**Bourne Shell.**

**2.2.1.1**      **Основные возможности Shell.**

**Запуск шелла и выход из шелла**

Для запуска Bourne Shell достаточно ввести sh в текущем шеле. Выход из шела возможен либо по команде exit либо введя в терминале символ коца файла Ctrl-D.

**Последовательное выполнение команд.**

Несколько команд можно последовательно выполнять разграничивая точкой с запятой. Например, фрагмент

            $ who

$ ps –ef

$ ls –l

и

            $ who; ps –ef; ls -l

дадут полностью одинаковый результат.

**Фоновое выполнение.**

Запуск программы в фоновом режиме (без блокировки текущего шела) достигается добавлением в конец команды знака **&**

**Перенаправление ввода-вывода.**

Каждая запущенная программа имеет три ассоциированных канала: стандартный ввод, стандартный вывод и стандартный канал диагностики ошибок. По умолчанию стандартный ввод закреплен за клавиатурой а стандартный вывод и канал ошибок закреплены за монитором. Однако шелл позволяет связать эти каналы с файлами абсолютно прозрачно для выполняемойпрограмммы, т.к. изменения вывода и ввода осуществляются на системном уровне. Примеры перенаправлений ввода-вывода:

|  |  |  |
| --- | --- | --- |
| **Символ** | **Функция** | **Пример** |
| < | Перенаправление ввода из файла | program < in\_file |
| > | Перенаправление вывода в файл | program > out\_file |
| >> | Перенаправление вывода в файл с добавлением | Program >> out\_file |

**Пайпы.** Две или более программ  могут быть объединены таким образом что результат одной программы попадет на вход другой. При этом данные от одной программы к другой следуют через програмный канал, например:

program1 | program2 | program3

или более реалистичный пример:

            ls –l /etc | more

*Примечание:* пайпы всегда работают лишь в одном направлении – со стандартного выхода одной к стандартному входу другой программы.

**Расширение имен файлов.**

В целях уменьшения количества набираемых символов при вводе имен файлов шел поддерживает метасимволы. Нпример для того чтобы вывести листинг  всех файлов в текущей директории имена которых  начинаются с буквы ***a*** достаточно воспользоваться командой **ls –l  a\***

ниже приведена таблица основных метасимволов

|  |  |
| --- | --- |
| Метасимвол | Описание |
| \* | Означает любой символ в любом количестве |
| ~ | Означает путь к домашнему каталогу |
| ? | Любой символ в количестве один или больше |
| [ … ] | Равенство любого одного из символов заключенных в скобки. Парасимволов разделенная знаком минус означает любой символ из промежутка между ними. Например [a-zA-Z] – любаялатинская буква |

**2.2.1.2**                              **Shell скрипты**

**Выполнение скриптов.**

Несколько команд обьедененных одной последовательностью выполнения называется скриптом. Например последовательное выполнения команд, пайпы являются простейшими скриптами. Обычно команды из которых состоят скрипты сохраняют в файлах. Для запуска скрипта можно воспользоваться двумя методами:

sh <*имя\_файла\_скрипта*>

или установить атрибут выполняемости на файл командой

chmod +x <*имя\_файла\_скрипта*>

После чего можно выполнять скрипт:

                ./<*имя\_файла\_скрипта*>

**Вывод текста.**

Для формирования вывода текстовой информации в скриптах используют команду

echo “*строка*”

***Файл .profile****.*

Каждый раз во время входа в систему Bourne Shell автоматически запускает файл .profile (скрипт) находящийся в вашей домашней директории. Этот скрипт устанавливает “окружение” в котором Вам предстоит работать. Это различные переменныеокружения отвечающие за вид системной подсказки, путь поиска исполняемых файлов, тип терминала и.т.п.  Приведем списокосновных переменных окружения.

         PATH устанавливает путь поиска исполняемых файлов и представляет собой набор директорий разделенных двоеточием

         MAIL определяет имя файла почтового ящика для уведомления о приходе новой электронной почты

         MAILCHECK  параметр показывающий как часто (в секундах) следует проверять почтовый ящик на предмет прихода новой почты

         HOME определяет домашний каталог (каталог по умолчанию). Команда **cd** без параметров выполняет переход в этот каталог.

         PS1 основная системная подсказка (по умолчаию $)

         PS2 вторичная системная подсказка (по умолчанию >)

         SHELL имя логин шелла пользователя

         TERM тип терминала пользователя

         EDITOR имя текстового редактора по умолчанию

Устанавливать переменные окружения можно по разному. Например команды

PATH=/bin:/usr/bin:/usr/sbin:/usr/contrib/bin:/usr/local/bin

и

            PATH=/bin:/usr/bin:/usr/sbin

            PATH=$PATH: :/usr/contrib/bin:/usr/local/bin

полностью эквивалентны. Для того чтобы эти переменные вошли в системное окружение (кроме шела, стали доступны и другим программам) необходимо выполнит команду

export *<имя\_переменной>*

**Комментарии***.*

Текст следующий за символом # рассматривается шелом как комментарий и не оказывает никакого влияния на работу скрипта.

**Основы программирования на языке shell**

**Параметры.**

В дополнение к стандартным параметрам шела можно создавать свои параметры, кторые затем можно использовать в командах.Например

$ x=test

$ echo $x

test

$ aaa=/

$ ls $aaa

cdrom                    home      nsmail                    tcb          var

SD\_CDROM        dev         lib                           opt          tmp

TT\_DB  etc          lost+found            sbin        tmp\_mnt

Bin                         export     net                          stand      usr

При включении параметров в строку или в выражение знак $ должен предшествовать параметру. Также необходимо пользоваться следующими правилами:

${parameter}                         значение параметра заключенного в скобки будет использовано в выражении. Скобки {}используются когда за параметром следуют буквы или цифры не относящиеся к парамаметру. Напримересли значение параметра *xxx*равно test то значение выражения *${xxx}*123 будет равняться test123

${parameter:-word}             если параметр установлен и не пустой то результатом выражения будет являться значение параметра, в противном случае в результате будет использовано  значение параметра *word*. Например: *${xxx:-/usr/bin/sh}*

                                                если *xxx*пустой то результатом выражения будет являться строка */usr/bin/sh*

${parameter:=word}            если параметр неустановлен или пустой, тогда значение *word*будет результатом выражения.

${parameter:+word}            если параметр установлен и не пустой, тогда результатом выражения является *word*, в противном случае результат пустой.

**Аргументы командной строки.**

Когда Вы запускаете на выполнение программы, Вы можете передавать ей один или больше аргументов. Шелл скрипты имеют доступ к этим аргументам посредством параметров *$0, $1, $2 …$9*.  Если аргументов больше девяти, их значения помещаются вбуффер и могут быть  доступны с использованием команды *shift* которая будет обсуждаться ниже. Параметр $0 всегда принимает значение имени выполняемого файла, $1 – первого аргумента командной строки, $2 – второго, и.т.п.

Количество агрументов командной строки всегда можно узнать проанализировав параметр *$#*. В том случае если Вам необходим один параметр содержащий все аргументы командной строки разделенные пробелом нужно воспоьзоваться параметром *$\**.

Для получения доступа к аргументам с номером больше девяти необходимо воспользоваться командой *shift.* После очередного выполнения команды *shift*происходит сдвиг на один элемент в буффере параметров в результате которого значение параметра $2 получает параметр $1, значение параметра $3 получает $2, и.т.п. Используя цикл (цыклы будут рассвотрены ниже) в сочетании с командой *shift*можно получать доступ ко всем аргументам командной строки с номерами превышающими 9.

**Квотинг.**

Очень часто случаются ситуации когда необходима особая трактовка тех символов которые шелл воспринимает как спец символы. Например случай когда строку текста включая пробелы нужно передать как один аргумент или знак *$* должен не быть воспринят как указатель на параметр. Для этих целей используется квотинг (quoting).

Например символ бекслэша ( **\** ) можно использовать для квотинга символа *$*.

$ param=aaa

$ echo $param

aaa

$ echo \$param

$param

В сочетании с бекслэшем для квотинга можно использовать двойные кавычки. Например:

            echo "$param is a \"new directory\""

aaa is a "new directory"

При этом знак $ внутри двойных кавычек интерпретируется как указатель на параметр а бэкслэш используется для “экранирования” внутренних кавычек.

Одинарные кавычки также можно использовать для квотинга но в отличии от двойных они “экранируют” все что находится внутри них. Разница сразу становится понятной после замены в предыдущем примере двойных кавычек однираными:

$ echo '$param is a \"new directory\"'

$param is a \"new directory\"

**Подстановка команд.**

*Символ обратного ударения (****`****)* используемый в скриптах указывает на подстановку команд. Это подразумевает что результат вывода команды подставляется в шелл как параметр. Нпример:

$ echo "The current date is `date`"

The current date is Sat Jan  6 04:16:35 GMT 2001

или

$ users=`who`

$ echo "Users currentrly logged in the system:\n $users"

Users currentrly logged in the system:

 root       console      Jan  6 03:57

 roman   pts/0          Jan  5 23:12

**Условия.**

Очень часто случаются ситуации при написании скриптов когда необходимо выполнять те или иные команды в зависимости от конкретных условий.

Например, нужно отсортировать файл, а в случае его отсутствия вывести сообщение об ошибке. На этот случай шелл имеет условный оператор, и указанная задача решается с его использованием следующим образом:

if test –f $1

then

sort $1

       else

echo “file $1 doesn’t exist”

fi

Условный оператор имеет следующий синтаксис:

if *<список\_комманд1>*

then *<список\_комманд2>*

elif  *<список\_комманд3>*

then *<список\_комманд4>*

…..

else *<список\_коммандN>*

fi

Оператор ifпроверяет статус выполненной команды (в данном примере  test –f)

И в случае успеха (программа возвращает 0) выполняет команды стоящие после then, в противном случае выполняются командыelse/elif.

Наиболее часто используемой командой в оператореусловия является команда **test**. Она имеет множество опций, полный список которых можно получить обратившись к ее документации выполнив команду man test.Очень часто вместо команды testиспользуют команду **[**  которая является ее фунуциональным аналогом. Здесь приводится лишь краткий список основных опций команды test.

-r file         возвращает успех если файл существует и доступен для чтения.

                -w file        возвращает успех если файл существует и доступен для записи.

                -x file      возвращает успех если файл существует и выполняемый.

                -f file          возвращает успех если это регулярный файл.

                -d file         возвращает успех если это директория.

                -c file         возвращает успех если это специальный файл.

                -s file         возвращает успех если файл имеет ненулевой размер.

-h file        возвращает успех если это симлинк.

                -z s1           возвращает успех если длинна строки s1 нулевая.

-n s1          возвращает успех если длинна строки s1 ненулевая.

                s1 = s2    проверка двух строк на равенство.

s1 != s2   проверка двух строк на неравенство

                s1               возвращает успех если строка s1 непустая

n1 -eq n2      Алгебраическая проверка двух чисел n1 и n2 на

равенство. Помимо –eq есть еще ряд опций для алгебраического

сравнения такие как -ne, -gt, -ge, …

Все вышеперечисленные условия  могут обьеденяться с использованием логических операторов:

           !      отрицание

           -a             Бинарный И

           -o             Юинарный ИЛИ (-а имеет приоритет больший чем –о)

           ( expr )     Группировка выражений скобками

***Ввод  данных.***

Для ввода иданных в скрипт можно воспользоваться командой

                read [parameter …]

которая в качестве аргументов принимает список из одного или более параметров.

*Примечание:*знак **$** перед именем параметра в команде read ставить не нужно.

**Циклы.**

Очень часто при составлении скриптов простого последовательного выполнения бывает недостаточно. Необходимы механизмыкоторые позволяли бы выполнять команды с изменяющимися параметрами. Для этих целей шелл предоставляет в распоряжение три цикличекие конструкции: циклы **for,while**и**untill**.

**Цыкл*for***

Этот тип циклов позволяет выполнять один и тот же набор команд каждый раз с новыми значениями полученными из списка параметров. Он имеет следующий формат:

for *parameter* [ in *wordlist* ]

do *command-list*

done

где parameter это любое имя параметра, wordlist – один или несколько значений последовательно присваиваемых параметру, command-list – набор команд выполняемых при каждом проходе цикла. wordlist может быть либо просто набором аргументов разделенных пробелом, либо командой шелла которая генерирует сама аргументы. Напримерследующий скрипт:

for i in 1 2 3 4 5

do

                        if mkdir $i

                        then

                                   echo “directory $i was created”

                        fi

            done

создает последовательно директории с именами 1 2 3 4 5 и в случае успешного создания директории выдает сообщение.Следующий скрипт:

            for i in `ls a\*`

            do

                        cp $i /tmp

                        echo “$i was copied”

            done

копирует все файлы начинающиеся с буквы *a* из текущего каталога в каталог /tmp.

**Цикл *while***

while *command-list1*

do *command-list2*

done

этот цикл запускает команды из списка *command-list1,*и если последняя команда из списка выполнилась успешно (код возврата равен 0) то начинают выполняться команды из списка *command-list2,*в противном случае цикл заканчивается. Цикл изследующего примера:

                while [ -r $1 ]

                do

                               echo “processing $1”

                               cat $1 >> summary

                               shift

                done

по очереди считывает аргументы командной строки, в случае когда в текущем каталоге присутствует файл доступный на чтение с именем совпадающим с этим аргументом, содержимое этого файла дописывается к файлу с именем summary. В противном случае, скрипт прекращает свою работу.

**Цикл *util***

until command-list1

do *command-list2*

done

конструкция этого цикла полностью аналогична циклу while за исключением того что тело цикла (набор команд *command-list2)*выполняется как минимум один раз не зависимо от успешности выполнения условия цикла (набор команд *command-list1).*

**Оператор *case***

Оператор case является расширением стандартного условного оператора **if**. Если есть условие при котором может реализовываться множество вариантов то вместо серии операторов **if**лучше использовать один **case**.

case *parameter* in

*pattern1* [ | *pattern2* …] ) *command-list1* ;;

*pattern2* [ | *pattern3* …] ) *command-list2* ;;

…

esac

шаблоны *pattern* определяют варианты параметра *parameter.* При совпадении параметра с одним из шаблонов будет выполнен соответствующий список команд *command-list.* В квадратных скобках ([ **|** *pattern2*…] ) указаны дополнительные вариантыразделенные символом ( **|** ) которые может принимать *parameter* помимо основного. Заканчивается список шаблонов скобкой.Более понятней структура этого оператора станет после рассмотрения примера:

case $i in

                        -d | -c ) mkdir dir1

                                   echo “directory dir1 was created” ;;

                        -r )        rmdir dir1

                                   echo “directory dir1 was removed” ;;

                        \* )         echo “invalid option” ;;

            esac

В этом примере скрипт получив при запуске один из аргументов –d или –c попытается создать директорию с именем dir1 , получив аргумент –r попытается ее стереть. Во всех остальных случаях (шаблон \*) он выдаст предупреждение о неправильном аргументе.

*Примечание:*обратите внимание на то что порядок следования гшаблонов в операторе case имеет большое значение. Так если строку

\* )        echo “invalid option” ;;

поставить в самое начало, то на ней будет все время терминироваться скрипт.

**Команда *. (точка****)*

Каждый раз когда Вы запускаете шелл программы, создается еще одна копия шелла в которой они выполняются. Поэтому если Вы написали шелл скрипт, то все переменные “живут” только во время выполнения скрипта. Если вы хотите чтобы все переменные из шелл программы были в вашем текущем окружении запускать скрипт нужно в текущем шеле, это достигается использованием команды . (точка).

            . scriptname

**Команда*eval***

eval [arg ...]

Аргументы  читаються и соединяются в одну команду. Затем эта команда выполняется шеллом и статус выхода команды возвращается как результат команды eval. Если аргументы в команде отсутствуют или пустые то команда возвращает нулевой статус.

В качестве примера рассмотрим два скрипта:

            d=’date &’ ; $d

и

                d=’eval date &’ ; $d

первый из них не выполнится так как команда date воспримет символ & как аргумент а не как признак запуска в фоновом режиме, в результате чего первый скрипт не выполнится в отличии от второго.

**Использование метасимволов.**

Во всех конструкциях циклов и операторе case возможно использование метасимволов. Например скрипт из примера оператора цикла for

for i in `ls a\*`

            do

                        cp $i /tmp

                        echo “$i was copied”

            done

можно переписать в более простом виде

for i in a\*

            do

                        cp $i /tmp

                        echo “$i was copied”

            done

результат будет один и тот же.

**2.3.1 Некоторые специальные команды**

***Разделители && и ||***

Это условные разделители. При разделении двух команд с помощью && вторая команда выполнится лишь в том случае когда первая завершится успешно. При разделении команд с помощью || вторая выполнится лишь тогда когда первая закончиласьнеуспешно (код возврата не равен нулю).

Например при выполнении скрипта:

test –d /usr/tools && cd /usr/tools

test –d /usr/tools || echo “directory doesn’t exist”

вход в директорию /usr/tools будет произведен только при ее наличии, в противном случае будет выдано сообщение о ееотсутствии.

**Определение функций**

Для сокращения обьема шелл программ и упрощения их понимания и сопровждения шелл допускает введение и использование функций. Для опредения функции используется следующий синтаксис:

name () { *list*; }

где name – это имя функции, а list – список команд из которых состоит тело функции. Ниже приведен пример функции возвращающей 0 если аргумент переданный ей является директорией и 1 в противном случае.

dir\_test () {

                        if [ -d $1 ]

                        then

                                   echo “$1 is a directory”

                                   return 0

                        else

                                   echo “$1 is not directory”

                                   return 1

                        fi;

}

вызов функции осуществляется следующим образом

                name [ *parameter* … ]

например в нашем случае это можно сделать так

            dir\_test  /usr/bin/sh

**Перенаправление ввода-вывода**

Как было сказано выше для перенаправления ввода-вывода используються символы (> перенаправление вывода,  >>перенаправление вывода с добавлением в файл и < перенаправление ввода). Кроме этого существует еще ряд конструкций, одна из которых:

                << [-] *word*

при этом все линии от первой и до содержащей строчку word будут использоваться как входные данные. Например:

                $ cat<<mark

> These words will be printed

> the cat command until the "mark"

> word is found

> mark

These words will be printed

the cat command until the "mark"

word is found

$

Если перед строчкой word стоит минус то все символы табуляции в начале строк будут вырезаны.

Другой часто используемой конструкцией при перенаправлении ввода-вывода является:

                <& цифра

            >& цифра

При этом задействуется файловый дискриптор ассоциированый с указаной цифрой. В большинстве программ со стандартным вводом связан дискриптор 0, со стандартным выводом дискриптор 1, и со стандартным потоком ошибок дискриптор 2. Всепрограммы которые работают друг с другом через пайпы по умолчанию пользуються дискрипторами 0 и 1. Наиболее частоиспользуемые перенаправления это 1>&2 и 2>&1. Рассмотрим следующий пример из которого станет понятен смысл этих конструкций:

            $ ls /no/such/file > out

/no/such/file not found

$ cat out

$

$ ls /no/such/file >out 2>&1

$ cat out

/no/such/file not found

$

первая команда пытается вывести листинг не суцествующего файла перенаправив стандартный вывод в файл out. Файл out при этом оказывется пустым т.к. сообщение об ошибке выводится в стандартный поток ошибок и появляется на терминале. Вторая команда объединяет стандартный поток ошибок со стандартным выводом который перенаправлен в файл out. При этом сообщения об ошибке попадают в файл, о чем свидетельствует команда cat.

**Команда *exec***

exec [*arg* …]

Эта команда выполняет замещение текущего шела новым шелом или программой. Разница между простым запуском шела и запуском через exec становится очевидной на следующем примере:

$ ksh

$ ps

PID TTY       TIME COMMAND

                2125 pts/0     0:00 ksh

                2094 pts/0     0:00 sh

                2126 pts/0     0:00 ps

                $

$ exec ksh

$ ps

                PID TTY       TIME COMMAND

                2127 pts/0     0:00 ps

                2094 pts/0     0:00 ksh

$

**Команда *expr***

                expr *expression* { +, -, \\*, /, \*, =, \>, \>=, \<, \<=, != } *expression*

Это очень полезная команда для выполнения арифметических операций в шелл скриптах. Например:

                x=10

            expr $x + 5

                y=`expr $x – 10`

            if expr $x \<= $y

            then

                        echo “$x is less or equal than $y”

            fi

**Команда *set***

Эта команда используется во многих модификациях. Основное назначение – это устанавливать значение параметров. Если Вы просто запустите эту команду без аргументов то увидите все параметры вашего окружения, большинство из которых было установлено при входе в систему из файла .profile.

Наример команда:

            set  bob brr kab ram

установит параметры $1, $2, $3, $4 следующим образом $1=”bob”, $2=”brr”, $3=”kab”, $4=”ram”. Команда set имеет множество опций, полный список которых можно получить обратившись к man-странице (man set).

**Команда *trap***

Команда trap ожидает приход сигналов посланных шелу (от внешних процессов или сигнал от шела вследствии неуспешного запуска программы) и выполняет их обработку.

            trap [*command\_list*] [*s1* …]

Когда trap получает сигнал s1 (сигналы  s2 …) она выполняет список заранее предопределенных в *command\_list*  команд. Еслиs1 равен 0, то команды запускаются когда шелл заканчивает свою работу. Команда trap запущенная без аргументов выдает на печать список команд ассоциированных с каждым из номеров сигналов. Ниже приведен краткий список наиболее частоиспользуеиых сигналов.

|  |  |  |
| --- | --- | --- |
| **Номер сигнала** | **Описание** | **Возможность перехвата** |
| 1 | сигнал перезапуска | Есть |
| 2 | Прерывание процесса | Есть (если процесс не в фоне) |
| 3 | Сигнал выхода | Есть (если процесс не в фоне) |
| 6 | Сигнал завершения | Есть |
| 9 | удаление процесса | **Не** может быть перехвачен |
| 14 | Сигнал таймера | Нет |
| 15 | Програмный сигналзавершения процесса | Есть |

Попытка перехватить 11-й сигнал (memory fault) приводит к ошибке. Сигнал с номером 9 не может быть перехвачен т.к. он не перехватывается в принципе.

Очень часто возникает ситуация когда скрипт в процессе работы создает временные файлы которые затем нужно удалить, но в случае прерывания процесса какимто из сигналов эти файлы не будут удалены. Эта проблема решается в следующем примере:

trap “echo ‘removing temporaty file’ rm /tmp/temp” 0 1 2 3 15

**Команда *pwd***

Эта команда показывает текущую рабочую директорию.

**Команда *type***

type [ *prog\_name* …]

Эта команда ищет каталог в котором находится запускаемая программа указанная в аргументе команды.

Команда *times*

Эта команда выдает информацию о времени выполнения процесса запущеного из шелла.

**Параметры устанавливаемые шеллом**

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| $# | Количество аргументов командной строки |
| $? | Значение кода возврата предыдущей команды |
| $$ | Номер процесса шела в системе |
| $! | Номер процесса в системе последней команды запущеной в фоне |

**Основные опции для запуска *Bourn Shell***

|  |  |
| --- | --- |
| **Опция** | **Описание** |
| -с string | Выполнять команды из файла string |
| -s | Выполнять команды полученные со сотандартного ввода (в случе запускашела без опций он ведет себя также) |
| -r | Запустить шелл в ограниченном режиме (restricted mode) |

При запуске шела в restricted mode запрещается:

         Изменение директории командой cd

         Устанавливать переменную окружения PATH

         Запускать программы содержащие в имени символ /

         Перенаправлять ввод-вывод