Variables in shell scripting are used to store data that can be used and manipulated throughout the script. They can hold various types of values, such as strings, numbers, or command outputs. Here’s an in-depth guide to understanding variables in shell scripting:

1. Declaring and Assigning Variables

In shell scripting, you declare and assign a value to a variable using the `=` operator. There should be no spaces between the variable name, the `=` operator, and the value.

```
#!/bin/bash

# Declaring and assigning variables
greeting="Hello, World!"
number=42
```

- `greeting` is a variable containing the string "Hello, World!".
- `number` is a variable containing the integer `42`.

2. Accessing Variables

To access the value of a variable, you precede the variable name with a `$` symbol.

```
#!/bin/bash

# Assigning a value to a variable
greeting="Hello, World!"

# Accessing the variable
echo $greeting   # Output: Hello, World!
```

- `echo $greeting` prints the value of the `greeting` variable.

3. Using Double Quotes

When referencing variables, especially those containing spaces or special characters, it's best to enclose the variable in double quotes.

```
#!/bin/bash

# Variable with spaces
name="John Doe"

# Correct usage with double quotes
echo "Hello, $name!"   # Output: Hello, John Doe!

# Incorrect usage without quotes
echo Hello, $name!     # Output: Hello, John Doe!
```

4. Variables with Command Substitution

You can assign the output of a command to a variable using command substitution, which is done using backticks `` `command` `` or `$(command)`.

```
#!/bin/bash

# Assigning the output of a command to a variable
current_date=$(date)

echo "Today's date is: $current_date"
```

- `$(date)` captures the output of the `date` command and assigns it to the `current_date` variable.

5. Environment Variables

Environment variables are special variables that are available system-wide. Common examples include `$HOME`, `$PATH`, and `$USER`.

```
#!/bin/bash

# Accessing environment variables
echo "Home directory: $HOME"
echo "Current user: $USER"
```

- `$HOME` holds the path to the user's home directory.
- `$USER` holds the name of the currently logged-in user.

6. Exporting Variables

To make a variable available to child processes or other scripts, you need to export it.

```
#!/bin/bash

# Define a variable
greeting="Hello from parent script!"

# Export the variable
export greeting

# Call another script
./child_script.sh
```

In `child_script.sh`:

```
#!/bin/bash

# Access the exported variable
echo $greeting   # Output: Hello from parent script!
```

- The `export` command makes the `greeting` variable available to `child_script.sh`.

7. Arithmetic Operations

Variables can be used to perform arithmetic operations in shell scripts.

```
#!/bin/bash

# Define variables
num1=10
num2=5
```

```
# Perform arithmetic operations
sum=$((num1 + num2))
difference=$((num1 - num2))
product=$((num1 * num2))
quotient=$((num1 / num2))

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

- `$((expression))` is used for arithmetic operations.

8. Read-Only Variables

You can make a variable read-only, meaning its value cannot be changed after it's set.

```
#!/bin/bash

# Define a read-only variable
readonly pi=3.14

# Attempt to modify the variable (will result in an error)
pi=3.14159  # Error: pi: readonly variable
```

9. Unset Variables

To unset a variable (i.e., remove its value), you can use the `unset` command.

```
#!/bin/bash

# Define and unset a variable
name="John Doe"
unset name

echo $name  # Output will be empty
```

- After using `unset`, the `name` variable no longer has a value.

10. Example: Simple Script Using Variables

Here's a simple script that demonstrates the use of variables in various ways:

```
#!/bin/bash

# Define variables
first_name="John"
last_name="Doe"
year_of_birth=1990
current_year=$(date +%Y)

# Calculate age
age=$((current_year - year_of_birth))

# Display information
echo "Name: $first_name $last_name"
echo "Year of Birth: $year_of_birth"
echo "Current Year: $current_year"
echo "Age: $age"
```

Explanation:

- `first_name` and `last_name` are string variables.
- `year_of_birth` and `current_year` are integer variables.
- `current_year` is assigned using command substitution with `$(date +%Y)` to get the current year.
- The script calculates the age by subtracting `year_of_birth` from `current_year`.
- Finally, it prints the user's name, year of birth, current year, and age.

Summary

- **Declaring**: Variables are declared and assigned using `=` without spaces.
- **Accessing**: Use `$` before the variable name to access its value.
- **Quoting**: Double quotes help prevent word splitting and preserve special characters.
- **Command Substitution**: Use `$(command)` to capture command output into a variable.
- **Environment Variables**: Special system-wide variables like `$HOME` and `$USER`.

- **Exporting**: Use `export` to make a variable accessible to child scripts.
- **Arithmetic**: Perform arithmetic with `$((expression))`.
- **Read-Only**: `readonly` prevents further modification of a variable.
- **Unset**: `unset` removes a variable's value.

Variables are fundamental in shell scripting, enabling dynamic and flexible script behavior.

4o