Devops

Shell scripting part 3

COMMONLY USED SHELL COMMANDS AND INTERVIEW QUESTIONS

LIST OF MOST COMMONLY USED SHELL COMMANDS

1. ls - List files and folders
2. cp - Copy files
3. mv - Move or rename files
4. mkdir - Create directory
5. touch - Create empty file
6. vi - Text editor
7. grep - Search for patterns in files
8. df - Disk space usage
9. ps -ef - List all processes
10. awk - Text processing tool
11. curl - Transfer data from URLs
12. find - Find files and directories

SHELL SCRIPTING INTERVIEW QUESTIONS AND ANSWERS

Question 1: List all the processes with only Process IDs

Command: ps -ef | awk -F" " '{print $2}'

Explanation:

- ps -ef: Lists all running processes
- | (pipe): Sends output to next command
- awk -F" ": awk command with space as field separator
- '{print $2}': Prints the 2nd column (Process ID)

Output: Lists only the Process IDs of all running processes.

Question 2: Script to print error from a remote log

Command: curl logfile_url | grep ERROR

Explanation:

- curl logfile_url: Fetches content from remote log file
- | (pipe): Sends output to grep
- grep ERROR: Filters lines containing "ERROR"

Example: curl https://example.com/app.log | grep ERROR

Output: Displays only the lines containing "ERROR" from the remote log file.

Question 3: Shell script to print numbers divisible by 3 and 5 but NOT by 15

Script:

```
#!/bin/bash for i in {1..100}; do if ([ expr $i % 3 == 0 ] || [ expr $i % 5 == 0 ]) && [ expr $i %
15 != 0 ]; then echo $i fi done
```

Explanation:

- for i in {1..100}: Loop from 1 to 100
- expr $i % 3: Calculate remainder when i is divided by 3
- expr $i % 5: Calculate remainder when i is divided by 5
- expr $i % 15: Calculate remainder when i is divided by 15
- [expr $i % 3 == 0]: Check if divisible by 3
- [expr $i % 5 == 0]: Check if divisible by 5
- [expr $i % 15 != 0]: Check if NOT divisible by 15
- || (OR operator): Either condition can be true
- && (AND operator): Both conditions must be true
- echo $i: Print the number

Logic: Print numbers that are divisible by either 3 OR 5, AND NOT divisible by 15.

Example Output: 3, 5, 6, 9, 10, 12, 18, 20, 21, 24, 25, 27, 33, 35...

Question 4: Write a script to count the number of 's' in "mississippi"

Script:

```
x=mississippi grep -o "s" <<< "$x" | wc -l
```

Explanation:

- x=mississippi: Store the word in variable x
- grep -o "s": Search for letter "s", -o flag prints each match on new line
- <<< "$x": Here-string, passes the value of $x to grep
- | (pipe): Sends output to wc
- wc -l: Counts the number of lines (which equals number of 's')

Output: 4

(There are 4 's' letters in "mississippi")

Question 5: How will you debug your shell script?

Answer: Use set -x command

Script Example:

```
#!/bin/bash set -x echo "Starting script" df -h free -g
```

Explanation:

- set -x: Enables debug mode
- Shows each command before execution with + prefix
- Helps identify which command is failing
- Shows variable expansion and command substitution

Additional Debug Options:

- set -e: Exit on error
- set -v: Print shell input lines as they are read
- set -u: Exit when using undefined variable
- set -exo pipefail: Combination of all best practices

Question 6: What is crontab in Linux? Provide an example.

Answer: Crontab is like an alarm which when set at a particular time will execute the script automatically.

Syntax:

-
  -
    -
      -
        - command_to_execute

| | | | | | | | | +---- Day of week (0-7) (Sunday=0 or 7) | | | +------ Month (1-12) | | +-------- Day of month (1-31) | +---------- Hour (0-23) +------------ Minute (0-59)

Examples:

1. Run script every day at 2:30 AM: 30 2 * * * /home/user/backup.sh
2. Run script every Monday at 5:00 PM: 0 17 * * 1 /home/user/weekly-report.sh
3. Run script every 5 minutes: */5 * * * * /home/user/monitor.sh
4. Run script on 1st day of every month at midnight: 0 0 1 * * /home/user/monthly-cleanup.sh

Commands:

- crontab -e: Edit crontab
- crontab -l: List all cron jobs
- crontab -r: Remove crontab

Use Cases:

- Automated backups
- Log rotation
- System monitoring
- Scheduled reports
- Database maintenance

Question 7: How to open a read-only file?

Answer:

Command: vi -R test.txt

Explanation:

- vi: Text editor
- -R flag: Opens file in read-only mode
- test.txt: File name

Alternative: view test.txt

This command also opens file in read-only mode.

Why Read-Only?

- Prevent accidental modifications
- View configuration files safely
- Review logs without changing them

Question 8: What is the difference between soft link and hard link?

Hard Link:

Let's say you create a file and save this file. Now this file is saved in the memory. Let's say now you want to reuse this file multiple times, then in this case we can use hard link using ln command.

Command: ln original_file hard_link_file

Characteristics:

- Hard link creates a copy reference to the original file
- Even if actual file gets deleted from memory, hard link will still exist and keep backup of file
- Both point to the same data on disk
- Both have same inode number
- Can only link files, not directories
- Cannot cross filesystem boundaries

Example: ln test.txt test_hardlink.txt

If you delete test.txt, test_hardlink.txt will still have the content.

Soft Link (Symbolic Link):

Soft links are not like that. For example, we have a python3 file. Whenever you execute your python file, Linux terminal will send output to python3 because python has soft link to python3.

Command: ln -s original_file soft_link_file

Characteristics:

- If you delete your original file, soft link will also become broken because both of them are pointing to the same file in the memory
- Soft link is like a shortcut
- Can link directories
- Can cross filesystem boundaries
- Different inode number
- If original is deleted, link becomes broken

Example: ln -s /usr/bin/python3 /usr/bin/python

If you delete python3, the python link will be broken.

Key Difference: Hard Link = Actual backup copy (survives original deletion) Soft Link = Shortcut/pointer (breaks if original deleted)

Question 9: Difference between break and continue statements

break Statement: Break means breaking execution - completely exits the loop.

Example:

#!/bin/bash for i in {1..10}; do if [$i -eq 5]; then break fi echo $i done

Output: 1 2 3 4

(Loop stops at 5)

continue Statement: Continue means continue the execution - skip this iteration and continue to the next.

Example:

#!/bin/bash for i in {1..10}; do if [$i -eq 5]; then continue fi echo $i done

Output: 1 2 3 4 6 7 8 9 10

(Skips 5 but continues the loop)

Summary:

- break: Exit the entire loop
- continue: Skip current iteration, move to next

Question 10: What are the disadvantages of shell scripting?

Disadvantages:

1. **Errors are frequent and costly:**
    - A single error can alter the command
    - Difficult to debug complex scripts
2. **Execution speed is slow:**
    - Every shell command execution launches a new process
    - Slower than compiled languages
3. **Bugs or inadequacies:**
    - Issues in the language's syntax or implementation
    - Different shells (bash, sh, zsh) have different behaviors
4. **Not suitable for large complex tasks:**
    - Limited data structures
    - No object-oriented programming
    - Difficult to maintain large codebases
5. **Minimal data structures:**
    - Provides minimal built-in data structures
    - Limited string manipulation capabilities
6. **Process overhead:**
    - Every time a shell command is executed, a new process is launched
    - Memory and CPU intensive for complex operations
7. **Portability issues:**
    - Scripts may not work across different Unix/Linux variants

    - Different shells have different features

8. **Security concerns:**
    - Vulnerable to injection attacks
    - Difficult to sanitize user input
9. **Limited error handling:**
    - Basic error handling compared to other languages
    - Difficult to implement exception handling
10. **Not ideal for:**
    - GUI applications
    - Heavy mathematical computations
    - Network programming
    - Large-scale applications

Question 11: Is Bash dynamically or statically typed and why?

Answer: Bash is **dynamically typed**.

Explanation:

Dynamic Typing means:

- Variable types are determined at runtime
- No need to declare variable types
- Same variable can hold different types of data

Example:

#!/bin/bash x=10 # x is a number echo $x # Output: 10

x="Hello" # x is now a string echo $x # Output: Hello

x=3.14 # x is now treated as string (bash doesn't have float) echo $x # Output: 3.14

Why Dynamically Typed?

- Bash doesn't require type declarations
- Variables can change type during execution
- Type checking happens at runtime, not compile time

- Provides flexibility but less type safety

Comparison:

- **Static Typing (C, Java):** Must declare type: int x = 10;
- **Dynamic Typing (Bash, Python):** No type declaration: x=10

Question 12: Explain about a network troubleshooting utility

Answer: traceroute

What is traceroute? traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination.

Purpose:

- Identify network routing issues
- Find where packets are getting dropped
- Measure network latency
- Troubleshoot slow connections
- Identify bottlenecks in network path

Command:

Linux: traceroute google.com Windows: tracert google.com

How it works:

1. Sends packets with increasing TTL (Time To Live) values
2. Each router decrements TTL by 1
3. When TTL reaches 0, router sends back ICMP "Time Exceeded" message
4. This reveals each hop in the path

Example Output:

traceroute to google.com (142.250.185.46), 30 hops max 1 192.168.1.1 (192.168.1.1) 1.234 ms 2 10.0.0.1 (10.0.0.1) 5.678 ms 3 isp-router.net (203.0.113.1) 15.234 ms 4 google.com (142.250.185.46) 20.567 ms

Information provided:

- Hop number
- Router IP address

- Router hostname
- Round-trip time (latency)

Use Cases:

- Website not loading - check where connection fails
- Slow network - identify which hop has high latency
- Network configuration issues
- ISP routing problems

Other Network Troubleshooting Tools:

- ping: Check connectivity
- netstat: Network statistics
- nslookup: DNS lookup
- dig: DNS queries
- ifconfig/ip: Network interface configuration
- tcpdump: Packet analyzer
- wireshark: Network protocol analyzer

Question 13: How will you manage logs of a system that generates huge log files everyday?

Answer: Use **logrotate** with compression (gzip, zip)

What is logrotate? logrotate is a system utility that manages automatic rotation, compression, removal, and mailing of log files.

Why needed?

- Log files grow continuously
- Can fill up disk space
- Difficult to analyze huge files
- System performance degrades

How logrotate works:

1. Rotates logs based on size or time
2. Compresses old logs to save space

3. Deletes very old logs automatically
4. Maintains a specific number of old logs

Configuration file: /etc/logrotate.conf

Example Configuration:

```
/var/log/myapp/*.log { daily # Rotate daily rotate 7 # Keep 7 days of logs compress #
Compress old logs delaycompress # Don't compress most recent old log missingok #
Don't error if log file is missing notifempty # Don't rotate if log is empty create 0644 root
root # Create new log with these permissions postrotate # Commands to run after
rotation systemctl reload myapp endscript }
```

Rotation Options:

By Time:

- daily: Rotate every day
- weekly: Rotate every week
- monthly: Rotate every month
- yearly: Rotate every year

By Size:

- size 100M: Rotate when file reaches 100MB
- size 1G: Rotate when file reaches 1GB

Example:

```
/var/log/application.log { size 100M rotate 5 compress delaycompress missingok
notifempty copytruncate }
```

What happens:

1. application.log reaches 100MB
2. application.log renamed to application.log.1
3. application.log.1 compressed to application.log.1.gz
4. New empty application.log created
5. After 5 rotations, oldest log deleted

Compression Methods:

gzip (default):

- Good compression ratio
- Fast
- Command: gzip file.log
- Result: file.log.gz

bzip2:

- Better compression than gzip
- Slower
- Command: bzip2 file.log
- Result: file.log.bz2

xz:

- Best compression
- Slowest
- Command: xz file.log
- Result: file.log.xz

Benefits:

- Saves disk space (10:1 compression ratio typical)
- Organized logs by date
- Automatic cleanup of old logs
- Prevents disk space issues
- Easier log analysis

Manual Commands:

Test logrotate config: logrotate -d /etc/logrotate.conf

Force log rotation: logrotate -f /etc/logrotate.conf

Check logrotate status: cat /var/lib/logrotate/status

Alternative Solutions:

1. Send logs to centralized logging system (ELK Stack, Splunk)
2. Use cloud logging services (CloudWatch, Stackdriver)
3. Stream logs instead of storing locally

4. Use log aggregation tools (Fluentd, Logstash)