**Лабораторная работа №9**

**Введение в shell-программирование.**

Эта работа посвящена ознакомлению со средствами языка shell, для создания процедур обработки данных. Изучаются вопросы оформления shell-процедур.

**Порядок выполнения работы**

* **Подготовьте свою виртуальную машину, на которой будете выполнять лабораторную работу**
* **Запустите Oracle VM VirtualBox, импортируйте машину Ubuntu Server01**
* **Пока идёт импорт (около 15 минут) ознакомьтесь с теоретической частью**
* **Запустите Ubuntu Server01, войдите в него с логином и паролем администратора, создайте пользователя с именем – вашей фамилией, включите его в группу sudo. Перезагрузите сервер и войдите под вашим логином.**
* **Выполните все действия, описанные в теоретической части (для тренировки).**
* **Выполните задания, изложенные в конце данных методических указаний, сохраните их при помощи скриншотов и включите в отчёт по лабораторной работе.**
* **Ответьте письменно на два любых контрольных вопроса, ответы также включите в отчёт.**
* **Сохраните отчёт в общей папке на сервере, для проверки, и на своём носителе.**
* **Экспортируйте образ сервера на свой носитель или в своё** **облако для последующей работы.**

***Изучение команд:***

Set – присваивание значения параметрам, передаваемым процедурам;

Echo, read, banner – вспомогательные команды для ввода и вывода информации;

If, then, else – команды проверки условий и ветвления вычисления в процедуре;

Test – проверка файлов, числовых величин, строк символов;

While, until, for – команды построения циклических процедур;

- правила построения и постановки значений переменных;

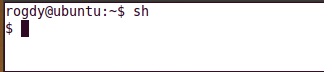
- вычисление арифметических выражений;

- обработка символьных строк.

Командный язык shell (в переводе - раковина, скорлупа) фактически есть язык программирования очень высокого уровня. На этом языке пользователь осуществляет управление компьютером. Обычно, после входа в систему вы начинаете взаимодействовать с командной оболочкой.

Программирование на языке Shell происходит в окне терминала, а сами операции выполняются сразу, как только нажимается клавиша Enter после введения текста функции. (В отличие, скажем, от языка С++, где для того, чтобы компьютер выполнил введённые вами функции, необходимо запускать написанную программу нажатием клавиши F9. В Shell такого нет, а значит, нужно быть внимательным при вводе функций и команд, т.к. в случае ошибки придётся переписывать всю программу заново).

Для того, чтобы, работая в терминале, переключиться в среду Shell, необходимо ввести команду sh. Признаком того, что оболочка (shell) готова к приему команд служит выдаваемый ею на экран промптер("$").



Имя shell-переменной - это начинающаяся с буквы последовательность букв, цифр и подчеркиваний.

Значение shell-переменной - строка символов.

Для присваивания значений переменным может использоваться оператор присваивания "=".

var\_1=13 - "13" - это не число, а строка из двух цифр. (аналог из С++: char var\_1[3] = “13”) )

var\_2="ОС UBUNTU" - здесь двойные кавычки (" ") необходимы, так как в строке есть пробел.

|  |
| --- |
| ВАЖНО: Обратим внимание на то, что, как переменная, так и ее значение должны быть записаны без пробелов относительно символа "=". |

Возможны и иные способы присваивания значений shell-переменным. Так например запись,



приводит к тому, что сначала выполняется команда "date" (обратные кавычки говорят о том, что сначала должна быть выполнена заключенная в них команда), а результат ее выполнения, вместо выдачи на стандартный выход, приписывается в качестве значения переменной, в данном случае "DAT".



Можно присвоить значение переменной и с помощью команды "read", которая обеспечивает прием значения переменной с (клавиатуры) дисплея в диалоговом режиме (аналог из С++: scanf или cin). Обычно команде "read" в командном файле предшествует команда "echo", которая позволяет предварительно выдать какое-то сообщение на экран. Например:



При выполнении этого фрагмента командного файла, после вывода на экран сообщения

Введите трехзначное число:

интерпретатор остановится и будет ждать ввода значения с клавиатуры. То число, которое пользователь введёт с клавиатуры, станет значением переменной "x".

Одна команда "read" может прочитать (присвоить) значения сразу для нескольких переменных. Если переменных в "read" больше, чем их введено (через пробелы), оставшимся присваивается пустая строка. Если передаваемых значений больше, чем переменных в команде "read", то лишние игнорируются.

|  |
| --- |
| ПРЕДУПРЕЖДЕНИЕ. На самом деле интерпретатор для продолжения работы ждет лишь нажатия клавиши. Введенное вами число воспринимается им не как число, а как последовательность символов(!). Интерпретатор не проверяет, что вы ввели. Поэтому в качестве значения переменной может оказаться любая введенная абракадабра или просто нажатие , как значение пустой строки. (Для обеспечения проверки формата ввода следует написать свою команду). |

При обращении к shell-переменной необходимо перед именем ставить символ "$" для получения Значения этой переменной. Сравните две команды, по-разному набранные с клавиатуры и заметьте разницу в результатах, распечатываемых программой на экране.  Запись echo x печатает исключительно те буквы, которые введены после команды echo. (Аналог в С: printf(“x”) и printf(“%d”, x)).

И еще один пример. Фрагмент командного файла:



выдаст на экран



Что наглядно показывает, что команда echo может за один раз выводить на экран несколько переменных.

Команда "set" устанавливает значения параметров. Это бывает очень удобно. Например, команда "date" выдает на экран текущую дату, скажем, "Mon Oct 13 16:46:28 2011", состоящую из пяти слов, тогда



выдаст на экран



Команда "set" позволяет также осуществлять контроль выполнения программы, например:

|  |  |
| --- | --- |
| **set -v** | на терминал выводятся строки, читаемые shell. |
| **Set +v** | отменяет предыдущий режим. |
| **Set -x** | на терминал выводятся команды перед выполнением. |
| **Set +x** | отменяет предыдущий режим. |

Команда "set" без параметров выводит на терминал состояние программной среды

Как во всяком языке программирования в тексте на языке shell могут быть комментарии. Для этого используется символ "#". Все, что находится в строке (в командном файле) левее этого символа, воспринимается интерпретатором как комментарий. Например,



Как во всяком процедурном языке программирования в языке shell есть операторы. Ряд операторов позволяет управлять последовательностью выполнения команд. В таких операторах часто необходима проверка условия, которая и определяет направление продолжения вычислений.

Команда test ("[ ]")

Команда test проверяет выполнение некоторого условия. С использованием этой (встроенной) команды формируются операторы выбора и цикла языка shell.

Два возможных формата команды:

test условие

или

[ условие ]

Пробелы должны быть и между значениями и символом сравнения или операции (как, кстати, и в команде "expr") . Про последнюю следует сказать подробнее. Именно команда expr даёт программе возможность воспринимать ту или иную переменную именно как число. (Иными словами, если сравнивать два ЯП, Shell и С, запись echo `expr $x` будет аналогична записи printf(“%d”, x). Разница в том, что в языке Си тип данных (char, int и пр.) определяется и устанавливается сразу при объявлении переменной, а в shell тип данных определяется после того, как значение было записано в переменную и может меняться в теле программы).

Пример работы с командами и операторами языка shell.



Попробуйте объяснить смысл каждой строки из данного примера. Какие операции запрашивает пользователь, и какие ответы даёт система?

В shell используются условия различных "типов".

Условный оператор "if"

В общем случае оператор "if" имеет структуру

if [условие] ; echo $0

then список

[elif условие

then список]

[else список]

Fi

Здесь "elif" сокращенный вариант от "else if" может быть использован наряду с полным, т.е. допускается вложение произвольного числа операторов "if" (как и других операторов). Разумеется "список" в каждом случае должен быть осмысленный и допустимый в данном контексте.

Конструкции

[elif условие

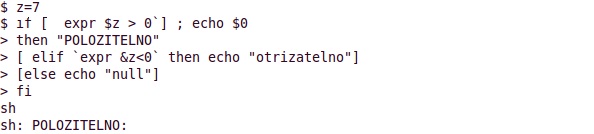
then список]

и

[else список]

не являются обязательными (в данном случае для указания на необязательность конструкций использованы квадратные скобки - не путать с квадратными скобками команды "test"!).

пример использования оператора if:



Самая усеченная структура этого оператора

if условие

then список

fi

Обратите внимание, что структура обязательно завершается служебным словом "fi". Число "fi", естественно, всегда должно соответствовать числу "if".

Примеры.

Пусть написан расчет "if-1"



Тогда вызов расчета

if-1 12 11

даст

/home/sae/STUDY/SHELL,

а

if-1 12 13

даст

if-1 : Hello!

Оператор вызова ("case")

Оператор выбора "case" имеет структуру:

case строка in

шаблон) список команд;;

шаблон) список команд;;

...

esac

Здесь "case" "in" и "esac" - служебные слова. "Строка" (это может быть и один символ) сравнивается с "шаблоном". Затем выполняется "список команд" выбранной строки. Непривычным будет служебное слово "esac", но оно необходимо для завершения структуры.

Пример.

###

# case-1: Структура "case".

# Уже рассматривавшийся в связи со

# структурой "if" пример проще и

# нагляднее можно реализовать с

# помощью структуры "case".

echo -n " А какую оценку получил на экзамене?: "

read z

case $z in

5) echo Молодец ! ;;

4) echo Все равно молодец ! ;;

3) echo Все равно ! ;;

2) echo Все ! ;;

\*) echo ! ;;

esac

Непривычно выглядят в конце строк выбора ";;", но написать здесь ";" было бы ошибкой. Для каждой альтернативы может быть выполнено несколько команд. Если эти команды будут записаны в одну строку, то символ ";" будет использоваться как разделитель команд.

Обычно последняя строка выбора имеет шаблон "\*", что в структуре "case" означает "любое значение". Эта строка выбирается, если не произошло совпадение значения переменной (здесь $z) ни с одним из ранее записанных шаблонов, ограниченных скобкой ")". Значения просматриваются в порядке записи.

###

# case-2: Справочник.

# Для различных фирм по имени выдается

# название холдинга, в который она входит

case $1 in

ONE|TWO|THREE) echo Холдинг: ZERO ;;

MMM|WWW) echo Холдинг: Not-Net ;;

Hi|Hello|Howdoing) echo Холдинг: Привет! ;;

\*) echo Нет такой фирмы ;;

esac

Оператор цикла с перечислением ("for")

Оператор цикла "for" имеет структуру:

for имя in список значений

do

список команд

done

где "for" - служебное слово определяющее тип цикла, "do" и "done" - служебные слова, выделяющие тело цикла. Не забывайте про "done"! Фрагмент "in список значений" может отсутствовать.

Пример работы с оператором done:



Т.о. мы видим, что переменная i в цикле пробегает по всем значением, заданным в условии (т.е. по очереди принимает значения всех переменных, указанных в условии), и для каждого из них выполняет ту операцию, которая заказана в теле цикла.

Оператор цикла с истинным условием ("while")

Структура "while", также обеспечивающая выполнение расчетов, предпочтительнее тогда, когда неизвестен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле.

Оператор цикла "while" имеет структуру:

while условие

do

список команд

done

где "while" - служебное слово определяющее тип цикла с истинным условием. Список команд в теле цикла (между "do" и "done") повторяется до тех пор, пока сохраняется истинность условия (т.е. код завершения последней команды в теле цикла равен "0") или цикл не будет прерван изнутри специальными командами ("break", "continue" или "exit"). При первом входе в цикл условие должно выполняться.



a=`expr $a + 1`

т.е. при каждой итерации значение "a" увеличивается на 1.

Оператор цикла с ложным условием ("until")

Оператор цикла "until" имеет структуру:

until условие

do

список команд

done

где "until" - служебное слово определяющее тип цикла с ложным условием. Список команд в теле цикла (между "do" и "done") повторяется до тех пор, пока сохраняется ложность условия или цикл не будет прерван изнутри специальными командами ("break", "continue" или "exit"). При первом входе в цикл условие не должно выполняться.

Отличие от оператора "while" состоит в том, что условие цикла проверяется на ложность (на ненулевой код завершения последней команды тела цикла) проверяется ПОСЛЕ каждого (в том числе и первого!) выполнения команд тела цикла.

Примеры. 

Здесь программа с бесконечным циклом ждет ввода слов (повторяя на экране фразу "some more"), пока не будет введено "5". После этого выдается "enough" и команда "break" прекращает выполнение цикла.

***6.1 Методика выполнения.***

1. Разработайте текст процедуры с использованием по заданию (см. ниже), вариант задания назначается преподавателем.

2. Отладьте, при необходимости отредактируйте и выполните процедуру.

3. Оформите процедуру с использованием вспомогательных команд и комментариев так, чтобы она легко читалась и чтобы результаты её работы легко анализировались.

***6.2 Контрольные вопросы.***

1. Что такое shell-процедура? Назначение?

2. Какого типа команды могут быть включены в тело процедуры?

3. Чем отличается обработка процедуры при выполнении от обработки программы на языке высокого уровня?

4. Что такое параметры? Для каких целей они используются? Какое число параметров может быть передано процедуре?

5. Какие вспомогательные команды вы использовали при оформлении процедуры?

6. Какого вида значения и как могут быть присвоены переменным языка shell?

7. Что такое локальные переменные и для каких целей их надо экспортировать в среду?

8. Как осуществляется ветвление вычислительного процесса процедуры?

9. Какого типа цикла в процедурах могут быть построены средствами языка shell?

10. Какие способы вызова процедур на исполнение вы знаете?

***6.3 Варианты заданий к лабораторной работе № 9 «Введение в shell-программирование»***

Разработать shell-процедуру с комментариями, выполняющую ниже перечисленные функции.

1. Вводит последовательность из N слов и подсчитывает в каждом введенном слове число символов. Если число символов больше М, то слово выводится на экран. Значения N и М передаются в качестве параметров.
2. Вводит строку из заданного числа слов. Выделяет слова, начинающиеся на указанную параметром букву, подсчитывает число таких слов.
3. Вводит строку N слов, анализирует длину каждого слова, упорядочивает слова по их алфавиту и выводит список на экран. Значение N задается параметром.
4. Вводит заданное параметром число слов и выводит каждое слово на печать, сопровождая его порядковым номером.
5. Вводит произвольное число коротких символьных параметров, подсчитывает длину каждого из них и выводит на экран список значений длин и общее число введенных параметров.
6. Вводит несколько коротких чисел в виде параметров, подсчитывает их сумму и результат выводит на экран.
7. Запрашивает последовательно ввод нескольких чисел со знаками и выводит на экран два списка чисел – положительных и отрицательных.
8. Запрашивает ввод строки символов, разделенных пробелами и заданной параметром длины, разбивает символы на пересекающиеся пары и выводит их на экран.
9. Ищет в личном головном каталоге пользователя созданные им файлы, выводит список их имен и распечатывается текст файла, заданного пользователем.
10. Создает новый подкаталог и помещает туда новые файлы, создаваемые пользователем по запросам процедуры. Имена новых файлов указываются параметрами.
11. Создает новый подкаталог и копирует туда из родительского каталога файлы заданного параметром типа.
12. Анализирует указанный параметром каталога и выводит на экран число файлов различного типа (обычные, директория, скрытые). Тип задается параметром.