Linux Programming: Assignment-7

Samriddhi Guha | ENG24CY0157 | 3-C (25)

1. What is a bash shell script? Give one example. (CO4)

A Bash shell script is a straight text document that has a series of commands that execute by the Bash (Bourne Again SHell) interpreter. These scripts can be used to automate tasks, bundle multiple commands into one executable script, or carry out system administration (or other repetitive) tasks within Unix-like operating systems. Bash scripts can contain the equivalent of any basic shell command, control flow structures (such as loops and conditional statements), and variable declarations.

Example:

```
#!/bin/bash
# This script greets the user and displays the current date and time.

echo "Hello, user!"
echo "Today is: $(date)"
```

Here,

- #!/bin/bash: This is called the "shebang" and specifies the interpreter that should be used to execute the script. In this case, it's the Bash interpreter located at /bin/bash.
- # This script greets the user and displays the current date and time. Comments are ignored by the interpreter and are used to provide explanations within the script.
- echo "Hello, user!": This command prints the string "Hello, user!" to the standard output (usually the terminal).
- echo "Today is: $(date)": This command also prints a string, but it includes the output of another command. $(date) executes the date command, which returns the current date and time, and substitutes its output into the echo command's argument.

2. Write a simple shell script to print “Hello World”. (CO4)

```
#!/bin/bash
# This script prints ‘Hello World’.
echo "Hello World"
```

Here,

- #!/bin/bash: This is called the "shebang" and specifies the interpreter (in this case, Bash) to execute the script.

- echo "Hello World": This command prints the string "Hello World" to the standard output (the terminal).
- Save and exit: Save the file and exit the text editor.
- Make the script executable: Grant execute permissions to the script using the chmod command.

3. What is the purpose of comments (#) in a shell script? (CO4)

In shell scripting, the hash sign (#) is primarily used to indicate a comment. Any text that follows a # on a line is ignored by the shell interpreter and will not run as part of the script.

The specific purposes of comments in shell scripts include:

- Explanation: Comments provide explantion of the script's logic, purpose or a particular command, making the code human readable. This is especially important when returning to complex scripts or when the script is shared with other developers.

- Readability: The addition of meaningfully placed comments means that any developer reading the code can follow the execution flow and understand the intent of different sections, providing for improved maintainability.

- Troubleshooting: Comments can be used as an temporary way to disable lines or blocks of code during debugging, allowing a developer to isolate problems or test different sections of the script by simply commenting the code.

- Collaboration: When working as part of a team, the addition of comments allows other team members to understand portions of the script.

- Self-Documentation: Comments can serve to provide a form of self-documentation for the original author of the code to remember details of how and why they implemented parts of the code.

4. How do you declare variables (int, float, double, string, Boolean, and char in a shell script? (CO4)

In Bash shell scripting, variables are natively typeless; they are all strings by default. We do not do any explicit type declaration like int, float, or string in a strongly typed language.

A variable is declared and assigned a value using the syntax: VARIABLE_NAME=value. No spaces are allowed around the equal sign (=).

The declare command is used to impose specific attributes or types onto a variable, primarily for arithmetic.

1. **String (Default):** Assigned directly.

   Example: USER_NAME= "Samriddhi Guha"

2. **Integer (int):** Assigned directly, but **declare -i** is used to force arithmetic context.

   Example: COUNT=10; to enforce: declare -i COUNT

3. **Float/Double:** Bash does not handle floating-point numbers natively. They are treated as strings. Calculations require external tools like bc (basic calculator) or awk.

   Example: PRICE=19.99

4. **Boolean:** Handled as strings. Standard Bash logic uses for true and any non-zero value for false in conditional checks.

   Example: IS_ACTIVE="true"

5. **Character (char):** Handled as a single-character string.

   Example: INITIAL='S'

5. Write a shell script to display the current date and time of the system. (CO4)

The shell script uses the fundamental *date* command to display the system's current time.

```
#!/bin/bash
# Script displays the current date and time.

echo "--- System Date and Time ---"

# The 'date' command prints the current date and time.
echo "The current date and time is: $(date)"

# Using formatting options for a specific look (e.g., YYYY-MM-DD HH:MM:SS)
echo "Formatted Time (Y-M-D H:M:S): $(date +%Y-%m-%d\ %H:%M:%S)"

echo "----------------------------"
```

6. Explain the difference between a constant and a variable in bash script. (CO4)

The difference between a constant and variable concerns mutability, the ability to change a value after having assigned it.

- Variable: Named storage location which can change value at any point during the execution of the script. A variable is the default case or state for any name assigned something.

  Example: *MESSAGE="First value". Then later: MESSAGE="Second value".*

- Constant (Read-Only Variable): Named storage location which cannot change or be unset, once set throughout the execution of the script. You create constants with the readonly or declare -r command. It is an error to try to change a constant.

  Example: *readonly MAX_USERS=10. The script cannot later run MAX_USERS=20.*

7. Write a shell script to read two integer number from the user and compute the sum of both the number. (CO4)

The script uses the *read* command for input and the arithmetic expansion $(( )) construct for calculation.

```
#!/bin/bash
#Script to read two integers and compute their sum
# Read the first number from the user
echo -n "Enter the first integer (NUM1): "
read NUM1

# Read the second number from the user
echo -n "Enter the second integer (NUM2): "
read NUM2

# Compute the sum using arithmetic expansion: $(( expression ))
SUM=$(( NUM1 + NUM2 ))

# Display the final result
echo "-----------------------------------"
echo "The sum of $NUM1 and $NUM2 is: $SUM"
```

8. What is the use of source command in shell scripting? (CO4)

The *source* command (also referred to by a single dot: .) runs a script or file in the current shell environment or session instead of spawning a new subprocess.

- Main Use: To guarantee that the commands in the source file can change the environment of the current shell.

- Value: This is critical when loading configuration files (.bashrc), creating/updating environment variables, and declaring shell functions, so that the changes remain in the interactive or parent shell after the script has been run.

9. How can you debug a shell script? Give two methods. (CO4)

Debugging shell scripts involves keeping track of how commands are executed, variables and their values are created and expanded, and how variable expansions are used in the context of the command being executed. There are two popular and effective methods for debugging shell scripts:

1. Using the Execution Flag (bash -x):
   This is the general and more popular method. The visibility flag, -x, puts bash in a state known as trace mode. In trace mode, bash will output all commands after they have been expanded (substituting variables and processing wildcards) and before they are executed by creating a trace entry that is prefixed with a plus sign (+).

   Method: bash -x script_name.sh

2. Inserting set -xv inside the script:
   This method allows us to turn on debugging for a certain section of your script that is giving you trouble.
   The command set -x puts bash into trace mode.
   The command set -v puts bash in verbose mode (displays input lines as they are read).
   Turn off debugging with the command set +xv.

Example:

```
set -xv  # turn debugging on
# ... buggy code here ...
set +xv  # turn debugging off
```

10. Write a bash script to create and delete a file. (CO4)

This script uses the *touch* command to create a file and the rm command to delete it.

```
#!/bin/bash
# Script to create and delete a temporary file

FILE_NAME="temp_file.dat"

echo "--- File Management Script ---"

# 1. Create the file using touch
echo "Attempting to create file: $FILE_NAME"
touch "$FILE_NAME"

# Check if creation was successful
if [ -f "$FILE_NAME" ]; then
    echo "SUCCESS: File '$FILE_NAME' created."
else
    echo "ERROR: File creation failed."
    exit 1
fi

# 2. Delete the file using rm
echo "Pausing for 1 second before deletion..."
sleep 1

echo "Attempting to delete file: $FILE_NAME"
rm "$FILE_NAME"

# Check if deletion was successful
if [ ! -f "$FILE_NAME" ]; then
    echo "SUCCESS: File '$FILE_NAME' deleted."
else
    echo "ERROR: File deletion failed."
fi
```