Basic Shell Scripting Worksheet

1. Introduction to Shell Scripting

Objective: Understand the purpose of a shell script and how to create and execute one.

Key Concepts: - A **shell script** is a text file containing a series of commands that are executed by the shell. - Shell scripts can automate repetitive tasks. - Scripts typically use the `.sh` extension.

Commands to Learn: - `#!/bin/bash` – Shebang line to specify the shell to be used. - `chmod +x` – Make the script executable. - `./scriptname.sh` – Execute the script.

Exercise: 1. Open a text editor (e.g., `nano` or `vi`) and create a new file named `firstscript.sh`. 2. Add the following lines to the script: `bash #!/bin/bash echo "Hello, World!"` 3. Save the file and exit the editor. 4. Make the script executable using the command `chmod +x firstscript.sh`. 5. Run the script by typing `./firstscript.sh`. What output do you see?

---

2. Variables

Objective: Learn how to use variables in a shell script.

Key Concepts: - Variables store data that can be used and modified throughout the script. - Variables are defined as `variablename=value` with no spaces around the `=` sign. - Variables are accessed using the `$` symbol (e.g., `$variablename`).

Exercise: 1. Create a new script called `variablescript.sh`. 2. Add the following lines to your script: `bash #!/bin/bash name="John" echo "Hello, $name!"` 3. Save and run the script. 4. Modify the script to accept user input for the variable. Use the following code: `bash #!/bin/bash echo "Enter your name:" read name echo "Hello, $name!"` 5. Test the script by entering different names.

---

3. Conditional Statements (if-else)

Objective: Learn how to use conditional statements to control the flow of the script.

Key Concepts: - `if-else` statements allow the script to execute different code depending on conditions. - Conditions are placed inside square brackets `[ ]` with spaces around them. - `-eq`, `-ne`, `-lt`, `-gt`, `-le`, and `-ge` are used for numerical comparisons. - `=` and `!=` are used for string comparisons.

Exercise: 1. Create a script called `checknumber.sh`. 2. Add the following lines to your script: `bash #!/bin/bash echo "Enter a number:" read number if [ $number -gt 10 ]; then echo "The number is greater than 10." else echo "The number is 10 or less." fi` 3. Save and run the script. 4. Modify the script to check if the number is even or odd. Use the following code: `bash if [ $(($number % 2)) -eq 0 ]; then echo "The number is even." else echo "The number is odd." fi`

---

4. Loops

Objective: Use loops to repeat a set of commands.

Key Concepts: - `for`, `while`, and `until` loops allow a block of code to repeat. - `for` loops iterate over a list of values. - `while` loops run as long as a condition is true.

Exercise: 1. Create a script called `loopscript.sh`. 2. Add the following `for` loop to your script: `bash #!/bin/bash for i in 1 2 3 4 5 do echo "Loop iteration: $i" done` 3. Save and run the script. 4. Modify the script to use a `while` loop to count down from 5 to 1. Use the following code: `bash i=5 while [ $i -gt 0 ] do echo "Countdown: $i" i=$(($i - 1)) done`

---

5. Functions

Objective: Learn how to define and use functions in a shell script.

Key Concepts: - Functions allow you to group a set of commands that can be reused. - A function is defined as: `bash functionname() { commands }`

Exercise: 1. Create a script called `functionscript.sh`. 2. Add the following function to your script: `bash #!/bin/bash greet() { echo "Hello, $1!" } greet "Alice" greet "Bob"` 3. Save and run the script. What happens when you change the argument passed to `greet`? 4. Modify the function to take user input as its argument. Use this code: `bash echo "Enter your name:" read name greet $name`

---

6. Basic Arithmetic

Objective: Perform arithmetic operations within a script.

Key Concepts: - Use `$((expression))` for arithmetic calculations. - Arithmetic operations include `+`, `-`, `*`, `/`, and `%`.

Exercise: 1. Create a script called `arithmeticscript.sh`. 2. Add the following lines to perform basic arithmetic: `bash #!/bin/bash num1=5 num2=3 sum=$(($num1 + $num2)) echo "The sum of $num1 and $num2 is $sum."` 3. Save and run the script. 4. Modify the script to accept user input for the numbers. Use the following code: `bash echo "Enter the first number:" read num1 echo "Enter the second number:" read num2 sum=$(($num1 + $num2)) echo "The sum of $num1 and $num2 is $sum."`

---

7. Input and Output Redirection

Objective: Learn how to redirect input and output in shell scripts.

Key Concepts: - `>` redirects output to a file, overwriting its contents. - `>>` appends output to a file. - `<` redirects input from a file.

Exercise: 1. Create a script called `redirectionscript.sh`. 2. Add the following lines to redirect output to a file: `bash #!/bin/bash echo "This is a test." > output.txt echo "Another line." >> output.txt` 3. Save and run the script, then view the contents of `output.txt` using `cat`. 4. Modify the script to read input from a file using: `bash cat < inputfile.txt`

---

8. Bonus Challenge: Simple Backup Script

Objective: Write a script to automate a file backup.

Exercise: 1. Create a script called `backup.sh`. 2. Add the following code: `bash #!/bin/bash echo "Enter the directory to back up:" read dir tar -czvf backup.tar.gz $dir echo "Backup of $dir completed."` 3. Run the script and verify that a compressed archive (`backup.tar.gz`) has been created for the specified directory.
