LP ASSIGNMENT-8

ENG24CY0159

SHADIK KHAN

11. What is a user-defined function in shell scripting? Explain with an example.

Think of a **user-defined function** as creating your own custom command or a "mini-script" inside your main script. It's a block of code that you give a name to. Once defined, you can run that entire block of code anytime you want, simply by calling its name.

This is incredibly useful for organizing your code and avoiding repetition. If you have a task you need to perform multiple times, you just write the code for it once inside a function.

Example: Here's a simple function that greets a user.

Bash

```
#!/bin/bash


# Define the function
greet() {
  echo "Hello, $1! Welcome."
}
# Call the function with different arguments
greet "Alice"
greet "Bob"
```

Output:

```
Hello, Alice! Welcome.
Hello, Bob! Welcome.
```

Here, greet is our custom command, and $1 is a special variable that holds the first argument we pass to it.

12. Write a bash script with a function that multiplies two integer numbers.

This script defines a function that takes two numbers, multiplies them, and "returns" the result by printing it. We then capture that printed result using command substitution $(...).

Bash

```
#!/bin/bash

# A function to multiply two numbers
multiply() {
  # The input numbers are $1 and $2
  local num1=$1
  local num2=$2

  # Perform the multiplication
  local result=$((num1 * num2))
 # "Return" the result by printing it to standard output
  echo $result
}

# --- Main part of the script ---
echo "Let's multiply 7 and 6."

# Call the function and capture its output into a variable
product=$(multiply 7 6)

echo "The product is: $product"
```

---

13. Explain how arrays (1D, 2D, and 3D) are declared in bash scripting.

This is where bash shows its simplicity (and limitations) compared to other languages.

1D Arrays (The Standard Bash Array) Bash fully supports one-dimensional indexed arrays. You declare them with parentheses, separating elements with spaces.

Bash

```
# Declare a 1D array
distros=("Ubuntu" "Fedora" "Debian" "Arch Linux")


# Access an element (indexing starts at 0)
echo ${distros[0]}  # Outputs: Ubuntu
```

2D and 3D Arrays (Not Natively Supported) Here's the trick: **Bash does not support multi-dimensional arrays.** You cannot create a true matrix[row][col].

However, clever programmers have workarounds to simulate them:

- **Using Strings:** You can create strings with delimiters (like commas or semicolons) to represent rows and then parse them.
- **Using Associative Arrays:** You can use key-value pairs where the key is a string like "row,col" to simulate a 2D array. This is the more advanced method.

But for a direct answer, bash only handles simple, one-dimensional arrays.

---

14. Write a shell script to display elements of an array.

The best way to display all elements of an array is to loop through them. The special syntax "${my_array[@]}" is used to get all the elements correctly, even if they contain spaces.

Bash

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

```
# Declare an array with some elements
servers=("web-server-01" "db-server" "api-gateway" "backup server")

echo "--- Server List ---"

# Loop through each element in the array
for server in "${servers[@]}"
do
  echo "Server found: $server"
done

echo "--- End of List ---"
```

---

15. What is the purpose of cron in Linux?

Think of **cron** as a smart alarm clock for your computer. Its purpose is to be a **time-based job scheduler**.

It runs as a background process (a daemon) and constantly checks a configuration file called the crontab. You edit this file to give it a schedule of commands or scripts you want to run automatically. cron will then execute them for you at the precise minute, hour, day, or month you specified.

It's the engine behind almost all scheduled automation on Linux, used for things like:

- Running nightly backups.
- Cleaning up temporary files.
- Sending out automated reports.
- Checking for system updates.

---

16. Write a cron job to run a backup script every day at midnight.

To do this, you would edit your crontab file by running the command crontab -e. Then, you would add the following line at the bottom.

Cron Job Entry:

0 0 * * * /home/user/scripts/run_backup.sh

Here's what that line means:

- 0: Run at the 0th minute.
- 0: Run at the 0th hour (midnight).
- *: Run on every day of the month.
- *: Run in every month.
- *: Run on every day of the week.
- /home/user/scripts/run_backup.sh: This is the absolute path to the command or script you want to execute.

---

17. How do you schedule a one-time job using the at command?

If cron is for recurring tasks, at is for tasks you want to run **only once** in the future. It's perfect for scheduling a command to run later today or tomorrow.

Example: Let's say you want to safely shut down your server at 11:30 PM tonight.

1. Type the at command followed by the time:

at 11:30 PM

2. Your terminal will change to an at> prompt. Type the command you want to run:
3. at> /sbin/shutdown -h now
4. Press **Ctrl+D** on your keyboard to save and exit.

The system will respond with a confirmation, and your command is now scheduled to run at 11:30 PM, just once.

---

18. Write a script to display disk usage using df and du.

This script uses df to get an overview of the whole filesystem and du to get a summary of the current user's home directory.

df (disk free) is like asking, "How full is the entire filing cabinet?" **du (disk usage)** is like asking, "How much space is this one specific folder taking up?"

Bash

```
#!/bin/bash


# This script provides a summary of disk usage.


echo "========================================="
echo "      Filesystem Disk Space Usage (df)"
echo "========================================="
# The -h flag makes the output "human-readable" (e.g., KB, MB, GB)
df -h


echo ""
echo "========================================="
echo "    Home Directory Disk Usage (du)"
echo "========================================="
# -s provides a grand total (summary)
# -h makes it human-readable
du -sh /home/$USER
echo "========================================="
```

---

19. How can you log the output of a script using the tee command?

The tee command is perfect for this. It acts like a T-splitter in a pipe, taking the output from your script and sending it to two places at once: **to your screen** so you can watch it, and **to a file** for logging.

You use the pipe | to connect your script's output to tee.

Example: To see the output and save it to a new log file:

Bash

```
bash my_script.sh | tee script_output.log
```

To see the output and **append** it to an existing log file (without erasing it), use the -a flag:

Bash

```
bash my_script.sh | tee -a script_output.log
```

---

20. Explain with an example how shell scripting can automate system administration tasks.

System administrators have to do many repetitive tasks every day to keep servers healthy. Shell scripting is their best friend because it can automate these tasks, saving time and preventing human error.

Example: A Daily Server Health Check Report

A sysadmin needs to check three things every morning: disk space, memory usage, and whether the critical web server is still running. Instead of doing this manually, they can write a shell script to be their "robot assistant."

Bash

```
#!/bin/bash


# --- Automated Daily Health Check Script ---


REPORT_FILE="/tmp/health_report_$(date +%F).txt"

ADMIN_EMAIL="admin@example.com"
```

```
# Clear the report file
> "$REPORT_FILE"

echo "--- Health Report for $(hostname) on $(date) ---" >> "$REPORT_FILE"
echo "" >> "$REPORT_FILE"

# 1. Check disk space
echo "--- Disk Usage ---" >> "$REPORT_FILE"
df -h >> "$REPORT_FILE"
echo "" >> "$REPORT_FILE"

# 2. Check memory usage
echo "--- Memory Usage ---" >> "$REPORT_FILE"
free -h >> "$REPORT_FILE"
echo "" >> "$REPORT_FILE"

# 3. Check if web server (apache2) is running
echo "--- Web Server Status ---" >> "$REPORT_FILE"
if pgrep -x "apache2" > /dev/null
then
   echo "Apache web server is RUNNING." >> "$REPORT_FILE"
else
   echo "CRITICAL: Apache web server is DOWN." >> "$REPORT_FILE"
fi
```

```
# 4. Email the report to the administrator
mail -s "Daily Health Report for $(hostname)" "$ADMIN_EMAIL" < "$REPORT_FILE"
```

The Automation: The best part is that the sysadmin can add this script to cron to run automatically every morning at 8 AM. They can walk into the office, and the complete health report will already be waiting in their inbox. This is the power of automation.
