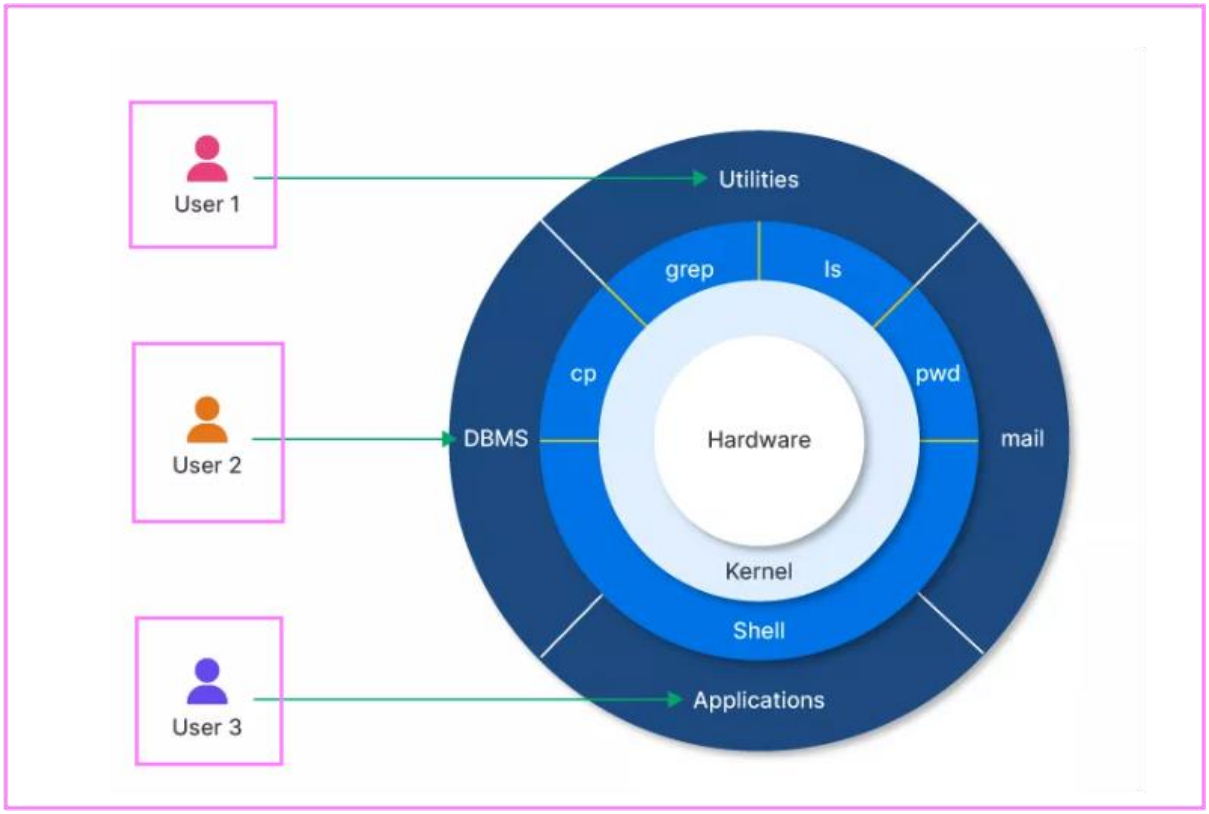
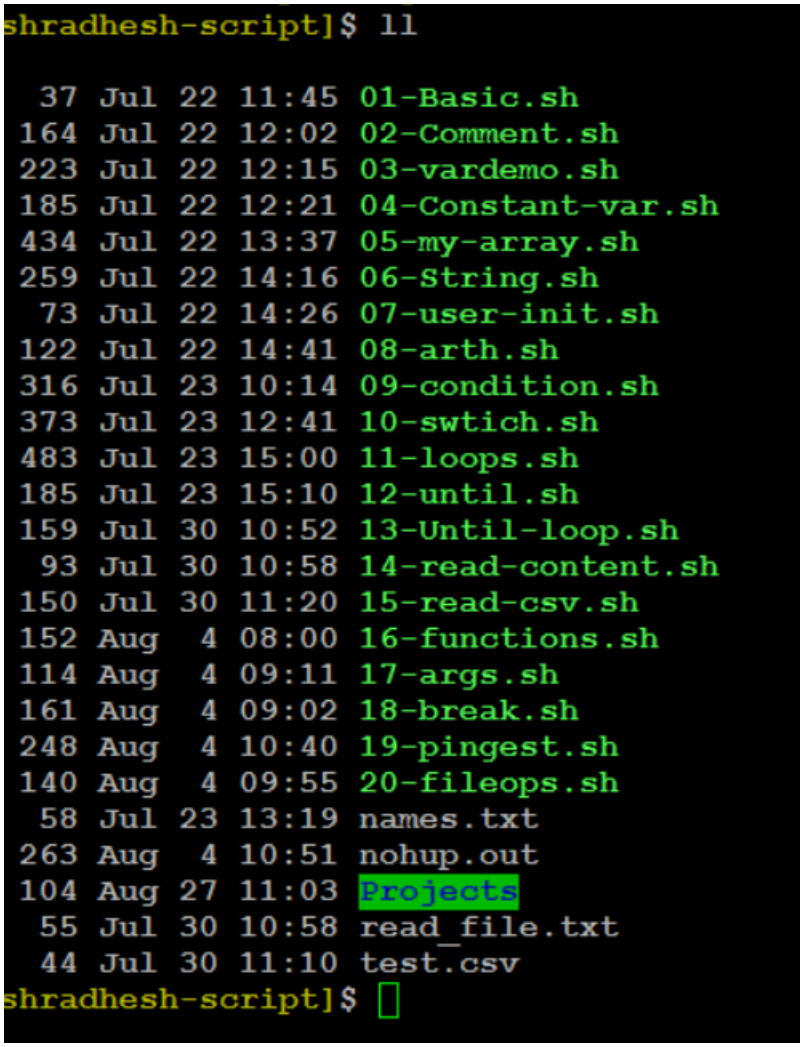
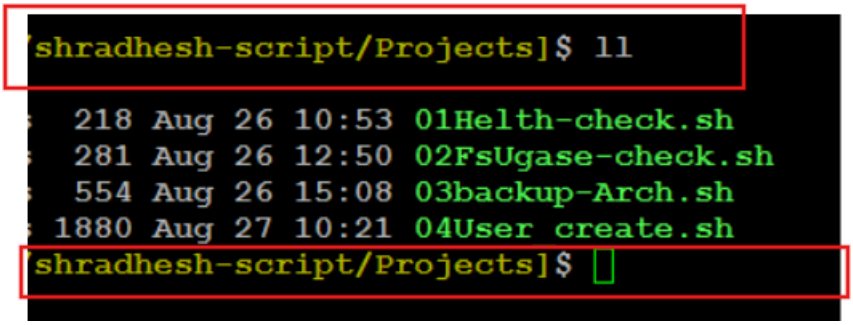
Shell Scripting Guide



"All the essential scripts have been created in a simple and easy-to-understand manner, making them suitable even for beginners "



"Some Realtime -Project "



Shell scripting is the process of writing a series of commands in a text file (called a script) to automate tasks in a Unix/Linux environment.

Key Points:

- A shell script is written for the **shell** (command-line interpreter) like **bash**, **sh**, **zsh**, etc.
- It allows you to **automate repetitive tasks** such as backups, monitoring, deployments, and system administration.
- The script typically has a **.sh** extension and is executed line by line by the shell.
- It supports **variables, loops, conditions, and functions** like a programming language.

1. Comments

Comments are ignored by the shell during execution.
They are used to explain the purpose of code.

- **Single line comment** starts with #.
- Multi-line comments can be simulated using special constructs.

Ref: Example: 02-Comment : https://github.com/Shradhesh1/shell-scripts/tree/Projects

2. Variables

Variables in shell scripting are used to store data such as strings, numbers, or file names. They do not require explicit data type declarations. Variables can be:

- **Local Variables**: Exist only within the current shell or function.
- **Environment Variables**: Accessible to the current shell and its child processes; created using export.
- **Read-only Variables**: Variables whose values cannot be modified after being assigned.

Ref: Example: 03-vardemo https://github.com/Shradhesh1/shell-scripts/tree/Projects

3. Constants

A constant is a value that remains unchanged throughout the execution of a script.

- **Purpose**: Used for fixed configurations like file paths, environment settings, or constant numeric values.
- **Key Point**: Although shell does not have a dedicated constant keyword, you can create one by assigning a value to a variable and avoiding reassignment.

Ref: Example: 04 constant: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

4. Arrays

Arrays allow storing multiple values under a single variable name, accessed by index.

- **Types:**
  - **Indexed Arrays:** Store ordered elements referenced by numeric indices.
  - **Associative Arrays (Bash):** Store key-value pairs.
- **Usage:** Useful for handling lists like filenames, options, or numeric datasets.

Ref: Example: 05 May-Array: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

5. Strings

Strings are sequences of characters used for storing text-based data.

- **Operations:**
  - Concatenation of multiple strings.
  - Substring extraction and pattern matching.
  - Comparison of string values.
- **Application:** Essential in user prompts, messages, file names, and command outputs.

Ref: Example: 06-String: : ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

---

6. User Input

User input enables dynamic interaction with a script.

- **How It Works**: Prompts the user to provide data during runtime and stores it in a variable.
- **Use Cases**: Interactive scripts, configuration input, or command-driven execution.

Ref: Example: 07-User-input: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

7. Arithmetic Operations

Arithmetic operations perform calculations within scripts.

- **Supported Operations**: Addition, subtraction, multiplication, division, and modulus.
- **Purpose**: Commonly used for counters, resource checks, and mathematical evaluations.

Ref: Example: 08-arth: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

8. Conditions (Decision Making)

Conditions control script execution based on logical evaluation.

- **Structures:**
  - if, else, elif for branching decisions.
  - Comparisons for numbers, strings, and files.
- **Application:** Validating inputs, checking file existence, and executing conditional tasks.

Ref: Example: 09-condition: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

9. Switch Case

The switch-case (case statement) handles multiple predefined options efficiently.

- **How It Works**: Matches a variable against multiple patterns and executes the matching block.
- **Use Cases:** Menus, option handling, and structured multi-condition execution.

Ref: Example: 10-switch: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

---

10. Loops

Loops repeatedly execute a set of commands until a condition is met.

- **Types:**
  - **For Loop:** Iterates over lists or ranges.
  - **While Loop:** Repeats while a condition is true.
  - **Until Loop:** Repeats until a condition becomes true.
- **Application:** File processing, repeated automation tasks, and iterative logic.

Ref: Example: 11-loops.sh: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

11. Functions

Functions are reusable blocks of code for modular scripting.

- **Advantages:**
  - Avoid code duplication.
  - Simplify maintenance and readability.
- **Features:** Can accept arguments and return values.

Ref: Example: 16-function.sh : ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

12. Command Line Arguments

Command line arguments allow passing external inputs to a script at execution time.

- **Purpose:** Makes scripts dynamic and adaptable.
- **Common Uses:** Automating repetitive tasks with different inputs, batch processing, and configurable scripts.

Ref: Example: 17-args.sh: ***https://github.com/Shradhesh1/shell-scripts/tree/Projects***

---

13. Break

The break statement terminates a loop prematurely.

- **Use Case:** Exit loops early when a desired condition is satisfied, such as finding a specific file or reaching a stop condition.

Ref: Example: https://github.com/Shradhesh1/shell-scripts/blob/main/18-break.sh

---

14. Continue

The continue statement skips the current iteration of a loop and moves to the next.

- **Use Case:** Ignoring certain values or conditions during loop execution while continuing the process.

Ref: Example: https://github.com/Shradhesh1/shell-scripts/blob/main/18-break.sh

---

15. File Operations

File operations involve creating, reading, modifying, and deleting files.

- **Common Operations:**
  - Reading data for processing.
  - Writing or appending logs.
  - Deleting or updating configuration files.
- **Importance:** Core part of automation, backup scripts, and data handling tasks.

Ref: Example: ***https://github.com/Shradhesh1/shell-scripts/blob/main/20-fileops.sh***

Conclusion

Understanding these concepts is essential for building effective shell scripts. By mastering constants, arrays, strings, input handling, control structures, and file operations, you can create robust automation solutions for Linux environments.