в области ОП - области локальных данных.

$count=3

$color=red belt

$fildir=lev/d1/d12

$

Еще пример:

# текст процедуры

b=”1 + 2”

echo c=$b

#в результате выполнения процедуры выводится текст,

# включающий текст переменной b

c=1+2

**3 ПОДСТАНОВКА ЗНАЧЕНИЙ ПЕРЕМЕННЫХ**

Процедура подстановки выполняется shell-ом каждый раз когда надо обработать значение переменной если используется следующая конструкция: **первый вид подстановки**

**$имя\_переменной** -. на место этой конструкции будет подставлено значение переменной.

Примеры подстановки длинных маршрутных имен:

$echo $HOME

/home/lev

$filname=$HOME/f1

$more $filname

<текст файла f1 из каталога lev>

$

Использование полного маршрутного имени в качестве значения переменной позволяет пользователю независимо от местонахождения в файловой системе получать доступ к требуемому файлу или каталогу.

Если в строке несколько присваиваний, то последовательность их выполнения - справа налево.

|  |  |  |
| --- | --- | --- |
| Пример 1.  $ z=$y y=123  $ echo $z $y  123 123  $ y=abc z=$y  $ echo “z”  123  $ echo “y”  abc  $ | Пример 2.  $ var=/user/lab/ivanov  $ cd $var  $ pwd  /udd/lab/ivanov  $ | Пример 3.  $ namdir='ls'  $ $namdir  fil1  fil2  fil3  ...  $ |

В последнем примере переменной **namdir** присвоено значение, которое затем используется в качестве командной строки запускаемой команды. Это команда ls.

**Второй вид подстановки** – подстановка результатов работы команды вместо самой команды.

**Пример 4.**

В данном случае команда ls непосредственно выполняется уже в первой строке, и переменной **filnam** присваивается результат ее работы.

$ filnam=`ls`

$ echo $filnam

fil1

fil2

fil3

...

$

**Пример 5.**

$ A=1 B=2

$ dat="$A + $B"

$ echo $dat

1 + 2

$

С переменными можно выполнять арифметические действия как и с обычными числами c использованием специального оператора:

expr - вычисление выражений.

Для **арифметических операций**, выполнимых командой **expr**, используются операторы:

**+** сложение; - вычитание; \\* умножение (обратная прямая скобка \ используется для отмены действия управляющих символов, здесь \*); / деление нацело; % остаток от деления.

Для **логических операций** арифметического сравнения чисел командой **expr** используются следующие обозначения: = равно; != не равно; \< меньше; \<= меньше или равно;

\> больше; \>= больше или равно.

Все операнды и операторы являются самостоятельными аргументами команды **expr** и поэтому должны отделяться друг от друга и от имени команды **expr** пробелами.

**Пример 6.**

# Текст процедуры:

a=2

a=`expr $a + 7`

b=`expr $a / 3`

c=`expr $a - 1 + $b`

d=`expr $c % 5`

e=`expr $d - $b`

echo $a $b $c $d $e

#Результат работы процедуры:

9 3 11 1 -2

Команда **expr** выводит результат вычисления на экран. Поэтому, если он не присвоен никакой переменной, то не может быть использован в программе.

При решении логических задач, связанных с **обработкой символьных строк (текстов)** команда **expr** может быть использована, например, как средство для подсчета символов в строках или для вычленения из строки цепочки символов. Операция обработки строк символов задается **кодом операции** "**:**" и шаблонами. В частности:

**'.\*' - шаблон для подсчета числа символов в строке,**

'...\(.\*\)....' - шаблон для выделения подстроки удалением символов строки, соответствующих точкам в шаблоне .

|  |  |
| --- | --- |
| **Пример 7.**  $ m=aaaaaa  $ expr $m : '.\*'  6  $ | **Пример 8.**  $ n=abcdefgh  $ expr $n : '...\(.\*\)..'  def  $ |

Выводимая информация - количество символов или подстрока - может быть присвоена некоторой переменной и использована в дальнейших вычислениях.

**Третий вид подстановки** - применяется для **подстановки команд или целых shell-процедур.** Используется для замены кода команды или текста процедуры на результат их выполнения в той же командной строке:

**$(командная\_строка)** - подстановка осуществляется также перед запуском на исполнение командной строки.

В данном случае в качестве подставляемой команды может быть использована также любая имеющая смысл sh-процедура. Shell просматривает командную строку и выполняет все команды между открывающей и закрывающей скобками. Рассмотрим примеры присвоения значений и подстановки значений локальных переменных.

**Пример 9.**

$ A='string n'

$ count=$(expr $A : '.\*')

$ echo $count

8

$

#Продолжение примера:

$ B=$(expr $A : '..\(.\*\))

$ echo $B

ring

$

Рассмотрим пример на разработку простейшей линейной процедуры обработки переменных средствами языка shell.

**ЗАДАНИЕ:**

***Создать файл, содержащий процедуру сложения двух чисел. Числа передаются в виде параметров при обращении к процедуре. Выполнить процедуру.***

$ cat>comf

SUM=$(expr $1 + $2)

echo "$1 + $2 = $SUM"

<Ctrl\*D>

$ sh comf 3 5

3 + 5 = 8

$

На экране можно **просмотреть все заведенные локальные переменные** с помощью известной команды:

**$ set**

**$**

**Удаление переменных:**

**$unset перем1 [перем2 ........]**

**$**

**4 ЭКСПОРТИРОВАНИЕ ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ В СРЕДУ shell**

При выполнении процедуры ей можно передавать как позиционные параметры так и ключевые - локальные переменные порожденного процесса. **Локальные переменные** помещаются в область локальных переменных, связанную с конкретным текущим процессом, породившим переменную. Они доступны только этому процессу и недоступны порожденным процессам-потомкам (например – sh-процедурам). Переменные другим процессам можно передавать неявно через среду. Для этого локальная переменная должна быть экспортирована (включена) в среду, в которой исполняется процедура, использующая эту переменную. Среда пользователя, т.е. **глобальные переменные**, доступна всем процессам.

Три формата команды экспортирования:

**$export список имен локальных переменных**

**$export имя\_лок\_переменной=значение**

**$export** (без параметров) **- выводит перечень всех экспортированных локальных и переменных среды** (аналог команды env)**.**

Рассмотрим некоторый фрагмент протокола работы с системой.

$color = red переменная определена, но не экспортирована

$export count = 1 переменная определена и экспортирована, т.е. потенциально доступна всем порождаемым процессам

$export

PATH = …..

HOME = …..

color = red

count = 1

$cat proc1 создание порожденного процесса процедуры

echo $color

echo $count

exit завершение процесса

<ctrl.D>

$proc1 выполнение процедуры на экран выводится значение только одной, экспортированной переменной; вторая переменная – не определена

1

$cat proc2 еще одна процедура выводятся значения обеих переменных, т.к. они определены в самой процедуре

color = black

count = 2

echo $color

echo $count

exit

$proc2

black

1

.

$echo $color

red

$echo $count

1

$

На экран выводятся первоначальные значения переменных родительскoго процесса – shell. Новые (измененные) значения локальных переменных существуют только на время существования породившего их порожденного процесса. Чтобы изменить значене переменной родительского процесса ее надо экспортировать. Но после завершения порожденного среда родительского восстанавливается.

**5 ПРОВЕРКА УСЛОВИЙ И ВЕТВЛЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ**

Все команды UNIX вырабатывают код завершения (возврата), обычно для того, чтобы в дальнейшем можно было выполнить диагностику посредством проверки значения кода завершения и определить: нормально завершилось выполнение команды (=0 или true) или не нормально (# 0 или false). Например, если (=1), то ошибка синтаксическая.

Код завершения после выполнения каждой команды помещается автоматически в некоторую специальную.

**Системную переменную и ее значение можно вывести на экран**: **echo $?**

**Пример**:

$true

$echo $?

0

$ls

$echo $?

0

$false

$echo $?

1

$cp

<сообщение о некорректности заданной команды – нет параметров>

$echo $?

1

$echo $?

0

$

Код завершения используется для программирования условных переходов в sh-процедурах. Проверка истинности условий для последующего ветвления вычислительного процесса процедур может быть выполнена с помощью команды:

**test <проверяемое отношение/условие>**

Вместо мнемоники команды может использоваться конструкция c квадратными скобками:

**[проверяемое отношение/условие]** синоним команды test.

Аргументами этой команды могут быть имена файлов, числовые или нечисловые строки (цепочки символов). Командой вырабатывается код завершения (код возврата), соответствующий закодированному в команде test условию. Код завершения проверяется следующей командой. Если закодированное параметрами условие выполняется, то вырабатывается логический результат (значение некоторой системной переменной) - true, если нет - false.

Код возврата может обрабатываться как следующей за test командой, так и специальной конструкцией языка: **if-then-else-fi**.

1. **Проверка файлов:**

**test -ключ имя\_файла**

Ключи:

**-r** файл существует и доступен для чтения;

**-w** файл существует и доступен для записи;

-x файл существует и доступен для исполнения;

-f файл существует и имеет тип "-", т.е. обычный файл;

-s файл существует, имеет тип "-" и не пуст;

-d файл существует и имеет тип "d", т.е. файл - каталог.

**2. Сравнение числовых значений:**

**test число1 –к число2**

Числа могут быть как просто числовыми строками, так и переменными, которым эти строки присвоены в качестве значений. Ключи для анализа числовых значений:

**-eq** равно; **-ne** не равно;

**-lt** меньше; **-le** меньше или равно;

**-gt** больше; **-ge** больше или равно.

Пример:

$x=5

$[$x -lt 7]

$echo $?

0

$[$x -gt 7]

$echo $?

1

$

**3. Сравнение строк:**

test [-n] 'строка' - строка не пуста (n – число проверяемых строк)

test -z 'строка' - строка пуста

test 'строка1' = 'строка2' - строки равны

test 'строка1' != 'строка2' - строки не равны

Необходимо заметить, что все аргументы команды test - строки, имена, числа, ключи и знаки операций являются самостоятельными аргументами и должны разделяться пробелами.

Пример.

$x = abc

$["$x" = "abc"]

$echo $?

0

$["$x" != "abc"]

$echo $?

1

$

ЗАМЕЧАНИЕ: выражение вида "$переменная" лучше заключать в двойные кавычки, что предотвращает в некоторых ситуациях возможную неподходящую замену переменных shell-ом.

**Особенности сравнения чисел и строк.** Shell трактует все аргументы как числа в случае, если осуществляется сравнение чисел, и все аргументы как строки, если осуществляется сравнение строк. Пример:

$X = 03

$Y =3

$[$X -eq $Y] - сравниваются значения чисел

$echo $?

0

$["$X" = "$Y"] - числа сравниваются как строки символов

$echo $?

1

$

Ветвление вычислительного процесса в shell-процедурах осуществляется семантической конструкцией:

**if список\_команд1**

**then список\_команд2**

**[else список\_команд3**

**fi**

Список\_команд - это или одна команда или несколько команд, или фрагмент shell-процедуры. Если команды записаны на одной строке, то они разделяются точкой с запятой. Для задания пустого списка команд следует использовать специальный оператор:

: (двоеточие) -пустой оператор.

Список\_команд1 передает оператору if код завершения последней выполненной в нем команды. Если он равен 0, то выполняется список\_команд2.Таким образом, код возврата 0 эквивалентен логическому значению "истина". В противном случае он эквивалентен логическому значению "ложь" и выполняется либо список\_команд3 после конструкции else, либо - завершение конструкции if словом fi.

В качестве списка\_команд1 могут использоваться списки любых команд. Однако, чаще других используется команда test. В операторе if так же допускается две формы записи этой команды:

if test аргументы

if [ аргументы ]

Каждый оператор if произвольного уровня вложенности обязательно должен завершаться словом fi.

**ЗАДАНИЕ:**

Cоздать и выполнить файл с процедурой, сравнивающей передаваемый ей параметр с некоторым набором символов (паролем).

$ cat>com

if test 'param' = "$1" - сравниваются строки символов

then echo Y

else echo N

fi

<Ctrl\*D>

$ chmod u+x com

$ com param

Y

$ com parm

N

$

**ЗАДАНИЕ.**

Организовать ветвление вычислительного процесса в процедуре в зависимости от значения переменной Х (<10, >10, = 10).

if

[$X -lt 10]

then

echo X is less 10

else

if

[$X -gt 10]

then

echo X is greatr 10

else

echo X is equal to 10

fi

fi

Для улучшения восприятия программ и облегчения отладки целесообразно придерживаться структурированного стиля написания программы. Каждому if должен соответствовать свой fi.

**6 ПОСТРОЕНИЕ ЦИКЛОВ**

Циклы обеспечивают многократное выполнение отдельных участков процедуры до достижения заданных условий.

**Цикл типа while** (пока true)**:**

**while список\_команд1**

**do список\_команд2**

**done**

Список\_команд1 возвращает код возврата последней выполненной команды. Если условие истинно, выполняется список\_- команд2, затем снова список\_команд1 с целью проверки условия, а если ложно, выполнение цикла завершается. Таким образом, циклический список\_команд2 выполняется до тех пор, пока условие истинно.

**Пример 1.**

Проверка наличия параметров при обращении к данной процедуре. Вывод на экран сообщений о наличии параметров и тексты параметров.

Текст процедуры, которой присвоено имя Р1:

if $# –eq 0

then echo “No param”

else echo “Param:’;

while test ‘$1’

do

echo “$1”

shift

done

fi

Результат работы процедуры:

$P1

No param

$P1 abc df egh

Param:

abc

df

egh

$

**Пример 2.**

Ввод строки из нескольких слов. Подсчет и вывод числа символов в каждом слове. Текст процедуры, которой присвоено имя P2:

echo “Input string:”

read A

set $A

while [ “$1” ]

do

echo “$1 = `expr ”$1” : ‘.\*’`”

shift

done

Результат работы процедуры:

$P2

Input string:

df ghghhhh aqw

df = 2

ghghhhh = 7

aqw = 3

$

**Пример 3**.

Вывести на экран слово строки (поле), номер которого (переменная N) указан в параметре при обращении к процедуре, которой присвоено имя P3. Процедура запрашивает ввод строки с клавиатуры. Номер слова вводится как аргумент процедуры.

Текст процедуры P3:

i=1 - счетчик номеров слов в строке, формируется при каждом выполнении цикла

N=$1 -значение первого параметра

echo "Введи строку: "

read a

set $a

while test $i -lt $N

do

i=`expr $i + 1` - формирование номера следующего слова

shift

done

echo "$N поле строки: \"$1\""

Пример работы процедуры P3:

$P3 2

Введи строку: aa bb cc dd

2 поле строки: "bb"

$

**Цикл типа until** (пока false)**:**

**until список\_команд1**

**do список\_команд2**

**done**

Логическое условие, определяемое по коду возврата списка\_команд1, инвертируется, т.е. цикл выполняется до техпор, пока условие ложно.

Пример процедуры с именем P4, выполняющей заданное число циклов.

$cat>P4

X = 1 - cчетчик числа циклов

until test $X -gt 10 - задано число циклов = 10

do

echo X is $X

X = `expr $X + 1`

done

<Ctrl\*D>

$sh P4

X is 1

X is 2

.................

**X is 10**

$

**Цикл типа for:**

**for имя\_ переменной [in список\_значений]**

**do список\_команд**

**done**

Переменная с указанным в команде именем заводится автоматически. Переменной присваивается значение очередного слова из списка\_значений и для этого значения выполняется список\_команд. Количество итераций равно количеству значений в списке, разделенных пробелами (т.е. циклы выполняются пока список не будет исчерпан).

**Пример** текста процедуры, печатающей в столбец список имен файлов текущего каталога.

list =`ls`

for val in $list

do

echo "$val"

done

echo end

**Пример**

Процедура, должна скопировать все обычные файлы из текущего каталога в каталог, который задается в качестве аргумента при обращении к данной процедуре по имени comfil. Процедура проверяет так же наличие каталога-адресата и сообщает количество скопированных файлов.

m=0 - переменная для счетчика скопированных файлов

if [ -d $HOME/$1 ]

then echo "Каталог $1 существует"

else

mkdir $HOME/$1 .

echo "Каталог $1 создан"

fi

for file in \*

do

if [ -f "$file" ]

then cp "$file" $HOME/$1;

m=`expr $m + 1`

fi

done

echo "Число скопированных файлов: $m"

Выполнение процедуры:

$sh comfil dir1

Число скопированных файлов:….

$

Здесь символ \* - имеет смысл <список\_имен\_файлов\_текущего\_ каталога>

**Пример**

Отобразить сведения о файлах текущего каталога

for nam in `ls`

do

echo $ nam

done

**Пример**

Процедура PROC, выводит на экран имена файлов из текущего каталога, число символов в имени которых не превышает заданного параметром числа.

if [ “$1” = “” ]

then

exit

fi

for nam in \*

do

size = `expr $nam : ‘.\*’`

if [ “$size” –le “$1” ]

then echo “Длина имени $nam $size символа”

fi

done

Вывод содержимого текущего каталога для проверки работы процедуры:

$ ls –l

total 4

drwxrwxrwx 2 lev lev 1024 Feb 7 18:18 dir1

-rw-rw-r-- 1 lev lev 755 Feb 7 18:24 out

-rwxr-xr-x 1 lev lev 115 Feb 7 17:55 f1

-rwxr-xr-x 1 lev lev 96 Feb 7 18:00 f2

$

Результаты работы процедуры:

$ PROC 2

Длина имени f1 2 символа

Длина имени f2 2 символа

$PROC 3

Длина имени out 3 символа

Длина имени f1 2 символа

Длина имени f2 2 символа

$

**Пример.**

Процедура с именем PR выводит на экран из указанного параметром подкаталога имена файлов с указанием их типа.

cd $1

for fil in \*

do

If [ -d $fil]

then echo “$fil – catalog”

else echo “$fil – file”

fi

done

Вывод содержимого подкаталога для проверки работы процедуры:

$ ls –l pdir

total 4

drwxrwxrwx 2 lev lev 1024 Feb 7 18:18 dir1

-rw-rw-r-- 1 lev lev 755 Feb 7 18:24 out

-rwxr-xr-x 1 lev lev 115 Feb 7 17:55 f1

-rwxr-xr-x 1 lev lev 96 Feb 7 18:00 f2

Результаты работы процедуры:

$ PR pdir

dir1 – catalog

out – file

f1 – file

f2 – file

$

Ветвление по многим направлениям **case**. Команда **case** обеспечивает ветвление по многим направлениям в зависимости от значений аргументов команды. Формат:

**case <string> in**

**s1) <list1>;;**

**s2) <list2>;;**

**.**

**.**

**sn) <listn>;;**

**\*) <list>**

**esac**

3десь list1, list2 ... listn - список команд. Производится сравнение шаблона string с шаблонами s1, s2 ... sk ... sn. При совпадении выполняется список команд, стоящий между текущим шаблоном sk и соответствующими знаками ;;.

**Пример:**

echo -n 'Please, write down your age'

read age

case $age in

test $age -le 20) echo 'you are so young' ;;

test $age -le 40) echo 'you are still young' ;;

test $age -le 70) echo 'you are too young' ;;

\*)echo 'Please, write down once more'

esac

В конце текста помещена звездочка \* на случай неправильного ввода числа.

Некоторые дополнительные команды, которые могут быть использованы в процедурах:

**sleep t - приостанавливает выполнение процесса на t секунд**

**Пример.**

Бесконечная (циклическая) процедура выводит каждые пять секунд сообщение в указанный файл fil.

while true

do

echo “текст\_сообщения”>fil

sleep 5

done

**Примечание**: вместо файла (или экрана) может быть использовано фиктивное устройство /dev/null (например для отладки процедуры).

**Примечание**: в процедуре реализуется бесконечный цикл. Для ограничения числа циклов надо предусмотреть счетчик циклов (см. выше) или пркратить выполнение процесса процедуры с помощью команды управления процессами - $kill (см. ниже).

**exit [n]** - **прекращение выполнения процедуры с кодом завершения [n] или с кодом завершения последней выполненной команды.**

В качестве “n” можно использовать любое число, например, для идентификации выхода из сложной процедуры, имеющей несколько причин завершения выполнения.

**Практическая часть:**

1. Выполнить все примеры из теоретической части.

2. Создать сценарий реализующий в консольном режиме диалог с пользователем в виде меню. Сценарий должен выполняться циклически пока не выбран пункт «Выход». Первый пункт меню должен выводить информацию о создателе (ФИО, группа) и краткое описание выполняемых действий, второй пункт меню должен вычислять математическое выражение 2.1, а остальные пункты реализуют действия указанные в таблице в соответствии с вариантом. Все параметры задаются в результате диалога с пользователем.

A^2+3\*B-C/2 (2.1)

|  |  |
| --- | --- |
| **Вариант** | **Задание** |
| **1** | А)Проверить существует ли папка в указанном месте и если нет создать её.  Б) архивации файлов в заданном каталоге, с созданием отдельного архива для каждого файла и удалением заархивированных. |
| **2** | А)Копирование файлов с указанного места в заданное.  Б). очистки подкаталога с подтверждением. |
| **3** | А)Удаление файлов заданного расширения в заданной папке.  Б) Разработать пакетный файл для установки даты и времени (параметры – в командной строке) |
| **4** | А)Удаление содержимого заданной папки, с помещением всех удаленных объектов в папку tmp.  Б) Проверить существование указанного в параметре файла и выдать сообщение о результате поиска |
| **5** | А) Создание файла со списком папок в указанном месте.  Б) Проверить существует ли заданный файл и доступен ли он для записи |
| **6** | А)Копирование файлов заданного расширения с указанного места в папку «BackUp»в каталоге tmp.  Б) Запрос и ввод имени пользователя, сравнение с текущим логическим именем пользователя и вывод сообщения: верно/неверно. |
| **7** | А)Перенос файлов заданного расширения с указанного места в папку «BackUp»в каталоге tmp.  Б) Проверить существует ли заданный файл и доступен ли он для исполнения, если нет установить на него атрибуты исполняемого |
| **8** | А)Создание файла со списком папок в указанном месте.  Б)определить является ли заданный каталог пустым , если нет определить количество файлов в нем. |
| **9** | А)Создание файла со списком файлов с заданным расширением в указанном месте.  Б) Разработать пакетный файл для перехода в заданный, если он существует и его архивирования |
| **10** | А)Перенос файлов с указанного места в заданное.  Б) Разработать пакетный файл для перехода студента в личный каталог. В специальный файл (logon.data) в домашней папке записывается дата и время входа в систему. |
| **11** | А) Проверить существует ли папка в указанном месте и если да переименовать её.  Б) скопировать файлы из заданного каталога в папку tmp/Имя пользователя, с добавив расширение .bak |
| **12** | А)Архивирование заданной папки с копированием архива в папку «BackUp» в каталоге tmp  Б) В заданной папке вывести все файлы с длиной больше заданного значения |
| **13** | А)Архивирование файлов заданного расширения в заданной папке с копированием архива в папку «BackUp» в каталоге tmp  Б) В заданной папке определить файл с самым длинным именем |
| **14** | А). Разработать пакетный файл для построения системы студенческих каталогов с запросом на создание каталогов требуемых курсов, групп и запросом максимального числа пользователей в группе  Б). Переход в другой каталог, формирование файла с листингом каталога и возвращение в исходный каталог. |
| **15** | А)Копирование всех фалов с заданным расширением с указанного места в указанную папку.  Б)Сохранение сведений о файлах домашнего каталога в файле «текущая дата.txt», в домашней папке History. |