**Special variables in shell scripting**

**In shell scripting, special variables are predefined variables that hold information about the script's execution environment, arguments, and more. These variables provide valuable metadata and control over the script's behavior**

#### **1. $0**

* **Description: The name of the script or the command being executed.**

**Example:  
  
echo "The script name is $0"**

#### **2. $1, $2, ..., $n**

* **Description: Positional parameters that hold the arguments passed to the script.**

**Example:  
Script: simple\_greet.sh  
#!/bin/bash**

**# Directly use positional parameters**

**echo "Hello, $1 $2!"**

**./simple\_greet.sh John Doe**

**Output:**

**Hello, John Doe!**

#### **3. $#**

* **Description: The number of positional parameters (arguments) passed to the script.**

**Example:  
  
echo "Number of arguments: $#"**

#### **4. $@**

* **Description: All the positional parameters as a list, where each parameter is quoted individually.**

**Example:  
  
echo "All arguments: $@"**

#### **5. $\***

* **Description: All the positional parameters as a single string, with each parameter separated by the first character of the IFS variable (usually a space).**

**Example:  
  
echo "All arguments: $\*"**

#### **6. $$**

* **Description: The process ID (PID) of the current script or shell.**

**Example:  
  
echo "The process ID is $$"**

#### **7. $?**

* **Description: The exit status of the last command executed. A value of 0 indicates success, and any other value indicates an error.**

**Example:  
  
ls /some/directory**

**echo "Last command exit status: $?"**

#### **8. $!**

* **Description: The PID of the last background command.**

**Example:  
  
sleep 10 &**

**echo "PID of last background command: $!"**

#### **9. $-**

* **Description: The current options set for the shell.**

**Example:  
  
echo "Current shell options: $-"**

**set -x**

**echo "Debug mode is now enabled."**

**echo "Current shell options: $-"**

**Output**

**+ echo 'Debug mode is now enabled.'**

**Debug mode is now enabled.**

**Current shell options: xB**

**Here, x indicates that debug mode is enabled, and B could be another option depending on your shell configuration.**

**To turn off debug mode:**

**set +x**

**echo "Debug mode is now disabled."**

**echo "Current shell options: $-"**

**Output:-**

**Debug mode is now disabled.**

**Current shell options: B**

**The absence of x in the output indicates that debug mode has been disabled.**

### **Summary**

* **$- provides a snapshot of the current shell options and flags.**
* **Each character in the string represents a specific shell option or flag that is currently active.**
* **This can help you understand the current state of your shell environment, especially useful for debugging and managing shell behavior.**

#### **10. $\_**

* **Description: The last argument of the last command executed.**

**Example:  
  
echo "Hello World"**

**echo "Last argument: $\_"**

### **Examples Using Special Variables**

#### **Example 1: Displaying Script Name and Arguments**

**#!/bin/bash**

**echo "Script name: $0"**

**echo "Number of arguments: $#"**

**echo "First argument: $1"**

**echo "All arguments: $@"**

**Run the script with arguments:**

**./script.sh arg1 arg2 arg3**

**Output:**

**Script name: ./script.sh**

**Number of arguments: 3**

**First argument: arg1**

**All arguments: arg1 arg2 arg3**

#### **Example 2: Using Exit Status**

**#!/bin/bash**

**ls /some/directory**

**if [ $? -eq 0 ]; then**

**echo "Directory listing successful"**

**else**

**echo "Directory listing failed"**

**fi**

#### **Example 3: Background Process**

**#!/bin/bash**

**sleep 30 &**

**echo "Sleep command PID: $!"**

#### **Example 4: Handling Errors**

**#!/bin/bash**

**cp /nonexistent/file /some/destination**

**if [ $? -ne 0 ]; then**

**echo "Error occurred: $?"**

**fi**

### **Summary**

* **$0: Script name.**
* **$1, $2, ..., $n: Positional parameters.**
* **$#: Number of arguments.**
* **$@: All arguments as a list.**
* **$\*: All arguments as a single string.**
* **$$: PID of the current process.**
* **$?: Exit status of the last command.**
* **$!: PID of the last background command.**
* **$-: Current shell options.**
* **$\_: Last argument of the last command.**

**What is array in shell scripting**

**In shell scripting, arrays are used to store multiple values in a single variable. They allow you to handle collections of data efficiently. Different shells handle arrays differently, but I'll cover arrays in bash, which is one of the most commonly used shells.**

### **Defining Arrays**

**In bash, you can define arrays using parentheses () and separate elements with spaces. Here’s how you can define and use arrays in bash:**

#### **Example 1: Define and Access Array Elements**

# Define an array

fruits=("apple" "banana" "cherry")

# Access array elements

echo ${fruits[0]} # Output: apple

echo ${fruits[1]} # Output: banana

echo ${fruits[2]} # Output: cherry

#### **Example 2: Access All Elements**

To print all elements of the array:

# Print all elements

echo ${fruits[@]}

echo ${fruits[\*]}

#### **Example 3: Get the Number or length of Elements**

To get the number of elements in an array:

echo ${#fruits[\*]}

#### **Example 4: Iterating Over an Array**

To loop through all elements of an array:

for fruit in "${fruits[@]}"; do

echo $fruit

done

#### **Example 5: Associative Arrays**

In bash, you can also use associative arrays (key-value pairs):

# Declare an associative array

declare -A colors

# Assign values to keys

colors =([apple]=red [banana]=yellow [cherry]=dark red )

# Access associative array elements

echo ${colors[apple]} # Output: red

echo ${colors[banana]} # Output: yellow

echo ${colors[cherry]} # Output: dark red

#### **Example 6: Append to an Array**

To add an element to the end of an array:

# Append an element

fruits+=("date")

# Print all elements

echo ${fruits[@]}

apple banana cherry date

### **Summary**

* **Defining Arrays**: Use parentheses () to define an array.
* **Accessing Elements**: Use ${array[index]} to access elements.
* **Iterating**: Use loops to process each element.
* **Associative Arrays**: Use declare -A to define associative arrays and access values with keys.
* **Appending**: Use array+=("new\_element") to add elements.