Continue

# Shell script example

For all the Linux distributions, the shell script is like a magic wand that automates the process, saves users time, and increases productivity. This shall scripting tutorial will introduce you to the 25 plus shall scripting examples. But before we move on to the topic of shell scripting examples, let’s understand shell script and the actual use cases of shell scripting. What is Shell Script?Well, the shell is a CLI (command line interpreter), which runs in a text window where users can manage and execute shell commands. On the other hand, the process of writing a set of commands to be executed on a Linux system A file that includes such instructions is called a bash script. Uses of Shell ScriptsBelow are some common uses of Shell Script: Task Automation - It can be used to automate repetitive tasks like regular backups and software installation tasks. Customization - One can use shell scripts to design its command line environment and easily perform its task as per needs.File Management - The shell scripts can also be used to manage and manipulate files and directories, such as moving, copying, renaming, or deleting files.Shell Script Examples in Linux1) What does the shebang (#!) at the beginning of a shell script indicate?The shebang (#!) at the beginning of a script indicates the interpreter that should be used to execute the script. It tells the system which shell or interpreter should interpret the script's commands. For example: Suppose we have a script named myscript.sh written in the Bash shell: shebangIn this example: The #!/bin/bash at the beginning of the script indicates that the script should be interpreted using the Bash shell.The echo commands are used to print messages to the terminal.2) How do you run a shell script from the command line?To run a shell script from the command line, we need to follow these steps: Make sure the script file has executable permissions using the chmod command:chmod +x myscript.shExecute the script using its filename:./myscript.shHere you have to replace "myscrtipt.sh" with yors script name. 3) Write a shell script that prints "GeeksforGeeks" to the terminal.Create a script name `myscript.sh` (we are using `vim` editor, you can choose any editor) vim myscript.sh#!/bin/bash# This script prints "GeeksforGeeks" to the terminalecho "GeeksforGeeks" print nameWe make our script executable by using `chmod +x` then execute with `./myscipt.sh` and get our desired output "GeeksforGeeks". 4) Explain the purpose of the echo command in shell scripting.The echo command is used to display text or variables on the terminal. It's commonly used for printing messages, variable values, and generating program output. echo commandIn this example we have execute `echo` on terminal directely , as it works same inside shell script. 5) How can you assign a value to a variable in a shell script?Variables are assigned values using the assignment operator =. For example: #!/bin/bash# Assigning a value to a variablename="Jayesh"age=21echo $name $age Explanation: The name variable is assigned the value "Jayesh".The age variable is assigned the value 21.echo is used to print and `$name` `$age` is used to call the value stored in the variables. 6) Write a shell script that takes a user's name as input and greets them.Create a script name `example.sh`. #!/bin/bash# Ask the user for their nameecho "What's your name?"read name# Greet the userecho "Hello, $name! Nice to meet you." Explanation: #!/bin/bash: This is the shebang line. It tells the system to use the Bash interpreter to execute the script.# Ask the user for their name: This is a comment. It provides context about the upcoming code. Comments are ignored by the interpreter.echo "What's your name?": The echo command is used to display the text in double quotes on the terminal.read name: The read command waits for the user to input text and stores it in the variable name.echo "Hello, $name! Nice to meet you.": This line uses the echo command to print a greeting message that includes the value of the name variable, which was collected from the user's input. 7) How do you add comments to a shell script? Comments in shell scripting are used to provide explanations or context to the code. They are ignored by the interpreter and are only meant for humans reading the script. You can add comments using the # symbol. #!/bin/bash# This is a comment explaining the purpose of the scriptecho "gfg" 8) Create a shell script that checks if a file exists in the current directory.Here's a script that checks if a file named "example.txt" exists in the current directory: #!/bin/bashfile="example.txt"# Check if the file existsif [ -e "$file" ]; thenecho "File exists: $file"elseecho "File not found: $file"fi Explanation: #!/bin/bash: This is the shebang line that specifies the interpreter (/bin/bash) to be used for running the script.file="example.txt": This line defines the variable file and assigns the value "example.txt" to it. You can replace this with the name of the file you want to check for.if [ -e "$file" ]; then: This line starts an if statement. The condition [ -e "$file" ] checks if the file specified by the value of the file variable exists. The -e flag is used to check for file existence.echo "File exists: $file": If the condition is true (i.e., the file exists), this line prints a message indicating that the file exists, along with the file's name.else: If the condition is false (i.e., the file doesn't exist), the script executes the code under the else branch.echo "File not found: $file": This line prints an error message indicating that the specified file was not found, along with the file's name.fi: This line marks the end of the if statement.Finding file9) What is the difference between single quotes (') and double quotes (") in shell scripting?Single quotes (') and double quotes (") are used to enclose strings in shell scripting, but they have different behaviors: Single quotes: Everything between single quotes is treated as a literal string. Variable names and most special characters are not expanded.Double quotes: Variables and certain special characters within double quotes are expanded. The contents are subject to variable substitution and command substitution.#!/bin/bash abcd="Hello"echo '$abcd' # Output: $abcdecho "$abcd" # Output: Hello 10) How can you use command-line arguments in a shell script?Command-line arguments are values provided to a script when it's executed. They can be accessed within the script using special variables like $1, $2, etc., where $1 represents the first argument, $2 represents the second argument, and so on. For Example: If our script name in `example.sh` #!/bin/bash echo "Script name: $0"echo "First argument: $1"echo "Second argument: $2" If we run the script with `.example.sh hello_1 hello_2`, it will output: cli arguments11) How do you use the for loop to iterate through a list of values?Create a script name `example.sh`. #!/bin/bash fruits=("apple" "banana" "cherry" "date")for fruit in "${fruits[@]}"; doecho "Current fruit: $fruit"done Explanation: `fruits=` line creates an array named fruits with four elements: "apple", "banana", "cherry", and "date". for fruit in "${fruits[@]}"; do: This line starts a for loop. Here's what each part means:for fruit: This declares a loop variable called fruit. In each iteration of the loop, fruit will hold the value of the current element from the fruits array."${fruits[@]}": This is an array expansion that takes all the elements from the fruits array. The "${...}" syntax ensures that each element is treated as a separate item.do: This keyword marks the beginning of the loop body.echo "Current fruit: $fruit": Inside the loop, this line uses the echo command to display the current value of the loop variable fruit. It prints a message like "Current fruit: apple" for each fruit in the array.done: This keyword marks the end of the loop body. It tells the script that the loop has finished.for loop12) Write a shell script that calculates the sum of integers from 1 to N using a loop.Create a script name `example.sh`. #!/bin/bash echo "Enter a number (N):"read Nsum=0for (( i=1; i