# Практическое занятие. Программирование в Shell.

Целью занятия является формирование и закрепление навыков по написанию Shell - сценариев для Linux систем.

Задачи:

1. Научиться объединять заданные команды в рабочие Shell-сценарии.

2. Научиться определять и добавлять необходимые команды, правила и цели при создании Make-файлов.

**Необходимые теоретические сведения.**

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 (если угодно - она начинает взаимодействовать с вами). 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(из наиболее популярных в последнее время) и другие. Более того,каждый пользователь может создать свой командный язык. Может одновременно на одном экземпляре операционной системы работать с разными командными языками.

ОБРАТИТЕ ВНИМАНИЕ. shell - это одна из многих команд UNIX. То есть в набор команд оболочки (интерпретатора) "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 "f1":

echo Hello!

то это будет обычный текстовый файл, содержащий команду "echo", которая при выполнении выдает все написанное правее ее на экран. Можно сделать файл "f1" выполняемым с помощью команды "chmod 755 f1". Но его можно ВЫПОЛНИТЬ, вызвав явно команду (!) "sh" ("shell"):

sh f1

или

sh < f1

Файл можно выполнить и в текущем экземпляре "shell". Для этого существует специфическая команда "." (точка), т.е.

. f1

ВАЖНОЕ ПРЕДУПРЕЖДЕНИЕ. Не начинайте командные файлы с символа "#", хотя естественно начинать его с комментария. Дело в том, что такой командный файл в оболочке C-Shell ("csh") будет интерпретирован как выполняемый в "csh", в результате будет активизирован. интерпретатор "csh".

СОВЕТ. Начинайте командный sh-файл с пустой строки или пустого оператора ":".

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 свой пользователь со своей командной оболочкой...

Стержневым элементом языка shell является команда.

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, tar, dd, find и ряд других.

Как правило (но не всегда), первое слово (т.е. последовательность символов до пробела, табуляции или конца строки) 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.

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 амперсенд) на экран выводится номер процесса, соответствующий выполняемой команде, и система, запустив этот фоновый процесс, вновь выходит на диалог с пользователем.

Например, наберем (экзотическую) 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:

--------------------------------

| $ 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

k1 && k2; k3

где k1, k2 и k3 - какие-то команды, то "k2" будет выполнена только при успешном завершении "k1"; после любого из исходов обработки "k2" (т.е. "k2" будет выполнена, либо пропущена) будет выполнена "k3".

k1 && {k2; k3}

Здесь обе команды ("k2" и "k3") будут выполнены только при успешном завершении "k1".

{k1; k2} &

В фоновом режиме будет выполняться последовательность команд "k1" и "k2".

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 иметь имя привилегированного пользователя).

ПРЕДУПРЕЖДЕНИЕ. Если параллельно обрабатывается или создается файл с ОДНИМ именем (например, несколько пользователей вызвали в редактор один и тот же файл), то в системе продолжит существование тот вариант файла, который возвращен (записан) в систему последним. Это частая ошибка пользователей персональных компьютеров, которые редактируют один файл параллельно с нескольких экранов.

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.

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 >f1

команда "ls" сформирует список файлов текущего каталога и поместит его в файл "f1" (вместо выдачи на экран). Если файл "f1" до этого существовал, то он будет затерт новым.

pwd >>f1

команда pwd сформирует полное имя текущего каталога и поместит его в конец файла "f1", т.е. ">>" добавляет в файл, если он непустой.

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

wc -l f4 и wc -l >f4 f2

Данный конвейер из файла "f1" ("cat") выберет все строки, содержащие слово "result" ("grep"), отсортирует ("sort") полученные строки, а затем пронумерует ("cat -b") и выведет результат в файл "f2".

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

f1 < /dev/null обнулит файл "f1".

sort f1 | tee /dev/lp | tail -20

В этом случае будет отсортирован файл "f1" 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 и 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 f1 f2

которая должна выдать на экран последовательно содержимое файлов "f1" и "f2", выдаст вам, например, следующее

111111 222222

cat: f2: No such file or directory

где 111111 222222 - содержимое файла "f1", а файл "f2" 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 f1 f2 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 f1 f2 >>ff 2>ff

Можно указать не только какой из стандартных файлов перенаправлять, но и в какой стандартный файл осуществить перенаправление.

cat f1 f2 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.

Генерация имен файлов.

При генерации имен используют метасимволы:

\* - произвольная (возможно пустая) последовательность символов;

? - один произвольный символ;

[...] - любой из символов, указанных в скобках перечислением и/или с указанием диапазона;

cat f\* - выдаст все файлы каталога, начинающиеся с "f";

cat \*f\* - выдаст все файлы, содержащие "f";

cat program.? - выдаст файлы данного каталога с однобуквенными расширениями, скажем "program.c" и "program.o", но не выдаст "program.com";

cat [a-d]\* - выдаст файлы, которые начинаются с "a", "b", "c", "d". Аналогичный эффект дадут и команды "cat [abcd]\*" и "cat [bdac]\*".

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

ВНИМАНИЕ. Операция умножения ("\*") обязательно должна быть заэкранирована, поскольку в shell этот значек воспринимается, как спецсимвол, означающий, что на это место может быть подставлена любая последовательность символов.

Следует обратить также внимание на обязательные пробелы, отделяющие переменные и знаки операций.

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

Переменные локальны в рамках процесса, в котором они объявлены, т.е. где им присвоены значения (описание переменных отсутствует - они все одного типа). Для того, чтобы они были доступны и другим порождаемым процессам, надо передать их явным образом. Для этого используется встроенная команда "export".

Пример.

Пусть расчет (командный файл) "p", имеющий вид:

# расчет p

echo Расчет p

varX=0 varY=1

echo varX=$varX varY=$varY

export varY

p1 # вызов расчета p1

p2 # вызов расчета p2

echo Снова расчет p: varX=$varX varY=$varY

вызывает командные файлы "p1" и "p2", имеющие вид:

# расчет p1

echo Расчет p1

echo varX=$varX varY=$varY

varX=a varY=b

echo varX=$varX varY=$varY

export varX

# расчет p2

echo Расчет p2

echo varX=$varX varY=$varY

varX=A varY=B

echo varX=$varX varY=$varY

export varY

На экран будут выданы следующая информация:

Расчет p

varX=0 varY=1

Расчет p1

varX= varY=1

varX=a varY=b

Расчет p2

varX= varY=1

varX=A varY=B

Снова расчет p: varX=0 varY=1

Из примера видно, что значения переменных экспортируются только в вызываемые расчеты.

Экспортировать переменные можно и командой "set" с флагом "-a".

**Практическая часть**

Своеобразный подход к параметрам дает команда "set". Например, фрагмент сценария:

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

Примеры работы Shell-сценариев.

###Пример 1.

# Использование 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 Сойдет ! ;;

2) echo Готовься лучше ! ;;

\*) echo Нет такой оценки! ;;

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.

Структуру "case" можно использовать для реализации простейших меню:

Пример 2. Реализация меню с помощью команды "case".

echo "Назовите файл, а затем (через пробел)

наберите цифру, соответствующую требуемой

обработке:

1 - отсортировать

2 - выдать на экран

3 - определить число строк "

read x y # x - имя файла, y - что сделать

case $y in

1) sort < $x ;;

2) cat < $x ;;

3) wc -l < $x ;;

\*) echo "

Мы не знаем

такой команды ! " ;;

esac

Пример 3 "Ожидание полдня" иллюстрирует возможность использовать в условии вычисления.

until date | grep 12:00:

do

sleep 30

done

Здесь каждые 30 секунд выполняется командная строка условия. Команда "date" выдает текущую дату и время. Команда "grep" получает эту информацию через конвейер и пытается совместить заданный шаблон "12:00:" с временем, выдаваемым командой "date". При несовпадении "grep" выдает код возврата "1", что соответсвует значению "ложь", и цикл "выполняет ожидание" в течение 30 секунд, после чего повторяется выполнение условия. В полдень (возможно с несколькими секундами) произойдет сравнение, условие станет истинным, "grep" выдаст на экран соответсвующую строку и работа цикла закончится.

Утилита Make

Make - утилита GNU для обработки групп программ. В том числе она может управлять трансляцией и компоновкой. Достаточно записать нужные команды в управляющий Makefile один раз, а в дальнейшем нужно будет только выполнить команду make, чтобы перекомпилировать всю программу целиком. Поведение make описывается файлом Makefile, который должен присутствовать в текущем каталоге. Это текстовый файл, который может содержать:

• комментарии;

•правила;

•макроопределения.

Make не будет перекомпилировать весь проект целиком, а перекомпилирует только те файлы, которые устарели, то есть исходник новее соответствуующего ему объектного файла. Это важно для больших проектов, состоящих из множества исходников в разных файлах. Кроме собственно компиляции в Makefile можно вписать и команды для автоматизации прочих операций (например установки, прошивки и т.п.). Если на каком-либо этапе сборки произошла ошибка, то утилита Make останавливает сборку.

Пример:

all: prog

# чтобы выполнить цель all нужен файл prog

# правило компоновки исполняемого файла из объектного

# чтобы получить prog нужен prog.o

prog: prog.o

ld -m elf\_i386 prog.o -o prog

# правило сборки ассемблерного кода

# чтобы получить prog.o нужен prog.asm

prog.o: prog.asm

nasm -f elf32 prog.asm -o prog.o

clean:

rm \*.o

rm prog

В примере созданы два правила с именами ("целями") all и clean. Первое из них выполняет трансляцию и сборку программы prog.asm, а второе удаляет все созданные предыдущим правилом файлы. Таким образом, если набрать в командной строке

make all или просто make, то будет создан исполняемый файл из prog.asm. Если набрать make clean, то проект будет очищен: будут удалены все файлы, создаваемые nasm и ld. Для более подробной информации см. man make.

Примечание: утилита make необычно требовательна к содержимому make-файла. Ей необходимо, чтобы команды в правилах (в отличие от целей) начинались с отступа, и отступ обязательно должен создаваться символом табуляции, а не пробелами. Чтобы получить табуляцию в редакторе mcedit, нужно нажать Tab два раза.

Задание для закрепления практических навыков:

1. Модифицировать по своему усмотрению примеры 1, 2 и 3 сценариев и продемонстрировать преподавателю их работоспособность.

2. Объяснить преподавателю назначение используемых команд и параметров.

3. Создать Makefile, который будет автоматизировать сборку ассемблерной программы из лабораторной работы 5.1.