SHELLSCRIPT:

What is Shell?

A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human-readable commands from users and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

Shell is broadly classified into two categories –

Command Line Shell

Graphical shell

Shell Scripting

Usually, shells are interactive, which means they accept commands as input from users and execute them. However, sometimes we want to execute a bunch of commands routinely, so we have to type in all commands each time in the terminal.

As a shell can also take commands as input from file, we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called Shell Scripts or Shell Programs. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with `.sh` file extension e.g., myscript.sh.

A shell script has syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. It would be very easy to get started with it.

A shell script comprises the following elements –

Shell Keywords – if, else, break etc.

Shell commands – cd, ls, echo, pwd, touch etc.

Functions Control flow – if..then..else, case and shell loops etc.

Why do we need shell scripts?

There are many reasons to write shell scripts:

To avoid repetitive work and automation

System admins use shell scripting for routine backups.

System monitoring

Adding new functionality to the shell etc.

Here's a simple example of a shell script that prints "Hello, World!":

#!/bin/bash

This is a comment

echo "Hello, World!"

VARIABLES:

In shell scripting, variables are used to store data that can be referenced and manipulated within the script. They are similar to variables in other programming languages but have some unique characteristics in shell scripting.

Basic Rules for Shell Script Variables:

Naming: Variable names in shell scripts can consist of letters (a-z, A-Z), numbers (0-9), and underscores (_), but they cannot start with a number.

Assignment: Variables are assigned values using the syntax variable_name=value. There should be no spaces around the equal sign.

Accessing: To access the value of a variable, prefix its name with a dollar sign ($). For example, $variable_name.

Quoting: It's a good practice to quote variable references to avoid word splitting and globbing. Double quotes (") are often used, but single quotes (') can also be used when you want to prevent variable expansion.

Scope: By default, variables are global within the script. However, you can use local to declare variables with a local scope within functions.

```
# Variable assignment
variable_name="value"
number=42
name="John Doe"

# Accessing variables
echo $variable_name
echo "The answer is: $number"
echo "Hello, $name!"

# Quoting variables
message="Hello, world!"
echo "$message"    # This will print: Hello, world!
echo '$message'    # This will print: $message
```

Special Variables:

Shell scripts also have several special variables that have predefined meanings:

$0: Name of the script.

$1, $2, ...: Positional parameters passed to the script.

$@: All positional parameters.

$#: Number of positional parameters.

$?: Exit status of the last command.

$$: Process ID of the script itself.

$!: Process ID of the last background command.

```
echo "The script name is: $0"
echo "The first argument is: $1"
echo "All arguments are: $@"
echo "Number of arguments: $#"
```

Command line arguments:

Command-line arguments are parameters passed to a shell script when it is executed. They allow users to provide input to the script dynamically, making scripts more versatile and reusable. Here's how you can work with command-line arguments in a shell script:

Accessing Command-Line Arguments:

In a shell script, command-line arguments are accessed using positional parameters. Each argument passed to the script is assigned a positional parameter, starting from $1, $2, and so on.

For example, consider a script named example.sh:

```
#!/bin/bash

echo "The first argument is: $1"
echo "The second argument is: $2"
echo "All arguments are: $@"
echo "Number of arguments: $#"
```

FUNCTIONS:

Functions in shell scripting allow you to group code into reusable blocks, providing modularity to your scripts. They are defined using the function keyword or simply by declaring the function name followed by parentheses (). Here's the basic example:

```
add() {
    local sum=$(( $1 + $2 ))
    echo $sum
}

result=$(add 10 20)
echo "The sum is: $result"
```

Some Disadvantages of shell scripts

Prone to costly errors, a single mistake can change the command which might be harmful.

Slow execution speed

Design flaws within the language syntax or implementation

Not well suited for large and complex task

Provide minimal data structure unlike other scripting languages. etc.