Date: 21-08-2025

Variable Declaration

```
<<comment
#!/bin/bash
name="John Doe"
echo "Hello, $name!"
number=42
echo "The number is $number"
comment
number=42
echo "The number is $number"
comment


# Display the PATH environment variable

        echo "Your PATH is $PATH"


# Define a local variable in a function

my_function() {
  local local_var="I am local"
  echo $local_var
}
my_function


# String concatenation

greeting="Hello, "
name="World"
echo "$greeting$name"




# Natural numbers
num1=5
num2=10
echo “$num1 $num2"
```

```
# String example
greeting="Hello, World!"
name="Alice"
full_greeting="$greeting, $name!"
echo $full_greeting
```

Mathematical Expressions

```
# Natural numbers
num1=5
num2=10
sum=$((num1 + num2))
echo "The sum is $sum"


# Number example
num1=5
num2=10

s= $num1 + $num2  # error
d= $num1 - $num2   # error
p= $num1 * $num2   # error
q= $num1 / $num2   # error

<<comm
```

The expression sum=$((num1 + num2)) in a shell script is not performing typecasting in the sense of explicitly converting a data type from one to another, like you would see in strongly-typed programming languages such as C++ or Java. Instead, it's utilizing arithmetic expansion within the shell. Here's how it works:

Variable Interpretation:

When num1 and num2 are used within the $(()) construct, the shell implicitly treats their values as integers for the purpose of the arithmetic operation. If the variables contain values that cannot be interpreted as integers (e.g., "hello"), an error will typically occur, or they might be treated as zero depending on the shell's strictness.

Calculation:

The + operator performs integer addition on these interpreted numeric values.

Result Assignment:
The result of the arithmetic operation is then assigned back to the sum variable, where it is stored as a string, as all shell variables are fundamentally strings.

Therefore, while the shell interprets the variable contents as numbers for calculation, it's not a formal "typecast" operation that changes the underlying data type of the variable
itself. Shell variables remain string-based, even when their contents are treated numerically during arithmetic expansion
comm

```
sum=$((num1 + num2))
difference=$((num2 - num1))
product=$((num1 * num2))
quotient=$((num2 / num1))

echo sum
echo difference
echo product
echo quotient

echo "Sum: $sum, Difference: $difference, Product: $product, Quotient: $quotient"
```

Array Creation

```
#!/bin/bash

# To declare a static Array

arr=("Jayesh" "Shivang" "1" "Vipul" "Nishant" "2")

# To print all elements of the array

echo "All elements of the array:"
echo ${arr[@]}
echo ${arr[*]}
echo $arr

# To print the first element

echo "The first element:"
echo "${arr[0]}"
```

```
read -p "Read a number:" number
echo $number
arr[0]=$number
echo "The revised array is ${arr[*]}"
```

Different Operators

Arithmetic Operators

+: Addition
-: Subtraction
*: Multiplication
/: Division
%: Modulus (remainder of division)
For exponentiation, use external tools like bc or awk.

Logical Operators

&&: Logical AND
||: Logical OR
!: Logical NOT

File Test Operators

-e: Checks if a file exists
-d: Checks if a directory exists
-f: Checks if a file is a regular file
-s: Checks if a file is not empty

Comparison Operators

-eq: Equal to
-ne: Not equal to
-lt: Less than
-le: Less than or equal to
-gt: Greater than
-ge: Greater than or equal to

String Comparison Operators

=: Equal to
!=: Not equal to
<: Less than, in ASCII alphabetical order
>: Greater than, in ASCII alphabetical order

Loop Concepts

```
# Basic if statement
num=15
if [ $num -gt 10 ]; then
  echo "Number is greater than 10"
fi
```

```
# If...else statement
num=8
if [ $num -gt 10 ]; then
  echo "Number is greater than 10"
else
  echo "Number is 10 or less"
fi
```

```
# If...elif...else statement
num=10
if [ $num -gt 10 ]; then
  echo "Number is greater than 10"
elif [ $num -eq 10 ]; then
  echo "Number is exactly 10"
else
  echo "Number is less than 10"
fi
```

```
# Nested if statement
```

```
num=5
if [ $num -gt 0 ]; then
  if [ $num -lt 10 ]; then
    echo "Number is between 1 and 9"
  fi
fi



# For loop example
for i in {1..5}; do
  echo "Iteration $i"
done


# While loop example
count=1
while [ $count -le 5 ]; do
  echo "Count is $count"
  ((count++))
done


# Until loop example
count=1
until [ $count -gt 5 ]; do
  echo "Count is $count"
  ((count++))
done


# Break and continue example
for i in {1..5}; do
  if [ $i -eq 3 ]; then
    continue
  fi
  echo "Number $i"
  if [ $i -eq 4 ]; then
    break
  fi
done
```

```
# Nested loops example
for i in {1..3}; do
  for j in {1..2}; do
    echo "Outer loop $i, Inner loop $j"
  done
done
```

Function creation

```
greet() {
  local name=$1
  echo "Hello, $name!"
}
greet "Alice"
```

```
greet()
{
  local name=$1
  local greet=$2
  echo "Hello, $name! $greet!"
}
greet "Alice" "Good Morning"
```

```
add() {
  local sum=$(($1 + $2))
  echo $sum
}
result=$(add 5 3)
echo "The sum is $result"
```

Date: 25-08-2025

Skill Enhancement Class Notes

Blog Source: https://www.hostinger.com/in/tutorials/linux-commands

What is an array?

What is a Set?

Home Task:

Differences between set and array:
- uniqueness
- ordering and indexing
- accessing elements
- performance
- use cases

Activity:

Provide your answer for the following question (in true or false manner)

? Set is faster than Array in terms of searching
? Set is collection of unique value
? Array is a sequential collection of data
? Accessing Array element is faster than Set
? Set element are accessed using index value
? Array element are accessed using hash value

1. Arrays allow you to efficiently store multiple values within a single variable.
2. It stores a collection of data.
3. It enables you to tackle complex tasks with ease.

We will do the hands-on in the following topics:

1. Read an input from the user
2. Print the input value to the terminal window.
3. array construction

4. accessing each/all element of an array
5. adding element in an array
6. remove an element from an array (resize)

```
#!/bin/bash
# Program-1: Reading input from the user and displaying in the terminal
echo "Please enter your name:"
read name
echo "Hello, $name!"
```

```
#!/bin/bash
# Program-2: 1D Array declaration, display array elements only

    FRUITS=("banana" "apple" "mango" "orange")
    echo "${FRUITS[@]}"
    for str in ${FRUITS[@]}
    do
    echo $str
    done
```

```
#!/bin/bash
# Program-3: Array Declaration, Display Elements only

    myarray=("a" "b" "c" "d")
    for s in ${myarray[@]}; do
    echo $s
    done

#Note: ';' is needed to separate two different instruction declarations.
```

```
#!/bin/bash
# Program-3.1: Array Declaration, Display element with Indices
myarray=("a" "b" "c" "d")
for i in ${!myarray[@]}
do
  echo "element $i is ${myarray[$i]}"
done
```

```
#!/usr/bin/bash
# Program-4 alternate option: Array Declaration, Display element with Indices

FRUITS=("banana" "apple" "mango" "orange")
# Access all elements in the array
echo "${FRUITS[@]}"
# Access an array element by its index
echo "The item at index 0 is: ${FRUITS[0]}"
echo "The item at index 1 is: ${FRUITS[1]}"
echo "The item at index 2 is: ${FRUITS[2]}"
echo "The item at index 3 is: ${FRUITS[3]}"
```

```
#!/usr/bin/bash
# Program-5: Add an element to the array
#! /usr/bin/bash

FRUITS=("banana" "apple" "mango" "orange")

# Access all elements in the array
echo "${FRUITS[@]}"

# Access an array element by its index

echo "The item at index 0 is: ${FRUITS[0]}"
echo "The item at index 1 is: ${FRUITS[1]}"
echo "The item at index 2 is: ${FRUITS[2]}"
echo "The item at index 3 is: ${FRUITS[3]}"

# Add an element to the array
FRUITS[4]="pawpaw"

# Print array with the added item
echo "Array with added item: ${FRUITS[@]}"
```

```
#!/usr/bin/bash
# Program-6: Add an element to the array

        FRUITS=("banana" "apple" "mango" "orange")

# Access all elements in the array
        echo "${FRUITS[@]}"

# Access an array element by its index

        echo "The item at index 0 is: ${FRUITS[0]}"
        echo "The item at index 1 is: ${FRUITS[1]}"
        echo "The item at index 2 is: ${FRUITS[2]}"
        echo "The item at index 3 is: ${FRUITS[3]}"

# Add an item to the array
        FRUITS[4]="pawpaw"

# Print array with the added item
        echo "Array with added item: ${FRUITS[@]}"

# Replace a value in an array
        FRUITS[0]="avocado"

# Print a new array with the value replaced

        echo "Replaced array: ${FRUITS[@]}"

#! /usr/bin/bash
# Program-7: Remove or delete an element from the array, and resize the array
#! /usr/bin/bash

        FRUITS=("banana" "apple" "mango" "orange")

# Access all elements in the array

        echo "${FRUITS[@]}"

# Access an array element by its index

        echo "The item at index 0 is: ${FRUITS[0]}"
        echo "The item at index 1 is: ${FRUITS[1]}"
        echo "The item at index 2 is: ${FRUITS[2]}"
        echo "The item at index 3 is: ${FRUITS[3]}"
```

```
# Add an item to the array
        FRUITS[4]="pawpaw"
# Print array with the added item
        echo "Array with added item: ${FRUITS[@]}"
# Replace a value in an array
        FRUITS[0]="avocado"
# Print a new array with the value replaced
        echo "Replaced array: ${FRUITS[@]}"
# Remove an item from an array
        unset FRUITS[1]
        unset FRUITS[2]
# Print the array after two items have been deleted
        echo "Array elements after deletion: ${FRUITS[@]}"
```

Output

```
banana apple mango orange
The item at index 0 is: banana
The item at index 1 is: apple
The item at index 2 is: mango
The item at index 3 is: orange
Array with added item: banana apple mango orange pawpaw
Replaced array: avocado apple mango orange pawpaw
Array elements after deletion: avocado orange pawpaw
```

Certain Programs for practices:

```
# Array example

fruits=("apple" "banana" "cherry")
for fruit in "${fruits[@]}"; do
  echo $fruit
```

done

```
#!/bin/bash

# To declare a static Array

arr=("Jayesh" "Shivang" "1" "Vipul" "Nishant" "2")

# To print all elements of the array

echo "All elements of the array:"
echo "${arr[@]}"
echo "${arr[*]}"

# To print the first element

echo "The first element:"
echo "${arr[0]}"


# Associative array example
declare -A colors
colors[apple]="red"
colors[banana]="yellow"
colors[grape]="purple"
unset colors[banana]
echo ${colors[apple]} # red
echo ${colors[grape]} # purple
```

Creating, accessing and modifying array elements

```
my_array=("value1" "value2" "value3")
echo ${my_array[0]}
my_array[1]="new_value"
```