Linux Shell Programming

Prachi Pandey

C-DAC Bangalore

Redirection

- Input redirection
  - Standard input (stdin) device is keyboard
  - “<” is the input redirection operator
- Output redirection
  - Standard output (stdout)device is screen
  - “>” is the output redirection operator
  - “>>” appends output to an existing file
- Error redirection
  - Error is displayed on screen by default
  - Error redirection is routing the error to a file other than the screen

Regular expressions

- RegEx are a set of characters to check patterns in strings
- Basic regular expressions
  - . Replaces any character
  - * Replaces 0 or more characters
  - ^ Matches start of the string
  - $ Matches end of the string
  - * {n} Matches the preceding character appearing ‘n’ times exactly
- Brace expansion

Shell

- Shell is an active instance of a program (a process) that takes commands typed by the user and calls the OS(using system library routines) to run those commands

- Shell acts as a wrapper around the OS – hence, known by the term shell

- Shell is a special utility that plays several roles – it is a versatile utility
  - Command interpreter
  - Command editor
  - Job controller
  - Programming language interpreter
  - CLI – command line interface to the OS

Different Shells

- Bourne shell(sh)
  - It was developed by Steve Bourne (AT&T) in 1977
- Had many new features and was foundation for many newer derivatives
  - C shell (csh) - 1978
  - Korn shell (ksh) - 1983
  - TENEX C Shell (tcsh) - 1983
  - Bash (bash) -1989

Bash shell features

- Compatible with Bourne shell
- Job control
- History list
- Command-line editing
- Aliases
- Functions
- Arrow keys for command editing
- Control structures for conditional testing and iteration
- Basic debugging and exception handling

Environment Variables

- Dynamic values
- Exist in every OS
- Can be created, edited, saved and deleted
- Change the way software/programs behave
- Eg.
    - PATH
    - USER
    - HOME
- Use **export** command to set the environment variable
- echo $VARIABLE : To display value of a variable

PATH Environment Variable

PATH: The search path for commands. It is a colon-separated list of directories that are searched when you type a command.

Usually, we type in the commands in the following way:

```
$ ./command
```

By setting PATH=$PATH:. our working directory is included in the search path for commands, and we simply type:

```
$ command
```

If we include the following lines in the ~/.bash_profile:

```
PATH=$PATH:$HOME/bin
export PATH
```

we obtain that the directory /home/userid/bin is included in the search path for commands.

Shell script

- Program which interprets user commands through CLI like terminal

- Shell scripting is writing a series of commands for the shell to execute

- Helps creating complex programs containing conditional statements, loops and functions

Basic Shell Programming

- A script is a file that contains shell commands
  - data structure: variables
  - control structure: sequence, decision, loop

- Shebang line for bash shell script:

  #! /bin/bash

  #! /bin/sh

- to run:
  - make executable: **% chmod +x script**
  - invoke via: **% ./script**

Bash program

- We write a program that copies all files into a directory, and then deletes the directory along with its contents. This can be done with the following commands:

```
$ mkdir temp
$ cp *.log temp
$ rm *.log
```

- Instead of having to type all that interactively on the shell, write a shell program instead:

```
$ cat log_temp.sh
        #!/bin/bash
        # this script copies log files to temp dir
        mkdir temp
        cp *.log temp
        rm *.log
        echo "Log files copied"
```

Shell Metacharacters

Symbol	Meaning
>	Output redirection,
>>	Output redirection
<	Input redirection
*	File substitution wildcard; zero or more characters
?	File substitution wildcard; one character
[ ]	File substitution wildcard; any character between brackets
\`cmd\`	Command Substitution
$(cmd)	Command Substitution
|	The Pipe (|)
;	Command sequence, Sequences of Commands
||	OR conditional execution
&&	AND conditional execution
( )	Group commands, Sequences of Commands
&	Run command in the background, Background Processes
#	Comment
$	Expand the value of a variable
\	Prevent or escape interpretation of the next character
<<	Input redirection

Variables

- Can use variables as in any programming languages.
- Values are always stored as strings
- Mathematical operators in the shell language convert variables to numbers for calculations.
- No need to declare a variable
- Format for setting a value to a variable:

  Name = Value
- Access the variable by $ symbol
- Rules
  - No space
  - No number in the beginning
  - No $ in name
  - Case sensitive
- Example

```
#!/bin/bash
STR="Hello World!"
echo $STR
```

Variables

- The shell programming language does not type-cast its variables.
- count=0

  count=Sunday

- It is recommended to use a variable for only a single TYPE of data in a script.

- \ is the bash escape character and it preserves the literal value of the next character that follows.
  - $ echo *

Single and double quote

- When assigning character data containing spaces or special characters, the data must be enclosed in either single or double quotes.

- Using double quotes to show a string of characters will allow any variables in the quotes to be resolved

```
$ var=“test string”
$ newvar=“Value of var is $var”
$ echo $newvar
Value of var is test string
```

- Using single quotes to show a string of characters will not allow variable resolution

```
$ var=’test string’
$ newvar=’Value of var is $var’
$ echo $newvar
Value of var is $var
```

Command Substitution

- The backquote "`" is different from the single quote "´". It is used for command substitution: `command`

```
$ LIST=`ls`
$ echo $LIST
hello.sh read.sh
```

- We can also perform the command substitution by means of $(command)

```
$ LIST=$(ls)
$ echo $LIST
hello.sh read.sh

$ rm $( find / -name "*.tmp" )
```

Read command

- The read command allows you to prompt for input and store it in a variable.

- Example:

```
#!/bin/bash
echo -n "Enter name of file to delete: "
read file
rm $file
```

- Line 2 prompts for a string that is read in line 3.

Shell parameters

- Positional parameters are assigned from the shell's argument when it is invoked. Positional parameter "N" may be referenced as "${N}", or as "$N" when "N" consists of a single digit.

Parameter	Meaning
$0	Name of the current shell script
$1-$9	Positional parameters 1 through 9
$#	The number of positional parameters
$*	All positional parameters, "$*" is one string
$@	All positional parameters, "$@" is a set of strings
$?	Return status of most recently executed command
$$	Process id of current process

Examples: Command Line Arguments

```
% set blue green red yellow
        $1   $2    $3    $4
% echo $*
blue green red yellow
% echo $#
4
% echo $1
blue
% echo $3 $4
red yellow
```

The 'set' command can be used to assign values to positional parameters

Arithmetic Evaluation

- The let statement can be used to do mathematical functions:

```
$ let X=10+2*7
$ echo $X
24
$ let Y=X+2*4
$ echo $Y
32
```

- An arithmetic expression can be evaluated by $[expression] or $((expression)) or "expr" command

```
$ echo "$((123+20))"
143

$ TEMP=$[123+20]
$ echo "$[123*$TEMP]"
17589
echo $(expr $x + $y )
```

Arithmetic Evaluation

- An arithmetic expression can be evaluated by $[expression] or $((expression)) or "expr" command

```
$ echo "$((123+20))"
143
$ TEMP=$[123+20]
$ echo "$[123*$TEMP]"
17589
echo $(expr $x + $y )
```

For floating point arithmetic operations, we need to use a tool "bc (basic calculator)"

```
$ echo "$x+$y" | bc
```

Arithmetic Evaluation

- Available operators: +, -, /, *, %

- Example : Accept two numbers as input, perform +,-,/,*,% functions on them and print the output.

Solution

```
$ cat arithmetic.sh
#!/bin/bash
echo -n "Enter the first number: "; read x
echo -n "Enter the second number: "; read y
add=$(($x + $y))
sub=$(($x - $y))
mul=$(($x * $y))
div=$(($x / $y))
mod=$(($x % $y))
# print out the answers:
echo "Sum: $add"
echo "Difference: $sub"
echo "Product: $mul"
echo "Quotient: $div"
echo "Remainder: $mod"
```