КОНТРОЛЬНАЯ РАБОТА № 2

**СОЗДАНИЕ И ВЫПОЛНЕНИЕ КОМАНДНЫХ ФАЙЛОВ**

**В СРЕДЕ ОС LINUX**

**1. Цель работы**

Целью работы является изучение методов создания и выполнения командных файлов на языке Shell - интерпретатора.

**2. Теоретическая часть**

Взаимодействие с командным интерпретатором Shell возможно с помощью командной строки. Однако Shell является также и языком программирования, который применяется для написания командных файлов (shell - файлов). Командные файлы также называются скриптами и сценариями. Shell - файл содержит одну или несколько выполняемых команд (процедур), а имя файла в этом случае используется как имя команды.

**2.1. Переменные командного интерпретатора**

Для обозначения переменных Shell используется последовательность букв, цифр и символов подчеркивания; переменные не могут начинаться с цифры. Присваивание значений переменным проводится с использованием знака = , например, РS2 = '<' . Для обращения к значению переменной перед ее именем ставится знак $. Их можно разделить на следующие группы:

- позиционные переменные вида $n, где n - целое число;

- простые переменные, значения которых может задавать пользователь или они могут устанавливаться интерпретатором;

- специальные переменные # ? - ! $ устанавливаются интерпретатором и позволяют получить информацию о числе позиционных переменных, коде завершения последней команды, идентификационном номере текущего и фонового процессов, о текущих флагах интерпретатора Shell.

**Простые переменные.** Shell присваивает значения переменным:

z=1000

х= $z

есhо $х

1000

Здесь переменной x присвоено значение z.

**Позиционные переменные.** Переменные вида $n, где n - целое число, используются для идентификации позиций элементов в командной строке с помощью номеров, начиная с нуля. Например, в командной строке

**саt text\_1 text\_2...text\_9**

аргументы идентифицируются параметрами $1...$9. Для имени команды всегда используется S0. В данном случае $0 - это саt, $1 - text\_1, $2 - text\_2 и т.д. Для присваивания значений позиционным переменным используется команда **set**, например:

**set arg\_1 arg\_2... arg\_9**

здесь $1 присваивается значение аргумента arg\_1, $2 - arg\_2 и т.д.

Для доступа к аргументам используется команда **echo**, например:

**echo $1 $2 $9**

arg\_1 arg\_2 arg\_9

Для получения информации обо всех аргументах (включая последний) используют метасимвол \*. Пример:

**echo $\***

arg\_2 arg\_3 ... arg\_10 arg\_11 arg\_12

С помощью позиционных переменных Shell можно сохранить имя команды и ее аргументы. При выполнении команды интерпретатор Shell должен передать ей аргументы, порядок которых может регулироваться также с помощью позиционных переменных.

**Специальные переменные.** Переменные - ? # $ ! устанавливаются только Shell. Они позволяют с помощью команды **echo** получить следующую информацию:

- – текущие флаги интерпретатора (установка флагов может быть изменена командой **set**);

# – число аргументов, которое было сохранено интерпретатором при выполнении какой-либо команды;

? – код возврата последней выполняемой команды;

$ – числовой идентификатор текущего процесса PID;

! – PID последнего фонового процесса.

**2.2. Арифметические операции**

Команда **expr** (express -- выражать) вычисляет выражение expression и записывает результат в стандартный вывод. Элементы выражения разделяются пробелами; символы, имеющие специальный смысл в командном языке, нужно экранировать. Строки, содержащие специальные символы, заключают в апострофы. Используя команду **expr**, можно выполнять сложение, вычитание, умножение, деление, взятие остатка, сопоставление символов и т. д.

Пример. Сложение, вычитание:

**b=190**

**a=` expr 200 - $b`**

где ` - обратная кавычка (левая верхняя клавиша). Умножение \*, деление /, взятие остатка %:

**d=` expr $a + 125 "\*" 10`**

**c=` expr $d % 13`**

Здесь знак умножения заключается в двойные кавычки, чтобы интерпретатор не воспринимал его как метасимвол. Во второй строке переменной *с* присваивается значение остатка от деления переменной *d* на 13.

Сопоставление символов с указанием числа совпадающих символов:

**concur=` expr "abcdefgh" : "abcde"`**

**echo $concur**

ответ 5.

Операция сопоставления обозначается двоеточием (:). Результат - переменная соncur.

Подсчет числа символов в цепочках символов. Операция выполняется с использованием функции *length* в команде **expr**:

**chain="The program is written in Assembler"**

**str=` expr length "$chain"`**

**Echo $str**

ответ 35. Здесь результат подсчета обозначен переменной *str*.

**2.3. Встроенные команды**

Встроенные команды являются частью интерпретатора и не требуют для своего выполнения проведения последовательного поиска файла команды и создания новых процессов. Встроенные команды:

**cd [dir]** - назначение текущего каталога;

**exec [cmd [arg...]]** **<имя файла>** - выполнение команды, заданной аргументами cmd и arg, путем вызова соответствующего выполняемого файла.

**umask [ -o | -s] [nnn]** - устанавливает маску создания файла (маску режимов доступа создаваемого файла, равную восьмеричному числу nnn: 3 восьмеричных цифры для пользователя, группы и других). Если аргумент nnn отсутствует, то команда сообщает текущее значение маски. При наличии флага -o маска выводится в восьмеричном виде, при наличии флага -s - в символьном представлении;

**set, unset** - режим работы интерпретатора, присваивание значений параметрам;

**eval [ -arg]** - вычисление и выполнение команды;

**sh <filename.sh>** выполнение командного файла filename.sh;

**exit [n]** - приводит к прекращению выполнения программы, возвращает код возврата, равный нулю, в вызывающую программу;

**trap [cmd] [cond]** - перехват сигналов прерывания, где: cmd - выполняемая команда; cond=0 или EXIT - в этом случае команда cmd выполняется при завершении интерпретатора; cond=ERR - команда cmd выполняется при обнаружении ошибки; cond - символьное или числовое обозначение сигнала, в этом случае команда cmd выполняется при приходе этого сигнала;

**export [name [=word]...]** - включение в среду. Команда **export** объявляет, что переменные name будут включаться в среду всех вызываемых впоследствии команд;

**wait [n]** - ожидание завершения процесса. Команда без аргументов ожидает завершения процессов, запущенных синхронно. Если указан числовой аргумент n, то **wait** ожидает фоновый процесс с номером n;

**read name** - команда вводит строку со стандартного ввода и присваивает прочитанные слова переменным, заданным аргументами name.

Пример. Пусть имеется shell-файл *data*, содержащий две команды:

**echo -n "Please write down your name:"**

**read name**

Если вызвать файл на выполнение, введя его имя, то на экране появится сообщение:

Please write down your name:

Программа ожидает ввода с клавиатуры (в данном случае - фамилии пользователя). После ввода фамилии и нажатия клавиши *Enter* команда выполнится и на следующей строке появится знак - приглашение.

**2.4. Управление программами**

Команды **true** и **false** служат для установления требуемого кода завершения пpоцесса: true - успешное завершение, код завершения 0; false - неуспешное завершение, код может иметь несколько значений, с помощью которых определяется причина неуспешного завершения. Коды завершения команд используются для принятия решения о дальнейших действиях в операторах цикла **while** и **until** и в условном операторе **if**. Многие команды LINUX вырабатывают код завершения только для поддержки этих операторов.

**Условный оператор if** проверяет значение выражения. Если оно равно true, Shell выполняет следующий за **if** оператор, если false, то следующий оператор пропускается. Формат оператора **if**:

**if <условие>**

**then**

**list1**

**else**

**list2**

**fi**

Команда **test** (проверить) используется с условным оператором **if** и операторами циклов. Действия при этом зависят от кода возврата **test**. **Test** проводит анализ файлов, числовых значений, цепочек символов. Нулевой код выдается, если при проверке результат положителен, ненулевой код при отрицательном результате проверки.

В случае анализа файлов синтаксис команды следующий:

**test [ -rwfds] file**

где

-r – файл существует и его можно прочитать (код завершения 0);

-w – файл существует и в него можно записывать;

-f – файл существует и не является каталогом;

-d – файл существует и является каталогом;

-s – размер файла отличен от нуля.

При анализе числовых значений команда **test** проверяет, истинно ли данное отношение, например, равны ли А и В . Сравнение выполняется в формате:

-eq А = В

-ne А <> B

test A -ge B эквивалентно А >= В

-le А <= В

-gt А > В

-lt А < В

Отношения слева используются для числовых данных, справа – для символов.

Кроме команды **test** имеются еще некоторые средства для проверки:

! - операция отрицания инвертирует значение выражения, например, выражение **if test true** эквивалентно выражению **if test ! false**;

o - двуместная операция "ИЛИ" (or) дает значение true, если один из операндов имеет значение true;

a - двуместная операция "И" (and) дает значение true, если оба операнда имеют значение true.

**2.5. Циклы**

Оператор цикла с условием **while** true и **while** false. Команда **while** (пока) формирует циклы, которые выполняются до тех пор, пока команда **while** определяет значение следующего за ним выражения как true или false. Фоpмат оператора цикла с условием **while** true:

**while list1**

**do**

**list2**

**done**

Здесь list1 и list2 - списки команд. **While** проверяет код возврата списка команд, стоящих после **while**, и если его значение равно 0, то выполняются команды, стоящие между **do** и **done**. Оператор цикла с условием **while** false имеет формат:

**until list1**

**do**

**list2**

**done**

В отличие от предыдущего случая условием выполнения команд между **do** и **done** является ненулевое значение возврата. Программный цикл может быть размещен внутри другого цикла (вложенный цикл). Оператор **break** прерывает ближайший к нему цикл. Если в программу ввести оператор **break** с уровнем 2 (**break 2**), то это обеспечит выход за пределы двух циклов и завершение программы.

Оператор **continue** передает управление ближайшему в цикле оператору **while**.

Оператор цикла с перечислением **for**:

**for name in [wordlist]**

**do**

**list**

**done**

где name - переменная; wordlist - последовательность слов; list - список команд. Переменная name получает значение первого слова последовательности wordlist, после этого выполняется список команд, стоящий между **do** и **done**. Затем name получает значение второго слова wordlist и снова выполняется список list. Выполнение прекращается после того, как кончится список wordlist.

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' ;;**

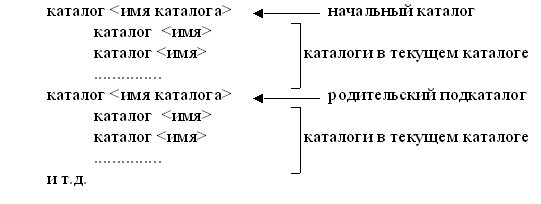
**\*)echo 'Please, write down once more'**

**esac**

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

**Задание.**

1. Shell-программа просматривает каталог, имя которого указано параметром Shell-программы и выводит имена встретившихся каталогов. Затем осуществляет переход в родительский каталог, который становится текущим и повторяются указанные действия до тех пор, пока текущим каталогом не станет корневой каталог. Форма вывода результата:



**4. Контрольные вопросы**

1. Какое назначение имеют shell - файлы?

2. Как создать shell - файл и сделать его выполняемым?

3. Какие типы переменных используются в shell - файлах?

4. В чем заключается анализ цепочки символов?

5. Какие встроенные команды используются в shell - файлах?

6. Как производится управление программами?

7. Назовите операторы создания циклов.