Программирование на языке Shell в системе UNIX. Примеры программирования

1. ВВЕДЕНИЕ

Среди операционных систем особое место занимает Unix. Беспрецедентным является

то, что ОС Unix может работать практически на всех выпускаемых платформах. UNIX

- это стандарт де факто открытых и мобильных операционных систем. (поскольку

название UNIX запатентовано компанией AT&T - различные юниксы называются

различно: SCO UNIX, BSDI, Solaris, Linux, DG/UX, AIX и т.д.).

Это не только многозадачная, но и многопользовательская система. Она

обеспечивает современный пользовательский интерфейс на базе системы X Window и

межмашинную связь на базе протоколов TCP/IP и т.п.

ОС Unix была создана Кеном Томпсоном и Деннисом Ритчи в Bell Laborotories

(AT&T). Широко распространяться Unix/v7 (версия 7) начала в 79 - 80-м годах.

Вручение создателям Unix в 1983 году Международной премии А.Тьюринга в области

программирования ознаменовало признание этой системы мировой научной (computer

science) общественностью. Что также беспрецедентно.

ОС Unix стоит на трех китах: язык Си, файловая система, командный язык. В

дальнейшем к ним добавились система X Window и протоколы TCP/IP.

2. ПРОСТЕЙШИЕ СРЕДСТВА SHELL

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. Признаком того, что оболочка (shell)

готова к приему команд служит выдаваемый ею на экран промптер. В простейшем

случае это один доллар ("$").

Shell не является необходимым и единственным командным языком (хотя именно он

стандартизован в рамках POSIX [POSIX 1003.2] - стандарта мобильных систем).

Например, немалой популярностью пользуется язык cshell, есть также kshell,

bashell и другие. Более того, каждый пользователь может создать свой командный

язык. Может одновременно на одном экземпляре операционной системы работать с

разными командными языками.

shell - это одна из многих команд UNIX. То есть в набор команд оболочки "shell"

входит команда "sh" - вызов интерпретатора "shell". Первый "shell" вызывается

автоматически при вашем входе в систему и выдает на экран промтер. После этого

вы можете вызывать на выполнение любые команды, в том числе и снова сам "shell",

который вам создаст новую оболочку внутри прежней.

Так например, если вы подготовите в редакторе файл "file\_1":

echo Hello!

то это будет обычный текстовый файл, содержащий команду "echo", которая при

выполнении выдает все написанное правее ее на экран. Можно сделать файл "file\_1"

выполняемым с помощью команды "chmod 755 file\_1". Но его можно выполнить, вызвав

явно команду "sh" ("shell"):

sh file\_1

или

sh < file1

Файл можно выполнить и в текущем экземпляре "shell". Для этого существует

специфическая команда "." (точка), т.е.

. file\_1

Поскольку UNIX - система многопользовательская, вы можете даже на персональном

компьютере работать параллельно, скажем, на 12-ти экранах (переход с экрана на

экран ALT/функциональная клавиша), имея на каждом экране нового (или одного и

того же) пользователя со своей командной оболочкой. Можете и в графическом

режиме X-Window также открыть большое число окон, а в каждом окне может быть

свой пользователь со своей командной оболочкой...

Стержневым элементом языка shell является команда.

2.1. Структура команд

Команды в shell обычно имеют следующий формат:

<имя команды> <флаги> <аргумент(ы)>

Например:

ls -ls /usr/bin

Где ls - имя команды выдачи содержимого директория,

-ls - флаги ( "-" - признак флагов, l - длинный формат, s - об'ем файлов в

блоках),

/usr/bin - директорий, для которого выполняется команда.

Эта команда выдаст на экран в длинном формате содержимое директория /usr/bin,

при этом добавит информацию о размере каждого файла в блоках.

К сожалению, такая структура команды выдерживается далеко не всегда. Не всегда

перед флагами ставится минус, не всегда флаги идут одним словом. Есть

разнообразие и в представлении аргументов. К числу команд, имеющих экзотические

форматы, относятся и такие "ходовые" команды, как сс – вызов компилятора языка

С, tar – работа с архивами, dd – копирование файла с преобразованием, find –

поиск файлов и ряд других.

Как правило, первое слово shell воспринимает, как команду. Поэтому в командной

строке

cat cat

первое слово будет расшифровано shell, как команда (конкатенации), которая

выдаст на экран файл с именем "cat" (второе слово), находящийся в текущем

директории.

2.2. Группировка команд.

Средства группировки:

; и <перевод строки> - определяют последовательное выполнение команд;

& - асинхронное (фоновое) выполнение предшествующей команды;

&& - выполнение последующей команды при условии нормального завершения

предыдущей, иначе игнорировать;

- выполнение последующей команды при ненормальном завершении предыдущей, иначе

игнорировать.

При выполнении команды в асинхронном режиме (после команды стоит один амперсанд)

на экран выводится номер процесса, соответствующий выполняемой команде, и

система, запустив этот фоновый процесс, вновь выходит на диалог с пользователем.

Например, наберем команду "find" в фоновом режиме для поиска в системе , начиная

от корня "/", файла с именем "conf", а затем "pwd" в обычном режиме. На экране

этот фрагмент будет выглядеть следующим образом (курсивом выделены комментарии):

$ find/-name conf -print & ввод команды "find"

288 номер (PID) фонового процесса

$ pwd ввод команды "pwd"

/mnt/lab/asu результат работы "pwd"

$ возвращение shell в промптер

/usr/include/sys/conf результат работы "find"

Иногда необходимо, чтобы все фоновые процессы завершились, прежде чем будет

выполняться какой-то расчет. Для этого служит специальная команда "wait [PID]".

Эта команда ждет завершения указанного идентификатором (числом) фонового

процесса. Если команда без параметра, то она ждет завершения всех фоновых

процессов, дочерних для данного "sh".

Для группировки команд также могут использоваться фигурные "{}" и круглые "()"

скобки. Рассмотрим примеры, сочетающие различные способы группировки: Если

введена командная строка

command1 && command2; command3

где command1, command2 и command3 - какие-то команды, то "command2" будет

выполнена только при успешном завершении "command1"; после любого из исходов

обработки "command2" (т.е. "command2" будет выполнена, либо пропущена) будет

выполнена "command3".

command1 && { command2; command3}

Здесь обе команды ("command2" и "command3") будут выполнены только при успешном

завершении "command1".

{command1; command2} &

В фоновом режиме будет выполняться последовательность команд "command1" и

"command2".

Фоновые процессы сложно уничтожить, поскольку традиционная команда "CTL/C"

прерывает только процессы переднего плана. Для уничтожения фонового процесса

надо знать его номер. При запуске фонового процесса на экран выдается число,

соответствующее номеру (идентификатору) этого процесса (PID). Если этот номер

забыт или надо убедиться, что этот процесс не закончен, с помощью команды

ps -aux

можно получить перечень идентификаторов процессов (PID), имена пользователей,

текущее время, затраченное процессами, и т.д.

В выведенной таблице можно найти номера процессов, подлежащих уничтожению,

например это "849" и "866". Тогда командой

kill -9 866 849

можно уничтожить эти процессы. При уничтожении процессов надо вы должны иметь то

же имя пользователя, какое было приписано уничтожаемым процессам или root.

Круглые скобки "()", кроме выполнения функции группировки, выполняют и функцию

вызова нового экземпляра интерпретатора shell.

Пусть мы находились в начальном каталоге "/mnt/lab/asu"

Тогда в последовательности команд

cd ..; ls; ls

две команды "ls" выдадут 2 экземпляра содержимого каталога "/mnt/lab", а

последовательность

(cd ..; ls) ls

выдаст сначала содержимое каталога "/mnt/lab", а затем содержимое

"/mnt/lab/asu", т.к. при входе в скобки вызывается новый экземпляр shell, в

рамках которого и осуществляется переход. При выходе из круглых скобок

происходит возврат в старый shell и в старый каталог.

2.3. Перенаправление команд

Стандартный ввод (вход) - "stdin" в ОС UNIX осуществляется с клавиатуры

терминала, а стандартный вывод (выход) - "stdout" направлен на экран терминала.

Существует еще и стандартный файл диагностических сообщений - "stderr", о

котором речь будет чуть позже.

Команда, которая может работать со стандартным входом и выходом, называется

ФИЛЬТРОМ.

Пользователь имеет удобные средства перенаправления ввода и вывода на другие

файлы (устройства). Символы ">" и ">>" обозначают перенаправление вывода.

ls >file\_1

команда "ls" сформирует список файлов текущего каталога и поместит его в файл

"file\_1" (вместо выдачи на экран). Если файл "file\_1" до этого существовал, то

он будет затерт новым.

pwd >>file\_1

команда pwd сформирует полное имя текущего каталога и поместит его в конец файла

"file\_1", т.е. ">>" добавляет в файл, если он непустой.

Символы "<" и "<<" обозначают перенаправление ввода.

wc -l <file\_1

подсчитает и выдаст на экран число строк в файле file\_1.

ed file\_2 <<!

создаст с использованием редактора файл "file\_2", непосредственно с терминала.

Окончание ввода определяется по символу, стоящему правее "<<" (т. е. "!"). То

есть ввод будет закончен, когда первым в очередной строке будет "!".

Можно сочетать перенаправления. Так

wc -l <file\_3 >file\_4

и

wc -l >file\_4 <file\_3

выполняются одинаково: подсчитывается число строк файла "file\_3" и результат

помещается в файл "file\_4".

Средство, объединяющее стандартный выход одной команды со стандартным входом

другой, называется КОНВЕЙЕРОМ и обозначается вертикальной чертой "".

ls wc -l

список файлов текущего каталога будет направлен на вход команды "wc", которая на

экран выведет число строк каталога.

Конвейером можно объединять и более двух команд, когда все они, возможно кроме

первой и последней - фильтры:

cat file\_1 grep -h result sort cat -b > file\_2

Данный конвейер из файла "file\_1" ("cat") выберет все строки, содержащие слово

"result" ("grep"), отсортирует ("sort") полученные строки, а затем пронумерует

("cat -b") и выведет результат в файл "file\_2".

Поскольку устройства в ОС UNIX представлены специальными файлами, их можно

использовать при перенаправлениях. Специальные файлы находятся в каталоге

"/dev". Например, "lp" - печать; "console" - консоль; "ttyi" - i-ый терминал;

"null"- фиктивный (пустой) файл (устройство).

Тогда, например,

ls > /dev/lp

выведет содержимое текущего каталога на печать, а

file\_1 < /dev/null

обнулит файл "file\_1".

sort file\_1 tee /dev/lp tail -20

В этом случае будет отсортирован файл "file\_1" и передан на печать, а 20

последних строк также будут выданы на экран.

Вернемся к перенаправлению выхода. Стандартные файлы имеют номера:

0 - stdin,

1 - stdout

2 - stderr.

Если вам не желательно иметь на экране сообщение об ошибке, вы можете

перенаправить его с экрана в указанный вами файл (или вообще "выбросить",

перенаправив в файл "пустого устройства" - /dev/null). Например при выполнении

команды

cat file\_1 file\_2

которая должна выдать на экран последовательно содержимое файлов "file\_1" и

"file\_2", выдаст вам, например, следующее

111111 222222

cat: f2: No such file or directory

где 111111 222222 - содержимое файла "file\_1", а файл "file\_2" отсутствует, о

чем команда "cat" выдала сообщение в стандартный файл диагностики, по умолчанию,

как и стандартный выход, представленный экраном.

Если вам не желательно такое сообщение на экране, его можно перенаправить в

указанный вами файл:

cat file\_1 file\_2 2>f-err

сообщения об ошибках будут направляться (об этом говорит перенаправление "2>") в

файл "f-err". Кстати, вы можете всю информацию направлять в один файл "ff",

использовав в данном случае конструкцию

cat file\_1 file\_2 >>ff 2>ff

Можно указать не только какой из стандартных файлов перенаправлять, но и в какой

стандартный файл осуществить перенаправление.

cat file\_1 file\_2 2>>ff 1>&2

Здесь сначала "stderr" перенаправляется (в режиме добавления) в файл "ff", а

затем стандартный выход перенаправляется на "stderr", которым к этому моменту

является файл "ff". То есть результат будет аналогичен предыдущему.

Конструкция "1>&2" - означает, что кроме номера стандартного файла, в который

перенаправить, необходимо впереди ставить "&"; вся конструкция пишется без

пробелов.

 <- закрывает стандартный ввод.

 >- закрывает стандартный вывод.

2.4. Командные файлы.

Для того, чтобы текстовый файл можно было использовать как команду, существует

несколько возможностей.

Пусть с помощью редактора создан файл с именем "cmd", содержащий одну строку

следующего вида:

date; pwd; ls

Можно вызвать shell как команду, обозначаемую "sh", и передать ей файл "cmd",

как аргумент или как перенаправленный вход, т.е.

$ sh cmd

или

$ sh <cmd

В результате выполнения любой из этих команд будет выдана дата, затем имя

текущего каталога, а потом содержимое каталога.

Более интересный и удобный вариант работы с командным файлом - это превратить

его в выполняемый, т.е. просто сделать его командой, что достигается изменением

кода защиты. Для этого надо разрешить выполнение этого файла.

Например,

chmod 711 cmd

сделает код защиты "rwx\_\_x\_\_x". Тогда простой вызов

cmd

приведет к выполнению тех же трех команд.

Результат будет тот же, если файл с содержимым

date; pwd; ls

представлен в виде:

date

pwd

ls

так как переход на другую строку также является разделителем в

последовательности команд.

Таким образом, выполняемыми файлами могут быть не только файлы, полученные в

результате компиляции и сборки, но и файлы, написанные на языке shell. Их

выполнение происходит в режиме интерпретации с помощью shell-интерпретатора

2.5 Отладка командных файлов

В SHELL используются два механизма отладки командных файлов.

Первый из них: set -v выводит строки командного файла по мере их чтения. Этот

режим применяется при поиске синтаксических ошибок. Для его использования не

требуется производить модификацию командного файла, например:

sh -v proc... здесь proc - имя командного файла. Ключ -v может использоваться

вместе с ключом -n, предотвращающим выполнение следующих за ним команд (команда

set -n блокирует терминал до тех пор, пока не вводится признак конца файла EOF).

Команда set -х выводит команды по мере их выполнения, причём на терминал

выводятся строки программы и на место переменных подставляются их значения. Для

отмены ключей -x и -v можно воспользоваться командой set - а для установки -

присвоить соответствующее значение макропеременной.

3. СРЕДА SHELL (ПЕРЕМЕННЫЕ И ПАРАМЕТРЫ)

На языке shell можно писать командные файлы и с помощью команды "chmod" делать

их выполняемыми. После этого они ни чем не отличаются от прочих команд ОС UNIX.

3.1. shell-переменные

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

подчеркиваний.

Значение shell-переменной - строка символов.

То, что в shell всего два типа данных: строка символов и текстовый файл, с одной

стороны, позволяет легко вовлекать в программирование конечных пользователей,

никогда ранее программированием не занимавшихся, а с другой стороны, вызывает

некий внутренний протест у многих программистов, привыкших к существенно

большему разнообразию и большей гибкости языковых средств.

Однако интересно наблюдать то, как высококлассные программисты, освоившись с

"правилами игры" shell, пишут на нем программы вомного раз быстрее, чем на Си,

но, что особенно интересно, в ряде случаев эти программы работают даже быстрее,

чем реализованные на Си.

Имя переменной аналогично традиционному представлению об идентификаторе, т.е.

именем может быть последовательность букв, цифр и подчеркиваний, начинающаяся с

буквы или подчеркивания.

Для присваивания значений переменным может использоваться оператор присваивания

"=".

var\_1=13 - "13" - это не число, а строка из двух цифр.

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

пробел.

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

запись,

DAT=`date`

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. Обычно команде "read" в командном файле предшествует команда "echo",

которая позволяет предварительно выдать какое-то сообщение на экран. Например:

echo -n "Введите трехзначное число:"

read x

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

Введите трехзначное число:

интерпретатор остановится и будет ждать ввода значения с клавиатуры. Если вы

ввели, скажем, "753" то это и 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.

При обращении к shell-переменной необходимо перед именем ставить символ "$". Так

команды

echo $var\_2

echo var\_2

выдадут на экран

ОС UNIX

var\_2

3.2. Экранирование

Рассмотрим более подробно приемы экранирования, используемые в shell. В качестве

средств экранирования используются двойные кавычки (" "), одинарные кавычки ('

') и бэк-слэш (\).

Из примеров очевидно их действие:

Можно в одной строке записывать несколько приcваиваний.

x=22 y=33 z=$x

A="$x" B='$x' C=\$x

D="$x + $y + $z" E='$x + $y + $z' F=$x\ +\ $y\ +\ $z

(присваивание G=$x+$y не было бы выполнено из-за пробелов)

Тогда

echo A = $A B = $B C = $C

echo D = $D E = $E F = $F

eval echo evaluated A = $A

eval echo evaluated B = $B

eval echo evaluated C = $C

Выдадут на экран

A = 22 B = $x C = $x

D = 22 + 33 + 22 E = $x + $y + $z F = 22 + 33 + 22

evaluated A = 22

evaluated B = 22

evaluated C = 22

Приведем еще примеры, связанные с экранированием перевода строки. Пусть

переменной "string" присвоено значение "массива" 2x3:

abc

def

Обратим внимание, что для избежания присваивания лишних пробелов вторая строка

массива начата с первой позиции следующей строки:

string="abc

def"

Тогда три варианта записи переменной в команде "echo"

echo $string

echo '$string'

echo "$string"

дадут соответственно три различных результата:

abc def

$string

abc

def

а последовательность команд

echo "str\_1

str\_2" > file\_1

echo 'str\_1

str\_2' > file\_2

cat file\_1 file\_2

даст выдаст последовательно одинаковые файлы file\_1 и file\_2:

str\_1

str\_2

str\_1

str\_2

Заметим также, что бэк-слэш (\) не только экранирует следующий за ним символ,

что позволяет использовать специальные символы просто как символы,

представляющие сами себя (он может экранировать и сам себя - \\), но в командном

файле бэк-слэш позволяет об'единять строки в одну (экранировать конец строки).

Например, приводившийся ранее пример командной строки:

cat file\_1 grep -h result sort cat -b > file\_2

может быть записан в командном файле, скажем, как

cat file\_1 grep -h \

result sort cat -b > file\_2

Кстати, эффект продолжения командной строки обеспечивает и символ конвейера. В

данном случае это может дать более симпатичный результат, например:

cat file\_1

grep -h result

sort

cat -b > file\_2

3.3. Манипуляции с shell-переменными

Несмотря на то, что shell-переменные в общем случае воспринимаются как строки,

т. е. "35" - это не число, а строка из двух символов "3" и "5", в раде случаев

они могут интерпретироваться иначе, например, как целые числа.

Разнообразные возможности имеет команда "expr".

Проиллюстрируем некоторые на примерах:

Выполнение командного файла:

x=7 y=2

a=`expr $x + $y` ; echo a=$a

a=`expr $a + 1` ; echo a=$a

b=`expr $y - $x` ; echo b=$b

c=`expr $x '\*' $y` ; echo c=$c

d=`expr $x / $y` ; echo d=$d

e=`expr $x % $y` ; echo e=$e

выдаст на экран

a=9

a=10

b=-5

c=14

d=3

e=1

Операция умножения ("\*") обязательно должна быть заэкранирована, поскольку в

shell этот значок воспринимается, как спецсимвол, означающий, что на это место

может быть подставлена любая последовательность символов.

С командой "expr" возможны не только (целочисленные) арифметические операции, но

и строковые:

A=`expr 'cocktail' : 'cock'` ; echo $A

B=`expr 'cocktail' : 'tail'` ; echo $B

C=`expr 'cocktail' : 'cook'` ; echo $C

D=`expr 'cock' : 'cocktail'` ; echo $D

На экран будут выведены числа, показывающее число совпадающих символов в

цепочках (от начала). Вторая из строк не может быть длиннее первой :

4

0

0

0

3.4. Экспорт переменных

В ОС UNIX существует понятие процесса. Процесс возникает тогда, когда

запускается на выполнение какая-либо команда.

Например, при наборе на клавиатуре "р <Enter>" порождается процесс "р". В свою

очередь "р" может породить другие процессы. Допустим, что "р" вызывает "р1" и

"р2", которые последовательно порождают соответствующие процессы.

У каждого процесса есть своя среда - множество доступных ему переменных.

Например, до запуска "р" уже существовала среда, в которой уже были определены

некоторые переменные. Запуск "р" порождает новую среду; уже в ней будут

порождены "р1" и "р2".

Переменные локальны в рамках процесса, в котором они объявлены, т.е. где им

присвоены значения. Для того, чтобы они были доступны и другим порождаемым

процессам, надо передать их явным образом. Для этого используется встроенная

команда "export".

3.5. Параметры

В командный файл могут быть переданы параметры. В shell используются позиционные

параметры (т.е. существенна очередность их следования). В командном файле

соответствующие параметрам переменные (аналогично shell-переменным) начинаются с

символа "$", а далее следует одна из цифр от 0 до 9:

Пусть "examp-1" вызывается с параметрами "cock" и "tail". Эти параметры попадают

в новую среду под стандартными именами "1" и "2". В (стандартной) переменной с

именем "0" будет храниться имя вызванного расчета.

При обращении к параметрам перед цифрой ставится символ доллара "$" (как и при

обращении к переменным):

 $0 соответствует имени данного командного файла;

 $1 первый по порядку параметр;

 $2 второй параметр и т.д.

Поскольку число переменных, в которые могут передаваться параметры, ограничено

одной цифрой, т.е. 9-ю ("0", как уже отмечалось имеет особый смысл), то для

передачи большего числа параметров используется специальная команда "shift".

Своеобразный подход к параметрам дает команда "set".

Например, фрагмент

set a b с

echo первый=$1 второй=$2 третий=$3

выдаст на экран

первый=a второй=b третий=c

т.е. команда "set" устанавливает значения параметров. Это бывает очень удобно.

Например, команда "date" выдает на экран текущую дату, скажем, "Mon May 01

12:15:10 2000", состоящую из пяти слов, тогда

set `date`

echo $1 $3 $5

выдаст на экран

Mon 01 2000

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="Times New Roman">" без параметров выводит на терминал состояние

программной среды.

3.6. Подстановки shell-интерпретатора

Перед началом непосредственной интерпретации и выполнением команд, содержащихся

в командных файлах, shell выполняет различные виды подстановок:

ПОДСТАНОВКА РЕЗУЛЬТАТОВ. Выполняются все команды, заключенные в обратные

кавычки, и на их место подставляется результат.

ПОДСТАНОВКА ЗНАЧЕНИЙ ПАРАМЕТРОВ И ПЕРЕМЕННЫХ. То есть слова, начинающиеся на

"$", заменяются соответствующими значениями переменных и параметров.

ИНТЕРПРЕТАЦИЯ ПРОБЕЛОВ. Заэкранированные пробелы игнорируются.

ГЕНЕРАЦИЯ ИМЕН ФАЙЛОВ. Проверяются слова на наличие в них спецсимволов ("\*",

"?","[]") и выполняются соответствующие генерации.

3.7. Программная среда

Каждый процесс имеет среду, в которой он выполняется. Shell использует ряд

переменных этой среды.

Если вы наберете команду "set" без параметров, то на экран будет выдана

информация о ряде стандартных переменных, созданных при входе в систему (и

передаваемых далее всем вашим новым процессам "по наследству"), а также

переменных, созданных и экспортируемых вашими процессами.

Конкретный вид и содержание выдаваемой информации в немалой степени зависит от

того, какая версия UNIX используется и как инсталлирована система.

Результат выполнения команды set без параметров (не полный):

HOME=/root

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

Прокомментируем значения переменных.

HOME=/root - это имя домашнего директория, в котором пользователь оказывается

после входа в систему. То есть, правильно набрав имя и пароль, я окажусь в

директории "/root".

PATH=/bin:/usr/bin:.:/usr/local/bin:/usr/bin/X11 - эта переменная задает

последовательность файлов, которые просматривает "shell" в поисках команды.

Имена файлов разделяются здесь двоеточиями. Последовательность просмотра

соответствует очередности следования имен в тропе. Но первоначально поиск

происходит среди так называемых встроенных команд. В число встроенных команд

входят наиболее часто используемые команды, например "echo", "cd", "pwd",

"date". После этого система просматривает директорий "/bin", в котором могут

находиться команды "sh", "cp", "mv", "ls" и т.п. Затем директорий "/usr/bin" с

командами "cat", "сс", "expr", "nroff", "man" и многими другими. Далее поиск

происходит в текущем директории (".", или другое обозначение "пусто", т.е.""),

где скорее всего находятся написанные вами команды.

После набора командной строки и нажатия <Enter> "shell" (после выполнения

необходимых подстановок) распознает имя, соответствующее команде и осуществляет

ее поиск в директориях, перечисленных в PATH. Если команда размещена вне этих

директориев, она не будет найдена. Если присутствует несколько команд с

одинаковым именем, то вызвана будет та, которая расположена в директории,

просматриваемом первым.

PATH, как и прочие переменные, можно легко менять, добавляя, переставляя или

исключая директории.

IFS= (Внутренний Разделитель Полей) перечисляет символы, которые служат для

разделения слов (полей). Таковыми являются "пробел", "табуляция" и "перевод

строки", поэтому здесь слева от присваивания ничего не видно и занято две

строки.

LOGNAME=root - имя входа ("имя" пользователя).

MAIL=/var/spool/mail/root - имя файла, в который поступает (электронная) почта.

PWD=/root - имя текущего директория

PS1=${PWD}:" " - вид промтера. В данном случае в промптере будет выдаваться имя

текущего директория двоеточие и пробел. То есть здесь будет "/root: ".

PS2=> - этот промтер (здесь ">") используется как приглашение к продолжению

ввода (в очередной строке) незаконченной команды. Например, наберите открывающую

скобку "(" и после нажатия <Enter> в следующей строке вы увидите этот промптер.

Если пока не знаете, что дальше делать, наберите закрывающую скобку ")" и он

исчезнет.

SHELL=/bin/sh - эта переменная указывает оболочку, которую использует

пользователь. В данном случае используется стандартный shell ("sh").

Исходная среда устанавливается автоматически при входе в систему с

использованием файлов типа "/etc/rc" и "/etc/.profile".

Один из способов просто изменит среду (например, тропу поиска команд, вид

промтера, вид оболочки, цвет экрана и т.п.) можно, разместив эту информацию в

своем домашнем директории в специализированном файле ".profile"

(${HOME}/.profile), присвоив нужные значения переменным среды. То есть вызвать

это файл в редактор и написать, что пожелаете). Тогда при каждом вашем входе в

систему этот файл будет автоматически выполняться и устанавливать новую среду.

Этот файл должен ОБЯЗАТЕЛЬНО размещаться в вашем ДОМАШНЕМ директории (директории

входа).

Следует иметь в виду, что имена файлов, начинающиеся с точки, вообще имеют

особый статус. Так, они не выдаются на экран простой командой "ls" - необходимо

вызывать эту команду с флагом "-a". Кстати, и не уничтожаются огульно командой

"rm \*".

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 "$".

Важную роль при создании уникальных файлов играет специальная переменная "$$",

значение которой соответствует номеру процесса, выполняющего данный расчет.

Каждый новый расчет, выполняемый компьютером, инициирует один или несколько

процессов, автоматически получающих номера по порядку. Поэтому, используя номер

процесса в качестве имени файла, можно быть уверенным, что каждый новый файл

будет иметь новое имя (не запишется на место уже существующего). Достоинство

является и главным недостатком такого способа именования файлов. Неизвестно,

какие имена будут присвоены файлам. И, если в рамках данного процесса можно

найти файл "не глядя", т.е., обратившись к нему, используя $$, то потом такие

файлы можно легко потерять. Это создает дополнительные проблемы при отладке

программ.

3.8. Вызов интерпритатора

Вслед за регистрацией пользователя в системе (с помощью команды login)

вызывается интерпретатор языка SHELL. Если регистрационный справочник

пользователя содержит файл .profile, то прежде чем с терминала будет принята

хотя бы одна команда, интерпретатор выполняет этот файл (подразумевается, что

файл .profile содержит команды). При вызове могут указываться следующие ключи:

-c строка

Команды считываются из заданной строки.

-s

Команды читаются из стандартного файла ввода. Сообщения интерпретатора

записываются в стандартный файл диагностик.

-i

Интерактивный режим работы.

Если первым символом параметра "0" является знак -, то команды считываются из

файла .profile.

4. ПРОГРАММНЫЕ СТРУКТУРЫ

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,

# Это комментарий.

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.

4.1. 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 ]

мы будем пользоваться вторым вариантом, т.е. вместо того, чтобы писать перед

условием слово "test", будем заключать условие в скобки, что более привычно для

программистов.

На самом деле shell будет распознавать эту команду по открывающей скобке "[",

как слову, соответствующему команде "test". Между скобками и содержащимся в них

условием обязательно должны быть пробелы.

Пробелы должны быть и между значениями и символом сравнения или операции

В shell используются условия различных "типов".

УСЛОВИЯ ПРОВЕРКИ ФАЙЛОВ:

 -f file файл "file" является обычным файлом;

 -d file файл "file" - каталог;

 -с file файл "file" - специальный файл;

 -r file имеется разрешение на чтение файла "file";

 -w file имеется разрешение на запись в файл "file";

 -s file файл "file" не пустой.

УСЛОВИЯ ПРОВЕРКИ СТРОК:

 str1 = str2 строки "str1" и "str2" совпадают;

 str1 != str2 строки "str1" и "str2" не совпадают;

 -n str1 строка "str1" существует (непустая);

 -z str1 строка "str1" не существует (пустая).

Примеры.

x="who is who"; export x; [ "who is who" = "$x" ]; echo $?

0

x=abc ; export x ; [ abc = "$x" ] ; echo $?

0

x=abc ; export x ; [ -n "$x" ] ; echo $?

0

x="" ; export x ; [ -n "$x" ] ; echo $?

1

Кроме того, существуют два стандартных значения условия, которые могут

использоваться вместо условия (для этого не нужны скобки).

УСЛОВИЯ СРАВНЕНИЯ ЦЕЛЫХ ЧИСЕЛ:

 x -eq y "x" равно "y",

 x -ne y "x" неравно "y",

 x -gt y "x" больше "y",

 x -ge y "x" больше или равно "y",

 x -lt y "x" меньше "y",

 x -le y "x" меньше или равно "y".

СЛОЖНЫЕ УСЛОВИЯ:

Реализуются с помощью типовых логических операций:

 ! (not) инвертирует значение кода завершения.

 -o (or) соответствует логическому "ИЛИ".

 -a (and) соответствует логическому "И".

4.2. Условный оператор "if"

В общем случае оператор "if" имеет структуру

if условие

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.

Самая усеченная структура этого оператора

if условие

then список

fi

если выполнено условие (как правило это ком получен код завершения "0", то

выполняется "список", иначе он пропускается.

Примеры.

Пусть написан "if-1"

if [ $1 -gt $2 ]

then pwd

else echo $0 : Hello!

fi

Тогда вызов if-1 12 11 даст /home/sae/STUDY/SHELL

а if-1 12 13 даст if-1 : Hello!

4.3. Оператор вызова ("case")

Оператор выбора "case" имеет структуру:

case строка in

шаблон) список команд;;

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.

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.

4.4. 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.

Пусть команда "lsort" представлена командным файлом

for i in file\_1 file\_2 file\_3

do

proc\_sort $i

done

В этом примере имя "i" играет роль параметра цикла. Это имя можно рассматривать

как shell-переменную, которой последовательно присваиваются перечисленные

значения (i=file\_1, i=file\_2, i=file\_3), и выполняется в цикле команда

"proc\_sort".

Часто используется форма "for i in \*", означающая "для всех файлов текущего

каталога".

Пусть "proc\_sort" в свою очередь представляется командным файлом

cat $1 sort tee /dev/lp > ${1}\_sorted

т.е. последовательно сортируются указанные файлы, результаты сортировки

выводятся на печать ("/dev/lp") и направляются в файлы file\_1\_sorted

file\_2\_sorted и file\_3\_sorted

4.5. 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.

Команда "break [n]" позволяет выходить из цикла. Если "n" отсутствует, то это

эквивалентно "break 1". "n" указывает число вложенных циклов, из которых надо

выйти, например, "break 3" - выход из трех вложенных циклов.

В отличие от команды "break" команда "continue [n]" лишь прекращает выполнение

текущего цикла и возвращает на НАЧАЛО цикла. Она также может быть с параметром.

Например, "continue 2" означает выход на начало второго (если считать из

глубины) вложенного цикла.

Команда "exit [n]" позволяет выйти вообще из процедуры с кодом возврата "0" или

"n" (если параметр "n" указан). Эта команда может использоваться не только в

циклах. Даже в линейной последовательности команд она может быть полезна при

отладке, чтобы прекратит выполнение (текущего) расчета в заданной точке.

4.6. 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.

Пример.

until false

do

read x

if [ $x = 5 ]

then echo enough ; break

else echo some more

fi

done

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.

4.7. Пустой оператор

Пустой оператор имеет формат

:

Ничего не делает. Возвращает значение "0".".

4.8. Функции в shell

Функция позволяет подготовить список команд shell для последующего выполнения.

Описание функции имеет вид:

имя()

{

список команд

}

после чего обращение к функции происходит по имени. При выполнении функции не

создается нового процесса. Она выполняется в среде соответствующего процесса.

Аргументы функции становятся ее позиционными параметрами; имя функции - ее

нулевой параметр. Прервать выполнение функции можно оператором "return [n]", где

(необязательное) "n" - код возврата.

4.9. Обработка прерываний ("trap")

Бывает необходимо защитить выполнение программы от прерывания.

Наиболее часто приходится встречаться со следующими прерываниями,

соответствующими сигналам:

 0выход из интерпретатора,

 1отбой (отключение удаленного абонента),

 2прерывание от <Del>,

 9уничтожение (не перехватывается),

 15окончание выполнения.

Для защиты от прерываний существует команда "trap", имеющая формат:

trap 'список команд' сигналы

Если в системе возникнут прерывания, чьи сигналы перечислены через пробел в

"сигналы", то будет выполнен "список команд", после чего (если в списке команд

не была выполнена команда "exit") управление вернется в точку прерывания и

продолжится выполнение командного файла.

Например, если перед прекращением по прерываниям выполнения какого то командного

файла необходимо удалить файлы в "/tmp", то это может быть выполнено командой

"trap":

tarp 'rm /tmp/\* ; exit 1' 1 2 15

которая предшествует прочим командам файла. Здесь, после удаления файлов будет

осуществлен выход "exit" из командного файла.