SHELL Scripting Interview Questions and Answers

1. What is a shell script?

Question: What is a shell script?

Answer: A shell script is a text file that contains a sequence of commands for a UNIX-based operating system's shell to execute. Shell scripts are used to automate repetitive tasks, manage system operations, and simplify complex commands. They can contain standard UNIX commands, conditional statements, loops, and functions.

2. How do you create and execute a shell script?

Question: How do you create and execute a shell script?

Answer:

1. **Create a Shell Script:**
   - Open a text editor and write your script.
   - Save the file with a `.sh` extension (e.g., `myscript.sh`).

   Example script:

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

2. **Make the Script Executable:**

   ```
   chmod +x myscript.sh
   ```

3. **Execute the Script:**

   ```
   ./myscript.sh
   ```

3. What is the significance of `#!/bin/bash` at the beginning of a script?

Question: What is the significance of `#!/bin/bash` at the beginning of a script?

Answer: The `#!/bin/bash` line at the beginning of a shell script is called a shebang or hashbang. It specifies the path to the interpreter that should be used to execute the script. In this case, it indicates that the script should be run using the Bash shell. It ensures that the script runs with the correct interpreter, regardless of the user's default shell.

4. How do you define and use variables in a shell script?

Question: How do you define and use variables in a shell script?

Answer:

- **Define a Variable:**

  ```
  my_variable="Hello"
  ```

- **Use a Variable:**

  ```
  echo $my_variable
  ```

- Example script:

  ```
  #!/bin/bash
  greeting="Hello, World!"
  echo $greeting
  ```

5. How do you write comments in a shell script?

Question: How do you write comments in a shell script?

Answer: Comments in a shell script are written using the `#` symbol. Anything following `#` on a line is treated as a comment and ignored by the shell.

Example:

```
#!/bin/bash
# This is a comment
echo "Hello, World!" # This is also a comment
```

6. What are positional parameters in shell scripting?

Question: What are positional parameters in shell scripting?

Answer: Positional parameters are variables that hold the arguments passed to a script or function. They are referenced using `$1`, `$2`, `$3`, etc., where `$1` is the first argument, `$2` is the second, and so on. `$0` refers to the script's name.

Example script:

```
#!/bin/bash
echo "First argument: $1"
echo "Second argument: $2"
```

If the script is executed with `./myscript.sh arg1 arg2`, it will output:

```
First argument: arg1
Second argument: arg2
```

7. How do you use conditional statements in a shell script?

Question: How do you use conditional statements in a shell script?

Answer: Conditional statements in shell scripts are used to perform different actions based on conditions. The `if` statement is commonly used for this purpose.

Example:

```
#!/bin/bash
number=5
if [ $number -gt 3 ]; then
  echo "The number is greater than 3"
else
  echo "The number is not greater than 3"
fi
```

8. How do you loop through a list of items in a shell script?

Question: How do you loop through a list of items in a shell script?

Answer: The `for` loop is used to iterate through a list of items in a shell script.

Example:

```
#!/bin/bash
for item in apple banana cherry; then
  echo "Fruit: $item"
done
```

9. How do you read input from the user in a shell script?

Question: How do you read input from the user in a shell script?

Answer: You can use the `read` command to read input from the user.

Example:

```
#!/bin/bash
echo "Enter your name:"
read name
echo "Hello, $name!"
```

10. How do you debug a shell script?

Question: How do you debug a shell script?

Answer: You can debug a shell script using the `-x` option with `bash` to execute the script in debug mode.

Example:

```
bash -x myscript.sh
```

This will print each command and its arguments as they are executed, helping you to identify where the script is failing or producing unexpected results.

1. What is the difference between `[` and `[[` in shell scripting?

Question: What is the difference between `[` and `[[` in shell scripting?

Answer: `[` is a synonym for the `test` command and is POSIX compliant. `[[` is a keyword and provides more features, such as:

- Enhanced pattern matching.
- Logical operators (`&&`, `||`) without the need for `-a` or `-o`.
- Safer handling of complex expressions (e.g., regex).

Example:

```
#!/bin/bash
# Using [
if [ "$a" == "$b" ]; then
  echo "Equal"
fi

# Using [[
if [[ "$a" == "$b" ]]; then
  echo "Equal"
fi
```

2. How do you handle errors in a shell script?

Question: How do you handle errors in a shell script?

Answer: Use exit codes, `trap` command, and error checking after each command.

Example:

```
#!/bin/bash

# Exit on any error
set -e

# Function to handle errors
error_handler() {
  echo "Error occurred in script at line: $1"
  exit 1
}

# Trap errors
trap 'error_handler $LINENO' ERR

# Commands
```

```
command1
command2
```

3. How can you pass an array to a function in shell scripting?

Question: How can you pass an array to a function in shell scripting?

Answer: Use `local` and `eval` to handle array parameters.

Example:

```
#!/bin/bash

# Function to print an array
print_array() {
  eval "local arr=(\"\${$1[@]}\")"
  for element in "${arr[@]}"; do
    echo $element
  done
}

# Main script
my_array=("one" "two" "three")
print_array my_array[@]
```

4. How do you check if a process is running in shell scripting?

Question: How do you check if a process is running in shell scripting?

Answer: Use `pgrep` or `ps` to check for a running process.

Example:

```
#!/bin/bash

process_name="my_process"

if pgrep "$process_name" > /dev/null; then
  echo "$process_name is running"
else
  echo "$process_name is not running"
fi
```

5. How do you handle command substitution in shell scripting?

Question: How do you handle command substitution in shell scripting?

Answer: Use backticks (``) or `$()` for command substitution.

Example:

```
#!/bin/bash

# Using backticks
current_date=`date`

# Using $()
current_date=$(date)

echo "Current date and time: $current_date"
```

6. How do you create and use a Here Document in a shell script?

Question: How do you create and use a Here Document in a shell script?

Answer: A Here Document allows you to pass a block of text to a command.

Example:

```
#!/bin/bash

cat <<EOF
This is a Here Document
It allows multi-line strings
EOF
```

7. Explain the usage of `trap` command with an example.

Question: Explain the usage of `trap` command with an example.

Answer: The `trap` command is used to specify commands to be executed when the shell receives a signal.

Example:

```
#!/bin/bash

# Function to execute on exit
cleanup() {
  echo "Cleaning up..."
  rm -f /tmp/mytempfile
}

# Trap EXIT signal
trap cleanup EXIT

# Commands
echo "Script is running"
touch /tmp/mytempfile
```

8. How do you implement logging in a shell script?

Question: How do you implement logging in a shell script?

Answer: Redirect output to a log file and use `exec` to direct stdout and stderr.

Example:

```
#!/bin/bash

log_file="script.log"

# Redirect stdout and stderr
exec > >(tee -a $log_file) 2>&1

# Commands
echo "This is a log entry"
```

9. How do you perform arithmetic operations in a shell script?

Question: How do you perform arithmetic operations in a shell script?

Answer: Use `$(( ... ))` for arithmetic operations.

Example:

```
#!/bin/bash

a=10
b=5

sum=$((a + b))
difference=$((a - b))
product=$((a * b))
quotient=$((a / b))

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

10. How do you schedule a shell script to run at a specific time?

Question: How do you schedule a shell script to run at a specific time?

Answer: Use `cron` for scheduling scripts.

1. **Edit the crontab:**

   ```
   crontab -e
   ```

2. **Add a cron job:**

```
0 2 * * * /path/to/script.sh
```

This schedules `script.sh` to run every day at 2:00 AM.

11. What is the use of `set -e`, `set -u`, `set -o pipefail`?

Question: What is the use of `set -e`, `set -u`, `set -o pipefail`?

Answer:

- `set -e`: Exit immediately if a command exits with a non-zero status.
- `set -u`: Treat unset variables as an error and exit immediately.
- `set -o pipefail`: Return the exit status of the last command in the pipeline that failed.

Example:

```
#!/bin/bash
set -euo pipefail

command1
command2
```

12. How do you manage background processes in shell scripting?

Question: How do you manage background processes in shell scripting?

Answer: Use `&` to run a command in the background and `wait` to wait for background processes to finish.

Example:

```
#!/bin/bash

# Run in background
command1 &
pid1=$!

command2 &
pid2=$!

# Wait for both processes
wait $pid1
wait $pid2
```

13. How do you handle functions in a shell script?

Question: How do you handle functions in a shell script?

Answer: Define and call functions using the `function` keyword or directly with the function name.

Example:

```
#!/bin/bash

# Function definition
my_function() {
  echo "Hello from my_function"
}

# Call the function
my_function
```

14. How do you parse command-line arguments in a shell script?

Question: How do you parse command-line arguments in a shell script?

Answer: Use `getopts` to parse command-line options.

Example:

```
#!/bin/bash

while getopts "a:b:" opt; do
  case $opt in
    a) param_a=$OPTARG ;;
    b) param_b=$OPTARG ;;
    *) echo "Invalid option" ;;
  esac
done

echo "Param A: $param_a"
echo "Param B: $param_b"
```

15. How do you perform string manipulation in a shell script?

Question: How do you perform string manipulation in a shell script?

Answer:

- **Extract substring:**

  ```
  string="Hello, World!"
  substring=${string:7:5}  # Output: World
  ```

- **String length:**

  ```
  length=${#string}  # Output: 13
  ```

- **Replace substring:**

```
new_string=${string/World/Universe}  # Output: Hello, Universe!
```

1. Scenario: Monitoring Disk Usage

Question: How would you write a shell script to monitor disk usage and send an email alert if usage exceeds 90%?

Answer:

```
#!/bin/bash

THRESHOLD=90
EMAIL="admin@example.com"

df -h | grep -vE '^Filesystem|tmpfs|cdrom' | awk '{ print $5 " " $1 }' | while read output; do
  usage=$(echo $output | awk '{ print $1}' | sed 's/%//g')
  partition=$(echo $output | awk '{ print $2 }')
  if [ $usage -ge $THRESHOLD ]; then
    echo "Running out of space \"$partition ($usage%)\"" | mail -s "Disk Space Alert: $partition ($usage%)" $EMAIL
  fi
done
```

Explanation:

- `df -h`: Get disk usage in human-readable format.
- `grep -vE '^Filesystem|tmpfs|cdrom'`: Exclude certain filesystems.
- `awk '{ print $5 " " $1 }'`: Print usage and partition.
- Loop through each line and check if usage exceeds threshold, then send an email.

2. Scenario: Backup and Clean-up Logs

Question: Write a shell script to backup log files from `/var/log` to `/backup/logs` and delete log files older than 7 days.

Answer:

```
#!/bin/bash

SOURCE_DIR="/var/log"
BACKUP_DIR="/backup/logs"
DAYS=7

# Create backup directory if it doesn't exist
mkdir -p $BACKUP_DIR
```

```
# Backup log files
cp $SOURCE_DIR/*.log $BACKUP_DIR/

# Delete log files older than 7 days
find $SOURCE_DIR/*.log -type f -mtime +$DAYS -exec rm {} \;
```

Explanation:

- `mkdir -p $BACKUP_DIR`: Create backup directory if not exists.
- `cp $SOURCE_DIR/*.log $BACKUP_DIR/`: Copy log files to backup directory.
- `find $SOURCE_DIR/*.log -type f -mtime +$DAYS -exec rm {} \;`: Delete files older than 7 days.

3. Scenario: Checking Service Status

Question: How would you write a script to check if a service (e.g., `apache2`) is running and restart it if it's not?

Answer:

```
#!/bin/bash

SERVICE="apache2"

if systemctl is-active --quiet $SERVICE; then
  echo "$SERVICE is running"
else
  echo "$SERVICE is not running, restarting..."
  systemctl start $SERVICE
  if systemctl is-active --quiet $SERVICE; then
    echo "$SERVICE restarted successfully"
  else
    echo "Failed to restart $SERVICE"
  fi
fi
```

Explanation:

- `systemctl is-active --quiet $SERVICE`: Check if service is running.
- If not running, restart the service and check status again.

4. Scenario: User Account Management

Question: Write a shell script to create a user account, set a password, and ensure the user's home directory is created.

Answer:

```
#!/bin/bash
```

```
USERNAME=$1
PASSWORD=$2

if [ -z "$USERNAME" ] || [ -z "$PASSWORD" ]; then
  echo "Usage: $0 <username> <password>"
  exit 1
fi

# Create user account
useradd -m -s /bin/bash $USERNAME

# Set user password
echo "$USERNAME:$PASSWORD" | chpasswd

# Check if user was created successfully
if id "$USERNAME" &>/dev/null; then
  echo "User $USERNAME created successfully"
else
  echo "Failed to create user $USERNAME"
fi
```

Explanation:

- `$1` and `$2` for username and password.
- `useradd -m -s /bin/bash $USERNAME`: Create user with home directory and bash shell.
- `echo "$USERNAME:$PASSWORD" | chpasswd`: Set user password.

5. Scenario: File Synchronization

Question: Write a script to synchronize files between two directories (`/source` and `/destination`) and log the synchronization process.

Answer:

```
#!/bin/bash

SOURCE_DIR="/source"
DEST_DIR="/destination"
LOG_FILE="/var/log/sync.log"

# Synchronize files
rsync -av --delete $SOURCE_DIR/ $DEST_DIR/ > $LOG_FILE 2>&1

# Check if rsync was successful
if [ $? -eq 0 ]; then
  echo "Synchronization completed successfully" >> $LOG_FILE
else
  echo "Synchronization failed" >> $LOG_FILE
fi
```

Explanation:

- `rsync -av --delete $SOURCE_DIR/ $DEST_DIR/`: Synchronize directories.
- Log the output and check rsync exit status.

6. Scenario: Parsing a Configuration File

Question: Write a script to parse a configuration file (`config.cfg`) and print each key-value pair.

Answer:

```
#!/bin/bash

CONFIG_FILE="config.cfg"

if [ ! -f $CONFIG_FILE ]; then
  echo "Configuration file not found!"
  exit 1
fi

while IFS='=' read -r key value; do
  # Skip comments and empty lines
  if [[ "$key" =~ ^#.* ]] || [ -z "$key" ]; then
    continue
  fi
  echo "Key: $key, Value: $value"
done < $CONFIG_FILE
```

Explanation:

- `IFS='=' read -r key value`: Split each line by =.
- Skip comments and empty lines, then print key-value pairs.

7. Scenario: Archive Old Files

Question: Write a script to archive files in `/data` older than 30 days into `/archive`.

Answer:

```
#!/bin/bash

SOURCE_DIR="/data"
ARCHIVE_DIR="/archive"
DAYS=30

# Create archive directory if it doesn't exist
mkdir -p $ARCHIVE_DIR
```

```
# Find and archive files older than 30 days
find $SOURCE_DIR -type f -mtime +$DAYS -exec mv {}
$ARCHIVE_DIR/ \;

# Verify the files have been moved
if [ $? -eq 0 ]; then
  echo "Files older than $DAYS days have been archived."
else
  echo "Failed to archive files."
fi
```

Explanation:

- `mkdir -p $ARCHIVE_DIR`: Create archive directory if not exists.
- `find $SOURCE_DIR -type f -mtime +$DAYS -exec mv {} $ARCHIVE_DIR/ \;`: Move files older than 30 days to archive directory.

8. Scenario: Automatic Database Backup

Question: Write a script to backup a MySQL database and delete backups older than 7 days.

Answer:

```
#!/bin/bash

DB_NAME="mydatabase"
DB_USER="dbuser"
DB_PASS="dbpassword"
BACKUP_DIR="/backup/db"
DAYS=7

# Create backup directory if it doesn't exist
mkdir -p $BACKUP_DIR

# Backup database
backup_file="$BACKUP_DIR/$DB_NAME-$(date +%F).sql"
mysqldump -u $DB_USER -p$DB_PASS $DB_NAME > $backup_file

# Check if backup was successful
if [ $? -eq 0 ]; then
  echo "Database backup successful: $backup_file"
else
  echo "Database backup failed!"
  exit 1
fi

# Delete backups older than 7 days
find $BACKUP_DIR -type f -mtime +$DAYS -exec rm {} \;
```

Explanation:

- `mkdir -p $BACKUP_DIR`: Create backup directory if not exists.
- `mysqldump`: Backup the database.
- `find $BACKUP_DIR -type f -mtime +$DAYS -exec rm {} \;`: Delete backups older than 7 days.