Shell Scripting

13 July 2025 16:32

🐧 Shell Scripting: From Basics to Advanced (Complete Notes)

🧠 1. What is a Shell?

- A **Shell** is a program that acts as an interface between the user and the operating system.
- It interprets the commands entered by the user and executes them.

Types of Shells:

Shell	Description
sh	Bourne Shell (original UNIX shell)
bash	Bourne Again Shell (most common)
zsh	Z Shell (feature-rich)
ksh	Korn Shell
csh	C Shell (C-like syntax)

📜 2. What is Shell Scripting?

- **Shell scripting** is the process of writing a sequence of commands in a file to automate tasks.
- These files are called **shell scripts**, typically saved with .sh extension.

🛠️ 3. Getting Started with a Shell Script

🏁 Writing Your First Script

```
#!/bin/bash
echo "Welcome to Shell Scripting!"
```

- The line #!/bin/bash is called a **Shebang**. It tells the system to use the **bash interpreter**.
- Save the file as hello.sh

🔑 Make Script Executable

```
chmod +x hello.sh
./hello.sh
```

📦 4. Variables in Shell

◇ Declaring and Using Variables

```
name="Ravi"
age=25
echo "Name: $name"
echo "Age: $age"
```

No spaces around =

◇ Variable Types

Type	Example

String	name="Ravi"
Integer	num=5
Read-only	readonly pi=3.14
Unset	unset name

⌨ 5. User Input

```
read -p "Enter your name: " username
echo "Hello, $username!"
```

- read is used to get input from the user.
- -p allows prompting the user inline.

🔢 6. Arithmetic Operations

Basic Arithmetic

```
a=10
b=5
echo "Sum = $((a + b))"
echo "Product = $((a * b))"
```

With expr

```
expr $a + $b
```

Floating-point (with bc)

```
echo "scale=2; 5 / 3" | bc
```

🔁 7. Conditional Statements

If Statement

```
if [ $a -gt $b ]; then
  echo "$a is greater"
fi
```

If-Else

```
if [ $a -gt $b ]; then
  echo "a > b"
else
  echo "b >= a"
fi
```

If-ElseIf-Else (elif)

```
if [ $a -gt $b ]; then
  echo "a > b"
elif [ $a -lt $b ]; then
  echo "a < b"
else
  echo "a == b"
fi
```

Integer Comparison Operators

Operator	Description
-eq	Equal
-ne	Not equal

-gt	Greater than
-lt	Less than
-ge	Greater or equal
-le	Less or equal

String Comparison

```
if [ "$str1" = "$str2" ]; then
  echo "Strings are equal"
fi
```

🔄 8. Loops in Shell

For Loop

```
for i in 1 2 3 4 5
do
  echo $i
done
```

While Loop

```
i=1
while [ $i -le 5 ]
do
  echo $i
  ((i++))
done
```

Until Loop

```
i=1
until [ $i -gt 5 ]
do
  echo $i
  ((i++))
done
```

🧨 9. Arrays in Shell

Define and Access

```
fruits=("apple" "banana" "cherry")
echo ${fruits[0]}   # apple
echo ${fruits[@]}   # all
echo ${#fruits[@]}  # count
```

Loop Through Array

```
for item in "${fruits[@]}"
do
  echo $item
done
```

🎛️ 10. Case Statement (Switch-like)

```
read -p "Enter a number: " num
case $num in
  1) echo "One";;
  2) echo "Two";;
```

```
 *) echo "Unknown";;
esac
```

11. Functions

```
greet() {
  echo "Hello, $1"
}
greet "Ravi"
```

- $1, $2 are positional parameters

12. File and Directory Operations

Create a File

```
touch myfile.txt
```

Write to a File

```
echo "Hello" > file.txt
```

Append

```
echo "New Line" >> file.txt
```

Read Line by Line

```
while read line
do
  echo $line
done < file.txt
```

13. Test Conditions

```
if [ -f file.txt ]; then
  echo "File exists"
fi
```

Test Type	Syntax
File exists	[-f filename]
Dir exists	[-d dirname]
Null string	[-z "$var"]
Non-empty	[-n "$var"]

14. Command Line Arguments

```
echo $0   # script name
echo $1   # first arg
echo $#   # number of args
echo $@   # all args
```

15. Cron Jobs (Scheduling Scripts)

```
crontab -e
```

Cron Syntax

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

Field	Value

Minute	0–59
Hour	0–23
Day	1–31
Month	1–12
Weekday	0–6 (Sun=0)

16. Advanced Topics

➤ Debugging Script

```
bash -x script.sh   # step-by-step execution
```

➤ trap Command (Signal Handling)

```
trap "echo Exiting...; exit" SIGINT SIGTERM
```

➤ Nested Loops

```
for i in 1 2; do
  for j in a b; do
    echo "$i $j"
  done
done
```

17. Tools Often Used in Shell Scripts

- cut, awk, sed, xargs, grep, find, sort, uniq, tee, tr, wc, head, tail
- Example with awk:

```
awk -F: '{ print $1 }' /etc/passwd
```

18. Real-World Examples

◇ Backup Script

```
#!/bin/bash
cp /home/user/data.txt /backup/data-$(date +%F).txt
```

◇ Disk Usage Alert

```
#!/bin/bash
usage=$(df -h / | awk 'NR==2 {print $5}' | tr -d '%')
if [ $usage -gt 80 ]; then
  echo "Disk usage is above 80%!"
fi
```

◇ User Check Script

```
#!/bin/bash
read -p "Enter username: " user
if id "$user" >/dev/null 2>&1; then
  echo "User exists"
else
  echo "User does not exist"
fi
```

Here are **real-world mini Shell Scripting projects** you can **practice and build step-by-step**. Each one is

practical, widely used, and designed to help you learn **real Linux automation**.

🛠️ Mini Projects to Practice Shell Scripting

◇ Project 1: Daily System Health Monitor

📌 Goal:

Monitor CPU, RAM, and disk usage and log them daily with timestamp.

🗁 Features:

- Logs system metrics to a file
- Runs daily via cron
- Alerts if CPU or RAM > 80%

🗎 Script Outline:

```
#!/bin/bash
logfile="/var/log/system_health.log"
echo "----- $(date) -----" >> $logfile
cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')
ram_usage=$(free | awk '/Mem:/ { printf("%.2f"), $3/$2 * 100.0 }')
disk_usage=$(df -h / | awk 'NR==2 {print $5}' | tr -d '%')
echo "CPU Usage: $cpu_usage%" >> $logfile
echo "RAM Usage: $ram_usage%" >> $logfile
echo "Disk Usage: $disk_usage%" >> $logfile
if (( $(echo "$cpu_usage > 80.0" | bc -l) )); then
  echo "ALERT: High CPU usage!" >> $logfile
fi
```

🔄 Set in cron to run daily: crontab -e

◇ Project 2: Automatic File Backup Script

📌 Goal:

Backup files from a source to destination folder with timestamp.

🗁 Features:

- Adds current date to filename
- Checks if backup folder exists
- Supports logs

🗎 Script Outline:

```
#!/bin/bash
src="/home/user/data"
dest="/home/user/backup"
timestamp=$(date +%F-%H-%M-%S)
filename="data-backup-$timestamp.tar.gz"
mkdir -p $dest
tar -czf $dest/$filename $src
echo "Backup completed at $timestamp" >> /home/user/backup/backup.log
```

◇ Project 3: User Login Tracker

📌 Goal:

Track and log when users log in to the system.

🗁 Features:

- Uses last command
- Filters current date logins

- Outputs to daily file

Script Outline:

```
#!/bin/bash
log_date=$(date +%F)
log_file="/var/log/user-login-$log_date.log"
last | grep "$(date +'%b %_d')" > $log_file
```

You can run this every evening via cron.

Project 4: Dead URL Checker from a File

Goal:

Read a list of URLs and check if they are alive or dead.

Features:

- Uses curl
- Reads from file
- Outputs dead links

Script Outline:

```
#!/bin/bash
input="urls.txt"
output="dead_urls.txt"
> $output
while read url; do
 if ! curl -s --head --request GET $url | grep "200 OK" > /dev/null; then
  echo "$url is DEAD" >> $output
 fi
done < $input
```

Project 5: Simple Menu-Driven Utility

Goal:

Create a menu to show options like date, list files, check disk usage.

Features:

- Interactive loop
- Uses case statement

Script Outline:

```
#!/bin/bash
while true; do
 echo "1. Show Date"
 echo "2. List Files"
 echo "3. Disk Usage"
 echo "4. Exit"
 read -p "Choose option: " opt
case $opt in
  1) date ;;
  2) ls -l ;;
  3) df -h ;;
  4) exit ;;
  *) echo "Invalid Option" ;;
 esac
done
```

Project 6: Monitor and Restart a Service

📌 Goal:
Check if a service (e.g., nginx) is running. If not, restart it.

🗂️ Features:
- Checks using systemctl
- Sends alert
- Logs to file

📄 Script Outline:
```
#!/bin/bash
service="nginx"
logfile="/var/log/service_monitor.log"
if ! systemctl is-active --quiet $service; then
  echo "$(date): $service is down. Restarting..." >> $logfile
  systemctl restart $service
else
  echo "$(date): $service is running." >> $logfile
fi
```

🔸 Project 7: Auto Compress Old Log Files

📌 Goal:
Find .log files older than 7 days and compress them.

🗂️ Features:
- Uses find
- Auto gzip compression
- Logs operation

📄 Script Outline:
```
#!/bin/bash
log_dir="/var/log/myapp"
find $log_dir -type f -name "*.log" -mtime +7 -exec gzip {} \;
echo "$(date): Old logs compressed" >> /var/log/log_cleanup.log
```

☑️ Bonus: Ideas for More Practice

Idea	Description
Auto Email Report	Use mail or mailx to send daily reports
SSH Login Alert	Trigger alert on remote login using .bashrc
Folder Watcher	Use inotifywait to monitor changes
Ping Monitor	Check server connectivity every X mins
Log Analyzer	Parse and summarize custom logs

Let's take **Project 1: Daily System Health Monitor** and break it down step-by-step. This is a **real-world automation script** commonly used by DevOps/System Admins to **monitor Linux server health** daily.

🛠️ Project: Daily System Health Monitor

A bash script to monitor CPU, RAM, and Disk usage with log alerts if thresholds are breached.

📌 Objective

☑ Monitor system health (CPU, RAM, Disk)
☑ Log the data with timestamps
☑ Alert if usage crosses thresholds (e.g., 80%)
☑ Automate via cron for daily monitoring

🗁 File Structure

You will create just **one file**:
system_health.sh
And it will write logs to:
/var/log/system_health.log

🗒 Step-by-Step Breakdown

☑ Step 1: Create Script File

touch system_health.sh
chmod +x system_health.sh
nano system_health.sh

☑ Step 2: Add Shebang

#!/bin/bash
Tells Linux to use bash interpreter.

☑ Step 3: Set Log File Path

logfile="/var/log/system_health.log"
You can change this path if you don't have sudo access.

☑ Step 4: Log the Current Date and Time

echo "---------- $(date) ----------" >> $logfile
Adds a header for each entry.

☑ Step 5: Get CPU Usage

cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')
Explanation:

- top -bn1: Run top once in batch mode
- grep "Cpu(s)": Filter CPU line
- awk '{print $2 + $4}': Sum user and system CPU usage

☑ Step 6: Get RAM Usage

ram_usage=$(free | awk '/Mem:/ { printf("%.2f"), $3/$2 * 100.0 }')
Explanation:

- free: Shows memory usage
- $3 = used, $2 = total, calculate used percentage

☑ Step 7: Get Disk Usage

disk_usage=$(df -h / | awk 'NR==2 {print $5}' | tr -d '%')
Explanation:

- df -h /: Disk usage of root (/)
- awk 'NR==2 {print $5}': Extract percentage

- tr -d '%': Remove % for numeric comparison

☑ Step 8: Log the Values

```
echo "CPU Usage: $cpu_usage%" >> $logfile
echo "RAM Usage: $ram_usage%" >> $logfile
echo "Disk Usage: $disk_usage%" >> $logfile
```

☑ Step 9: Add Alert Thresholds

```
if (( $(echo "$cpu_usage > 80.0" | bc -l) )); then
  echo "⚠ ALERT: High CPU usage!" >> $logfile
fi
if (( $(echo "$ram_usage > 80.0" | bc -l) )); then
  echo "⚠ ALERT: High RAM usage!" >> $logfile
fi
if [ $disk_usage -gt 80 ]; then
  echo "⚠ ALERT: High Disk usage!" >> $logfile
fi
```

bc -l: Enables floating point comparison

☑ Step 10: Final Script

```
#!/bin/bash
logfile="/var/log/system_health.log"
echo "---------- $(date) ----------" >> $logfile
cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')
ram_usage=$(free | awk '/Mem:/ { printf("%.2f"), $3/$2 * 100.0 }')
disk_usage=$(df -h / | awk 'NR==2 {print $5}' | tr -d '%')
echo "CPU Usage: $cpu_usage%" >> $logfile
echo "RAM Usage: $ram_usage%" >> $logfile
echo "Disk Usage: $disk_usage%" >> $logfile
if (( $(echo "$cpu_usage > 80.0" | bc -l) )); then
  echo "⚠ ALERT: High CPU usage!" >> $logfile
fi
if (( $(echo "$ram_usage > 80.0" | bc -l) )); then
  echo "⚠ ALERT: High RAM usage!" >> $logfile
fi
if [ $disk_usage -gt 80 ]; then
  echo "⚠ ALERT: High Disk usage!" >> $logfile
fi
echo "" >> $logfile
```

⏰ Step 11: Automate with Cron

To run daily at 9 AM:

```
crontab -e
```

Add:

```
0 9 * * * /path/to/system_health.sh
```

🗀 Output Example (Log File)

```
---------- Sun Jul 14 09:00:00 IST 2025 ----------
CPU Usage: 15.2%
```

RAM Usage: 63.54%
Disk Usage: 78%

☑ Bonus Tips

- You can send email alerts using mail command.
- Use uptime, iostat, and vmstat for more system info.
- Use logrotate to manage log file size.

Here's a **comprehensive list of Shell Scripting interview questions** — categorized by **beginner, intermediate, and advanced levels** — along with **sample answers**, real-world context, and tips to impress the interviewer.

📘 Beginner-Level Shell Scripting Questions

1. What is a Shell?

Answer: A shell is a command-line interpreter that allows users to interact with the operating system by executing commands. Common shells include sh, bash, zsh, and ksh.

2. How do you create a simple shell script?

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

Save as hello.sh, make it executable with chmod +x hello.sh, then run ./hello.sh.

3. What is the use of chmod +x?

Answer: It makes a shell script executable.

4. How do you pass arguments to a shell script?

Answer:

```
echo "First argument is $1"
echo "All arguments: $@"
```

5. What is the purpose of $0, $1, $#, $@, $??

Variable	Purpose
$0	Script name
$1	First argument
$@	All arguments as separate strings
$#	Number of arguments
$?	Exit status of the last command

6. What is a shebang (#!) in a script?

Answer: It's the first line that tells the OS which interpreter to use (e.g., #!/bin/bash).

7. How do you read user input in shell script?

```
read -p "Enter name: " name
```

```
echo "Hello $name"
```

8. How do you perform arithmetic operations?

```
sum=$((5 + 3))
echo $sum
```

9. What is the difference between > and >>?

Symbol	Meaning
>	Overwrite a file
>>	Append to a file

10. How do you comment a line in shell?

Answer: Use # for single-line comments.

⚙ Intermediate-Level Shell Scripting Questions

11. What are loops available in shell scripting?

- for
- while
- until

Example:

```
for i in 1 2 3; do echo $i; done
```

12. How do you check if a file exists?

```
if [ -f "file.txt" ]; then
  echo "File exists"
fi
```

13. How do you check if a variable is empty?

```
if [ -z "$var" ]; then
  echo "Variable is empty"
fi
```

14. What is the difference between [ ] and [[ ]]?

Brackets	Features
[ ]	Traditional test
[[ ]]	Bash extension (supports &&, `

15. Explain case statement with example.

```
case $var in
  1) echo "One" ;;
  2) echo "Two" ;;
  *) echo "Other" ;;
esac
```

16. What is set -e and set -x?

Command	Purpose
set -e	Exit on any command failure
set -x	Debug mode: shows each command

17. What is a function in shell script?

```
greet() {
  echo "Hello $1"
}
greet "Ravi"
```

18. How do you log script output to a file?

```
./script.sh > output.log 2>&1
```

19. What is the purpose of trap?

Answer: To catch signals (e.g., SIGINT) and execute a cleanup command.

```
trap "echo 'Exiting...'" SIGINT
```

20. What is the use of && and ||?

```
mkdir mydir && cd mydir     # executes cd only if mkdir succeeds
rm file || echo "Failed"    # echo only if rm fails
```

🚀 Advanced-Level Shell Scripting Questions

21. How do you handle floating-point arithmetic?

```
echo "scale=2; 10 / 3" | bc
```

22. How do you schedule a script using cron?

```
crontab -e
Example entry:
0 8 * * * /home/user/script.sh
```

23. What are associative arrays in Bash?

```
declare -A user
user[name]="Ravi"
user[age]=30
echo ${user[name]}
```

24. How do you monitor a process and restart it if it fails?

```
if ! pgrep nginx > /dev/null; then
  systemctl restart nginx
fi
```

25. How do you find and compress .log files older than 7 days?

```
find /var/log -name "*.log" -mtime +7 -exec gzip {} \;
```

26. How do you read a file line by line?

```
while read line; do
  echo $line
done < file.txt
```

27. Explain IFS variable in shell.

Answer: IFS (Internal Field Separator) defines the delimiter used to split input.

```
IFS=',' read -r name age <<< "Ravi,25"
```

28. How do you extract data from a string using cut, awk, or sed?

- cut:

```
echo "a:b:c" | cut -d ':' -f2
```

- awk:

```
awk -F':' '{print $2}' <<< "a:b:c"
```

- sed:

```
echo "Hello World" | sed 's/World/Universe/'
```

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

Expression	Meaning
$*	All arguments as a single string
$@	All arguments as separate strings

30. How do you write a script to check internet connectivity?

```
ping -c 1 google.com &> /dev/null
if [ $? -eq 0 ]; then
  echo "Online"
else
  echo "Offline"
fi
```

☑ Bonus: Behavioral Interview Questions (Shell Scripting)

◇ "Tell me about a time you automated a task using shell script."

☑ Talk about:

- The manual task
- Your approach to scripting
- Time saved or errors reduced
- Tools used (cron, systemd, logging)

◇ "Have you handled script failures in production?"

☑ Mention:

- Using set -e and trap
- Logging mechanisms
- Fallback plans or notifications