Ultimate Shell Scripting Interview Preparation Guide

Target Role: DevOps / Cloud Engineer (Fresher)

Part 1: Basic Level (Foundations)

1. What is a shell script ? Explain its purpose and how it differs from a regular program.

Answer: A shell script is a text file containing a sequence of commands that the shell executes line-by-line. Unlike regular programs (C++, Java) that are compiled into machine code, shell scripts are interpreted. We use them to automate repetitive tasks, manage system configurations, and simplify complex workflows effectively.

2. What are the different types of shells in Unix/Linux ?

Answer: The shell is the interface between the user and the kernel. Common types include:

- **Bash (Bourne Again Shell):** The standard default for Linux.
- **sh (Bourne Shell):** The original Unix shell.
- **zsh (Z Shell):** Popular for developers due to advanced customization.
- **ksh (Korn Shell):** Often used in enterprise Unix systems.

3. How do you execute a shell script ?

Answer: There are two steps:

1. **Grant Permission:** `chmod +x scriptname.sh` (Makes it executable).
2. **Run:** `./scriptname.sh` (if in the current directory) or `bash scriptname.sh`.

4. What are the basic shell scripting syntax elements ?

Answer:

- **Shebang (`#!/bin/bash`):** Tells the system which interpreter to use.
- **Variables:** `NAME="John"` (No spaces around `=`).

- **Comments:** Lines starting with `#`.
- **Control Flow:** `if`, `for`, `while`, `case`.
- **Redirection:** `>` (Output), `<` (Input).

5. How do you write a simple shell script to print "Hello, World!" ?

Answer:

```
Bash
#!/bin/bash
echo "Hello, World!"
```

6. Explain the concept of environment variables.

Answer: These are system-wide variables that pass settings to processes.

- **Examples:** `$PATH` (executable search paths), `$USER`, `$HOME`.
- **Usage:** We use `export VAR="value"` to make a variable available to child processes.

7. How do you perform arithmetic operations in Bash ?

Answer:

- **Integers:** Use double parentheses: `(( sum = 5 + 5 ))`.
- **Decimals:** Bash doesn't support decimals natively, so we use `bc`: `echo "5.5 + 2.5" | bc`.

8. What is the "Shebang" line ?

Answer: It is the absolute first line of a script (e.g., `#!/bin/bash`). It instructs the operating system's program loader which interpreter program to use to run the script.

9. What is an array in shell scripting? Answer: An array is a variable that can store multiple values under a single name. Each value is accessed using an index starting from 0. Arrays are useful for storing lists like names, numbers, or files.

10. How do you declare an array in Bash ?

Answer: I declare an array using parentheses `()`.

Example:

Bash

fruits=("apple" "banana" "orange")

11. How do you access elements from an array ?

Answer: I access array elements using index numbers inside curly braces.

Example:

Bash

echo ${fruits[0]}

12. Can Bash arrays store different data types ?

Answer: Yes. Bash arrays can store strings and numbers together, but internally all values are treated as strings.

Example:

Bash

data=("prod" 10 true)

Part 2: Intermediate Level (Logic, Data & Files)

13. What are command-line arguments in shell scripts ?

Answer: They allow us to pass input to the script at runtime.

- **$1, $2:** The first and second arguments.
- **$0:** The script name itself.
- **$#:** The total number of arguments.

14. What is the difference between $@ and $* ?

Answer: This distinction matters when using loops:

- **"$@"** treats arguments as separate strings ("arg1" "arg2"). This is the standard best practice.
- **"$*"** treats all arguments as a single string ("arg1 arg2").

Create a simple script (nano args.sh)

```
#!/bin/bash

echo 'Using "$@"'

for arg in "$@"; do

  echo "Argument: $arg"

done



echo

echo 'Using "$*"'

for arg in "$*"; do

  echo "Argument: $arg"

done
```

Run the script - ./args.sh apple banana "red fruit"

Output -

Using "$@"

```
Using "$@"

Argument: apple
```

Argument: banana

Argument: red fruit

Using "$*"

Copy code

Using "$*"

Argument: apple banana red fruit

15. How do you read user input from the keyboard ?

Answer: We use the `read` command.

- **Example:** `read -p "Enter your name: " username` stores the input in `$username`.

16. What are shell functions? How do you define them ?

Answer: Functions are reusable blocks of code. They make scripts modular and easier to debug. **Syntax:**

```
Bash
my_function() {
   echo "Task executed"
}
my_function  # Calling the function
```

17. How do you write a shell script to check if a file exists and is readable ?

Answer: I use the `test` command or brackets `[ ]`.

- `[ -f filename ]`: Checks if the file exists.
- `[ -r filename ]`: Checks if the file has read permission.

18. How do you write a shell script to sort a file numerically ?

Answer: I use the `sort` command with the `-n` flag.

- **Command:** `sort -n filename.txt` (Ensures 10 comes after 2, not before).

19. What is the difference between single quotes, double quotes, and backticks ?

Answer:

- **Single (' '):** Strict quoting. Everything inside single quotes is treated as plain text. Variables are **not replaced** by their values.
- **Double (" "):** Weak quoting. Variables inside double quotes are **replaced with their actual values**.
- **Backticks ( ):** They are used for **command substitution**. The command inside is executed and its output is used.

  Example:

  ```
   #!/bin/bash

  # Assume this value

  USER_NAME="vaibhav"

  echo "Using single quotes:"

  echo '$USER_NAME'

  —-------------------------

  echo "Using double quotes:"

  echo "$USER_NAME"

  —-------------------------

  echo "Using backticks (command substitution):"

  echo `date`
  ```

  Output :

Using single quotes:

$USER_NAME

—--------------------------

Using double quotes:

vaibhav

—--------------------------

Using backticks (command substitution):

Sun Dec 14 08:50:00 IST 2025

20. How do you find files based on name, size, or age ?

Answer: I use the `find` command.

- **Example:** `find /var/log -name "*.log" -mtime +7` (Finds logs older than 7 days).

21. How do you debug a shell script ?

Answer:

Debugging a shell script means checking which commands are executed and what values variables take during execution.

Method 1: Using execution trace (-x)

Command:
bash -x script.sh

This shows each command after variable expansion and before execution.

Example script:

```
 name="Vaibhav"
 echo "Hello $name"
```

Debug output:

- name=Vaibhav
- echo 'Hello Vaibhav'

Actual output:
Hello Vaibhav

Method 2: Debugging inside the script

Commands:
set -x
set +x

set -x turns debugging ON
set +x turns debugging OFF

Example:

```
#!/bin/bash
set -x
name="Vaibhav"
echo "Hello $name"
set +x
```

Important Note:
The debug output shows executed commands, not normal program output. It is different from echo output.

22. How do you check if the previous command was successful ? (Exit Status)

Answer: I check the `$?` variable immediately after the command.

- **0** = Success.
- **Non-zero (1-255)** = Failure.

23. How do you print all elements of an array ?

Answer: I use `@` or `*` inside curly braces.

Example:

Bash

```
echo "${fruits[@]}"
```

24. How do you find the length of an array ?

Answer: I use `#` with `@`.

Example:

Bash

```
echo ${#fruits[@]}
```

25. How do you loop through an array in Bash ?

Answer: I use a `for` loop.

Example:

Bash

```
for fruit in "${fruits[@]}"; do
  echo "Fruit: $fruit"
done
```

26. How do you add a new element to an array ?

Answer: I append a value using `+=`.

Example:

Bash

fruits+=("grape")

27. How do you remove an element from an array ?

Answer: I use the `unset` command with the index number. **Example:**

Bash

```
unset fruits[1]
```

28. What is the difference between `"${array[@]}"` and `"${array[*]}"` ?

Answer:

- `"${array[@]}"` treats each element as a separate value (Recommended).
- `"${array[*]}"` treats all elements as a single string.

29. How do you define and iterate over an Array in Bash ?

Answer: Arrays are used to handle lists of servers or resources.

- **Define:** `Servers=("web01" "db01" "app01")`

Iterate:
Bash

```
for server in "${Servers[@]}"; do
  ping -c 1 $server
done
```

Part 3: Advanced Level (Automation, DevOps & Networking)

30. How do you write a shell script to automate a backup process ?

Answer: A robust backup script follows these steps:

1. **Define Variables:** Source folder, destination, and timestamp format (`date +%F`).
2. **Compress:** Use `tar -czf` to create a `.tar.gz` archive.
3. **Transfer:** Use `scp` or `rsync` to send it to a remote backup server.
4. **Log:** Append the result ("Success" or "Failure") to a log file.
5. **Schedule:** Add it to `crontab` to run daily.

32. Explain the concept of pipelines (|).

Answer: Pipelines allow us to chain commands together. The Standard Output (STDOUT) of the first command becomes the Standard Input (STDIN) of the next.

- **Example:** `cat access.log | grep "404" | wc -l` (Counts 404 errors).

32. How do you parse a log file and extract specific information ?

Answer: I use the standard text processing tools:

- **grep:** To filter lines containing specific keywords (e.g., "ERROR").
- **awk:** To extract specific fields/columns (e.g., just the IP address).
- **sed:** To replace or modify text streams.

33. What are regular expressions? How are they used with grep/sed ?

Answer: Regex is a pattern-matching language.

- `^`: Matches start of a line.
- `$`: Matches end of a line.
- `[0-9]`: Matches any digit.
- **Example:** `grep "^Error" file.log` matches lines starting with "Error".

34. How do you write a shell script to send email notifications ?

Answer: I use the `mail` or `sendmail` command.

- **Command:** `echo "Job Failed" | mail -s "Alert" admin@example.com`.
- **Modern Context:** In DevOps, I often use `curl` to send a JSON payload to a Slack webhook instead.

35. What are shell traps? How are they used ?

Answer: `trap` is used to catch signals (like Ctrl+C or SIGINT). It ensures that if a script is interrupted, we can run a "cleanup" function to delete temporary files before the script exits, preventing clutter.

36. Explain the difference between > and >>.

Answer:

- **> is a Destroyer:** It deletes everything in the file and writes new text.
- **>> is a Keeper:** It keeps the old text and adds new text to the bottom.

1. Example of > (Overwrite)

Think of this as **"New Start"**. It cleans the slate.

# First, we write "Apple" into a file

echo "Apple" > fruits.txt

# Now, we use > again with "Banana"

echo "Banana" > fruits.txt

# RESULT: The file only contains:

Banana

2. Example of >> (Append)

Think of this as **"Add More"**. It adds to the list.

# First, we write "Apple" into a file

echo "Apple" > fruits.txt

# Now, we use >> with "Banana"

echo "Banana" >> fruits.txt

# RESULT: The file contains:

Apple

Banana

37. How do you run a script in the background ?

Answer: Add an ampersand `&` at the end (`./script.sh &`). To keep it running even if I close the terminal, I use `nohup ./script.sh &`.

38. How do you check memory and disk usage ?

Answer:

- **Memory:** `free -m` (shows RAM in MB).
- **Disk:** `df -h` (shows filesystem space) and `du -sh` (shows folder size).

39. What is `/dev/null` ?

Answer: It is a special file that discards all data written to it (the "black hole"). I use it to silence unwanted output: `command > /dev/null 2>&1`.

40. How do you handle file permissions ?

Answer: I use `chmod`.

- `chmod 755 file.sh`: Owner gets full access (7), Group/Others get Read+Execute (5).
- `chown user:group file.sh`: Changes ownership.

41. How do you schedule a script to run automatically ?

Answer: I use the `crontab -e` command.

- **Format:** `* * * * * /path/to/script.sh` (Minute, Hour, Day, Month, Weekday).

42. How are arrays used in real DevOps automation ?

Answer: Arrays are used to store server names, IP addresses, and environments so the same command can run on multiple systems efficiently using loops.

43. Write a shell script using an array to ping multiple servers.

Answer:

```
Bash

#!/bin/bash

servers=("google.com" "github.com")

for server in "${servers[@]}"; do

  ping -c 1 $server

done
```

44. Are arrays supported in all shells ?

Answer: No. Arrays are fully supported in Bash, Zsh, and Ksh, but **not** in basic `sh`.

Part 4: Modern DevOps & Cloud Scripting Essentials (Added)

45. What is an associative array in Bash ?

Answer: An associative array uses keys (strings) instead of numeric indexes. It is useful for structured data like Key-Value pairs.

Example:

```
Bash
declare -A user
user[name]="Vaibhav"
```

```
user[role]="DevOps"
```

46. Where are associative arrays used in DevOps ?

Answer: They are used to map relationships, such as:

- Server → IP Address
- Service → Port Number
- Environment → Configuration File

47. Write a script using an associative array.

Answer:

```
Bash
#!/bin/bash
declare -A ports
ports[nginx]=80
ports[ssh]=22

for service in "${!ports[@]}"; do
  echo "$service runs on port ${ports[$service]}"
done
```

48. How do you parse JSON data in a shell script ?

Answer: Since standard tools like `cut` struggle with JSON, I use **`jq`**, a command-line JSON processor. This is essential for handling output from AWS CLI or Kubernetes.

- **Example:** `cat aws_instance.json | jq '.Instances[0].InstanceId'`

49. What is the difference between a Hard Link and a Soft (Symbolic) Link ?

Answer:

- **Soft Link (`ln -s`):** Acts like a shortcut. If the original file is deleted, the link breaks. It can span across different file systems.

- **Hard Link (`ln`):** Points to the exact same physical data (inode) on the disk. If the original file is deleted, the data remains accessible via the hard link.

50. How do you execute a script on a remote server securely ?

Answer: I use **SSH (Secure Shell)**.

- **Command:** `ssh user@remote-server 'bash -s' < local_script.sh`
- **Automation:** I configure SSH Key-based authentication (public/private keys) so the script runs without pausing for a password.

51. Explain the difference between `exec` and `source` commands.

Answer:

- **`source script.sh`:** Runs the code in the *current* shell. Variables defined inside remain available after the script finishes.
- **`exec command`:** Replaces the *current* shell process with the new command. The shell does not return; the process is consumed by the new command.

52. What is a "Zombie Process" and how do you find it ?

Answer: A Zombie is a process that has completed execution but still has an entry in the process table.

- **Identify:** Run `top` or `ps aux` and look for `Z` in the status column.
- **Fix:** You cannot kill a zombie directly; you must kill its **parent process** so the system (`init`) can clean it up.