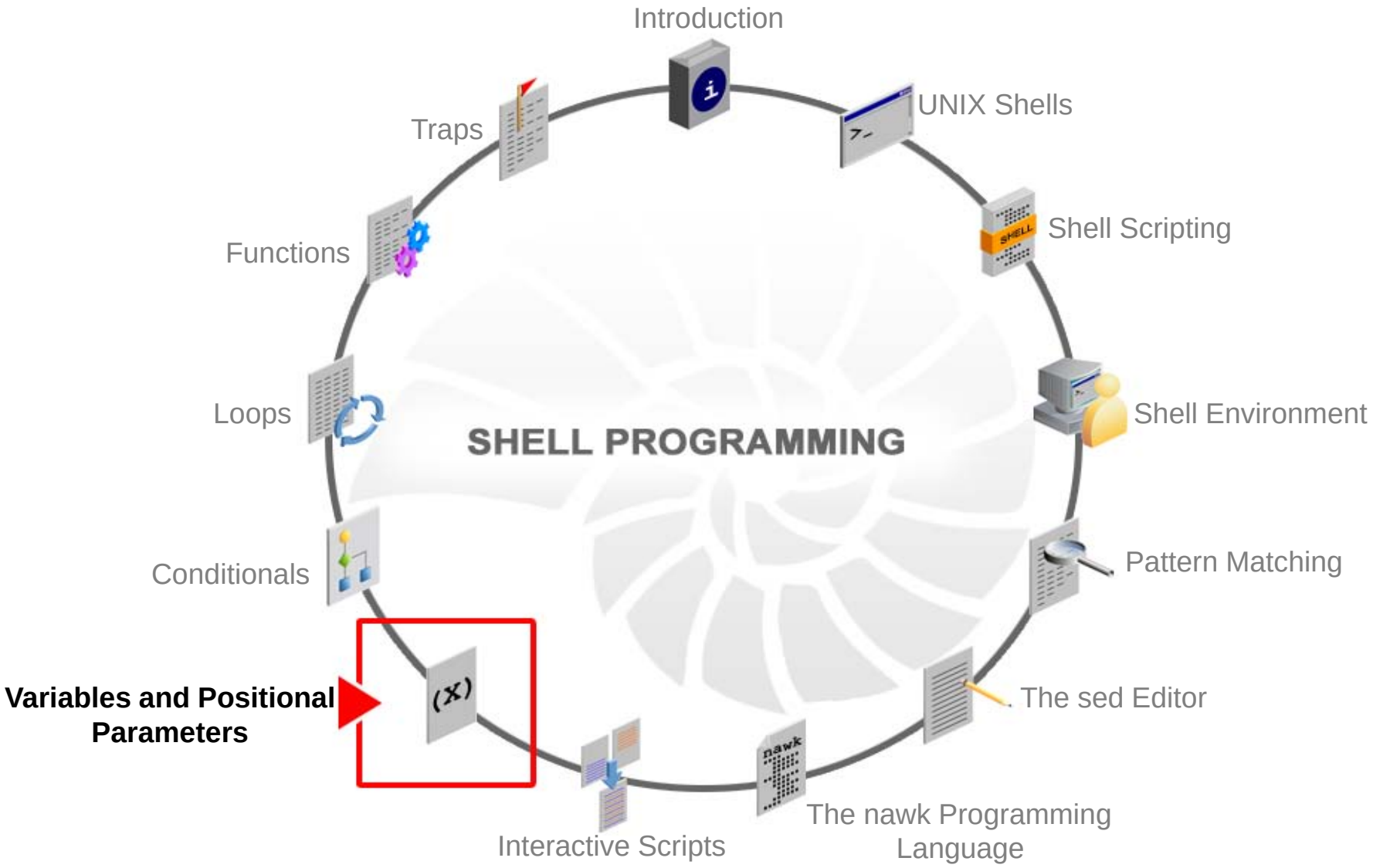
9

Variables and Positional Parameters



Objectives

After completing this lesson, you should be able to:

- Describe the various types of scripting variables
- Define positional parameters for accepting user input

Agenda

- Describing the various types of scripting variables
- Defining positional parameters for accepting user input

Scripting Variables

- In the bash shell, there are four variable types:
  - String
  - Integer
  - Constant
  - Array
- All variables are of type string unless declared otherwise.
- A variable data type determines the values that can be assigned to a variable.

Accessing Variable Values

- When you access the value of a variable by preceding its name with a $, you might need to isolate the variable name from the characters that immediately follow it.
- For example:

```
$ flower=rose
$ printf "$flower $flowers $flowerbush"
rose
$ printf "$flower ${flower}s ${flower}bush"
rose roses rosebush
```

The `typeset` Statement

- The `typeset` statement sets variable attributes.
- In some cases, it changes a variable value, such as a right- or left-justification.
- Following are some of the `typeset` statement options:

Syntax	Description
`typeset -LZ var`	Strips leading zeros from the string `var`.
`typeset -Lnum var`	Left-justifies `var` within the field width specified by num.
`typeset -Rnum var`	Right-justifies `var` within the field width specified by num.
`typeset -i var`	It specifies that `var` can contain only integer values.
`typeset -r var`	It specifies that `var` is read-only; the value in `var` cannot be changed by subsequent assignment.

The `typeset` Statement: Example

```
$ cat strman1.sh
#!/bin/bash
# Script name: strman1.sh
typeset -r word="happy“
typeset -l word1="depressed“

print "123456789"
print "$word"
print
print "123456789"
print "$word1"

$ ./strman1.sh
123456789
    happy

123456789
depressed
```

The typeset Statement: Example

```
$ cat strman2.sh
#!/bin/bash
# Script name: strman2.sh
typeset -r word="happy“
typeset -r word1="depressed"

print "123456789"
print $word
Print
print "123456789"
print $word1

$ ./strman2.sh
123456789
happy

123456789
depressed
```

The `declare` Statement

- The `declare` statement is an alternative to the `typeset` statement.
- Syntax:

```
declare [-afFrxi] [-p] [name[=value]]
```

- Following are some of the `declare` statement options:

Options	Description
`-a`	Treat each name as an array variable.
`-f`	Use function names only.
`-F`	Inhibit the display of function definitions.
`-i`	Treat the variable i as an integer.
`-r`	Make names read-only.

The declare Statement: Example

```
$ cat strman1.sh
!/bin/bash
# Script name: strman1.sh
declare -r word="happy“
declare -l word1="depressed"

echo "123456789"
echo "$word"
echo

echo "123456789"
echo "$word1"

$ ./strman1.sh
123456789
happy

123456789
depressed
```

Removing Portions of a String

Syntax	Description
`${str_var%pattern}`	Removes the smallest right-most substring of string `str_var` that matches pattern
`${str_var%%pattern}`	Removes the largest right-most substring of string `str_var` that matches pattern
`${str_var#pattern}`	Removes the smallest left-most substring of string `str_var` that matches pattern
`${str_var##pattern}`	Removes the largest left-most substring of string `str_var` that matches pattern

Declaring an Integer Variable

- You can declare an integer variable in two ways:
  - By using `typeset –i` or `declare -i`
  - By using integer in front of the variable name

```
typeset -i int_var1[=value] int_var2[=value] ...
int_varn[=value]
```

```
declare -i int_var1[=value] int_var2[=value] ...
int_varn[=value]
```

- An integer variable can have only integer numbers assigned to it.
- An assignment of a number with a decimal part results in the decimal being truncated.

Using Integer Variables: Examples

```
$ typeset -i num
$ num=5
$ print $num
5
$ typeset -i num
$ num=25.34
$ print $num
25
$ typeset -i num # base 10 integer
$ num=27
$ print $num
27
$ typeset -i8 num # change to base 8
$ print $num
8#33
$ num=two
/usr/bin/ksh: two: bad number
$ print $num
8#33
```

Using Integer Variables: Examples

```
$ typeset -i num
$ num=5
$ print $num
5

$ declare -i num
$ num=5
$ echo $num
25

$ typeset -i num # base 10 integer
$ num=27
$ print $num
27

$ declare –i number
$ number=3
$ echo "Number = $number”
number=3
```

Arithmetic Operations on Integer Variables

Operator	Operations	Example	Result
+	Addition	((x = 24 + 25))	49
-	Subtraction	((x = 100 - 25))	75
*	Multiplication	((x = 4 * 5))	20
/	Division	((x = 10 / 3))	3
%	Modulo (remainder)	((x = 10 % 3))	1
#	Base	2#1101010 16#6A	10#106
<<	Shift bits left	((x = 2#11 << 3))	2#11000
>>	Shift bits right	((x = 2#1001 >> 2))	2#10
&	Bit-wise AND	((x = 2#101 & 2#110))	2#100
|	Bit-wise OR	((x = 2#101 | 2#110))	2#111
^	Bit-wise exclusive OR	(((x = 2#101 ^ 2#110))	2#11

Creating Constants

- The value of a constant (read-only) variable cannot be changed.
- A variable can be made read-only with the following syntax:

```
readonly var[=value]
```

Declaring Arrays

- Arrays:
  - Contain multiple values
  - Are created when you use them
  - Are zero-based, the first element is indexed with the number `0`
- You can create arrays of strings or integers.
- By default, an array contains strings.
- To create an array of integers:
  - Declare a variable as an integer
  - Assign integer values to the array elements

```
integer my_array
my_array[1]=5
my_array[12]=16
```

Using Arrays: Examples

- Creating an array of three strings:

```
arr[0]=big
arr[1]=small
arr[2]="medium sized”
```

- Creating an array of three strings with the `set` command:

```
set arr big small "medium sized“
```

- Creating an array of five integers:

```
declare –a integer num
num[0]=0
num[1]=100
num[2]=200
num[3]=300
num[4]=400
```

Using Arrays: Examples

- Printing the number of array elements in array `num`:

```
$ printf ${#num[*]}
5
```

- Printing the values of all array elements in array `arr`:

```
$ printf ${arr[*]}
big small medium sized
```

- Destroying an array `arr`:

```
$ unset arr
```

Quiz

All variables, whether strings, integers, constants, or arrays are of type string unless declared otherwise.

a. True
b. False

Agenda

- Describing the various types of scripting variables
- Defining positional parameters for accepting user input

Positional Parameters

- The operation of a script varies depending on what arguments are given to the script on the command line.
- The shell automatically assigns special variable names, called *positional parameters*, to each such argument.

Parameter Name	Description
`$0`	The name of the script
`$1`	The first argument to the script
`$2`	The second argument to the script
`$9`	The ninth argument to the script
`$#`	The number of arguments to the script
`$@`	A list of all arguments to the script where each parameter is seen as a word
`$*`	A list of all arguments to the script where each parameter is quoted as a string

The `shift` Statement

- By default, the `shift` statement shifts the values held in the positional parameter.
- The `shift` statement drops the first value (`$1`) and shifts all other values to the left.

The shift Statement: Example

```
$ cat argtest.sh
#!/bin/bash
# Script name: argtest.sh

echo '$#: ' $#
echo '$@: ' $@
echo '$*: ' $*
echo
echo '$1 $2 $9 $10 are: ' $1 $2 $9 $10
echo

shift
echo '$#: ' $#
echo '$@: ' $@
echo '$*: ' $*
echo
echo '$1 $2 $9 are: ' $1 $2 $9

shift 2
echo '$#: ' $#
echo '$@: ' $@
echo '$*: ' $*
echo
echo '$1 $2 $9 are: ' $1 $2 $9
echo '${10}: ' ${10}
```

The shift Statement: Example

```
$ ./argtest.sh a b c d e f g h i j k l m n
'14: ' 14
'a b c d e f g h i j k l m n: ' a b c d e f g h i j k l m n
'a b c d e f g h i j k l m n: ' a b c d e f g h i j k l m n

'a b i a0 are: ' a b i a0

'13: ' 13
'b c d e f g h i j k l m n: ' b c d e f g h i j k l m n
'b c d e f g h i j k l m n: ' b c d e f g h i j k l m n

'b c j are: ' b c j
'11: ' 11
'd e f g h i j k l m n: ' d e f g h i j k l m n
'd e f g h i j k l m n: ' d e f g h i j k l m n

'd e l are: ' d e l
'm: ' m
./argtest.sh: bad substitution
```

The `set` Statement

- The `set` statement allows you to assign values to positional parameters.
- Syntax:

```
set value1 value2 ... valueN
```

Where:

- `$1` has value `value1`
- `$2` has value `value2`, and so on
- `$0` is the name of the script

- Use the `set` statement to create a parameter list by using the statement or variable substitution.

```
set $(cal)
set $var1
```

The set Statement: Example

```
$ cat pospara2.sh
#!/bin/bash
# Script name: pospara2.sh

print "Executing script $0 \n"
print "$1 $2 $3"

set uno duo tres
print "One two three in Latin is:"
print "$1"
print "$2"
print "$3 \n"

textline="name phone address birthdate salary"
set $textline
print "$*"
print 'At this time $1 =' $1 'and $4 =' $4 "\n"

set -s
print "$* \n"

set --
print "$0 $*"
```

The set Statement: Example

```
$ ./pospara2.sh a b c
Executing script ./pospara2.sh

a b c
One two three in Latin is:
uno
duo
tres

name phone address birthdate salary
At this time $1 = name and $4 = birthdate

address birthdate name phone salary
./pospara2.sh
…
…
…
<output omitted>
```

The Values of the"$@" and "$*" Positional Parameters

The values of $@ and $* are identical, but the values of "$@“ and "$*" are different.

- "$@" expands to "$1" "$2" "$3" ... "$n“
- "$*" expands to "$1x$2x$3x...$n“

Using the Values of “$@” and “$*”: Example

```
$ cat arginfo.sh

#!/bin/bash
# Purpose: Print out information about the positional parameters.
#
# Name: arginfo.sh
echo "The name of the script is: $0"

echo "Positional Parameter      Value"
echo "     \$1                    $1"
echo "     \$2                    $2"
echo "     \${10}                 ${10}"

echo "The number of positional parameters is: $#"
echo
if (( $# >= 5 )); then
     echo "Now shift all values over by 5 places."
     shift 5

     echo "Positional parameters remaining:"
     echo $*
     echo

     echo "The number of positional parameters is: $#"
fi
```

Using the Values of “$@” and “$*”: Example

```
$ ./arginfo.sh
The name of the script is: ./arginfo.sh
Positional Parameter      Value
     $1
     $2
     ${10}
The number of positional parameters is: 0
```

Quiz

The `$0` positional parameter refers to which of the following?

a. The list of all arguments to the script
b. The first argument to the script
c. The name of the script
d. The number of arguments to the script

Summary

In this lesson, you should have learned how to:

- Describe the various types of scripting variables
- Define positional parameters for accepting user input

Practice 9 Overview: Variables and Positional Parameters

This practice covers the following topic:

- Using Advanced Variables, Parameters, and Argument Lists