[****](https://linuxsimply.com/advanced-shell-script-examples/)

**Advanced Shell Script With Examples**

**Bash Scripting** offers the concepts of string, array, and loops for achieving advanced programming goals. In this article, I will explore concepts and tools of the **advanced shell script** that will elevate your shell scripting skills. I will equip you with the knowledge to tackle complex tasks.

[**Strings in Shell Scripting 3**](#_qculmtv30f9h)

[Example 1: Find the Length of a String 4](#_c9oaoyt1smwj)

[Example 2: Check if Two Strings are Equal 4](#_ly8m01abhh73)

[Example 3: Convert All Uppercase Letters in a String to Lowercase 4](#_kkkoxxrbcyqs)

[Example 4: Remove All Whitespace from a String 5](#_mt0uyf9pypy5)

[Example 5: Reverse a String 5](#_chjr6grr47ri)

[Example 6: Reverse a Sentence 5](#_t822nqeenmon)

[Example 7: Capitalize the First Letter of a Word 6](#_7i4gid3i5d99)

[Example 8: Replace a Word in a Sentence 6](#_b22h26phclme)

[**Loops in Shell Scripting 6**](#_40yolrepcqaa)

[Example 1: Print Numbers from 5 to 1 7](#_2ag5586kwy4z)

[Example 2: Print Even Numbers From 1 to 10 7](#_d0mc1f75ea8c)

[Example 3: Print the Multiplication Table of a Number 8](#_oitqmrbrwqac)

[Example 4: Calculate the Sum of Digits of a Given Number 8](#_u4c7lwf2m8r8)

[Example 5: Calculate the Factorial of a Number 9](#_4lcepc9cuedg)

[Example 6: Calculate the Sum of the First “n” Numbers 9](#_5y5o0xdvngh4)

[**Arrays in Shell Scripting 10**](#_4n1yn6bwqzrm)

[Example 1: Find the Smallest and Largest Elements in an Array 10](#_dxo3qrcaq67t)

[Example 2: Sort an Array of Integers in Ascending Order 11](#_ar4spndm516e)

[Example 3: Remove an Element from an Array 11](#_qskzz1d7po3q)

[Example 4: Inserting an Element Into an Array 11](#_11misgy3l61r)

[Example 5: Slicing an Array Using Bash Script 12](#_6yc0kvnnahfq)

[Example 6: Calculate the Average of an Array of Numbers 12](#_jmjxqsgf9ftn)

[Example 7: Find the Length of an Array 13](#_74la3wfzxz9l)

[**Functions in Shell Scripting 13**](#_oq9dbf1wkajq)

[Example 1: Check if a String is a Palindrome 14](#_yz1z2rs7s4xw)

[Example 2: Check if a Number is Prime 14](#_1998dyh579aw)

[Example 3: Convert Fahrenheit to Celsius 15](#_ilodjor0plru)

[Example 4: Calculate the Area of a Rectangle 15](#_qgrrg057titq)

[Example 5: Calculate the Area of a Circle 16](#_xwljwzhbg1ga)

[Example 6: Grading System 16](#_6i3irivry4vi)

[**Task-Specific Bash Scripts 17**](#_cr914xcttwe)

[Regular Expression Based Scripts 17](#_jq8rajodj9m0)

[1. Search for a Pattern inside a File 17](#_amdmpx74n0lr)

[2. Replace a Pattern in a Fille 18](#_n5kmoidh8bl9)

[File Operations with Shell Scripts 18](#_pb10rsw1nvc1)

[3. Take Multiple Filenames and Prints their Contents 18](#_kjbo9i2oiods)

[4. Copy a File to a New Location 19](#_pmxvxzmb0a8h)

[5. Create a New File and Write Text Inside 20](#_g0jbzv69pbda)

[6. Compare the Contents of Two Given Files 20](#_zg3dee50hgg)

[7. Delete a Given File If It Exists 21](#_kh7w1dzgjpb0)

[8. Renames a File from Script 21](#_2cvasoq5w22a)

[File Permission Based Shell Scripts 22](#_3zc4nrowci6x)

[9. Check the Permissions of a file 22](#_1rxfrcou42ue)

[10. Sets the Permissions of a Directory for the Owner 22](#_rmdu3d8sx167)

[11. Change the File Owner 23](#_9pggqoi3fne)

[12. File Permissions: Change the Overall Permissions of a File 23](#_mpdydymf1ql9)

[Network Connection Based Shell Scripts 24](#_4o9ts5a17pta)

[13. Check a Remote Host for its Availability 24](#_9yvhl1njy15k)

[14. Test if a Remote Port is Open 25](#_etgfkcc78kkr)

[15. Checking Network Connectivity 26](#_rmb3o1w2q4re)

[16. Automating Network Configuration 26](#_21zbq2othtrt)

[17. Check if a Process is Running 27](#_tsmj91fqm7wt)

[Process Management Based Shell Scripts 28](#_or25qp5me3xe)

[18. Start a Process if It's Not Already Running 28](#_tp7wslv5huys)

[19. Stop a Process 28](#_c4w8axroibl0)

[20. Restart a Process 29](#_80u5d4lzzwh9)

[21. Monitor a Process and Restart It If Crashes 29](#_elzgz2z5r3fj)

[22. Display the Top 10 CPU-Consuming Processes 30](#_cdblsofi40tr)

[23. Display the Top 10 Memory-Consuming Processes 31](#_s765j6lvpu7x)

[24. Kill Processes of a Specific User 31](#_ohz1qdv98l58)

[25. Kill All Processes That are Consuming More Than a Certain Amount of CPU 32](#_yvfv0oy1ucbh)

[26. Kill All Processes That are Consuming More Than a Certain Amount of Memory 32](#_3dpuk8pcuh7p)

[System Information Based Shell Scripts 33](#_z50qqxhdef2z)

[27. Check the Number of Logged-in Users 33](#_945eaz4yo7g1)

[28. Check the Operating System Information 33](#_hoic961e66nz)

[29. Check the System’s Memory Usage 34](#_1db13h1goh2x)

[30. Check the System’s Disk Usage 34](#_q8ht3kyzq0i1)

[31. Check the System’s Network Information 34](#_mr6nta5pp7l1)

[32. Check the Uptime 35](#_dhs836shjh3f)

[33. Check the System Load Average 35](#_axz1kf3nirjy)

[34. Check the System Architecture 36](#_np5596e3s5mt)

[35. Count the Number of Files in the System 36](#_6ah2opel4b4f)

[Advanced Tasks with Shell Scripts 36](#_ybptx59vh86l)

[36. Automated Backup 36](#_y2b3x4hq37kh)

[37. Generate Alert if Disk Space Usage Goes Over a Threshold 37](#_qssvl8j9amy7)

[38. Create a New User and Add to Sudo Group 37](#_85fiqyfo3oa)

[39. Monitor Network Traffic 38](#_wixral1l2lra)

[40. Monitor CPU and Memory Usage 39](#_mxgut23d6y7l)

[41. Creating a Script and Adding It to PATH 39](#_d6pqgw3xu1z)

[42. Running a Command At Regular Intervals 40](#_1l5zhuns9j2v)

[43. Downloading Files from a List of URLs 41](#_qwemrxqvajjw)

[44. Organizes Files in a Directory Based on Their File Types 42](#_2o91wj6jg6dd)

[**Conclusion 43**](#_ligq5vkfdet8)

## **Strings in Shell Scripting**

Similar to all the programming languages **Bash** also has the **String** data type which indicates a set of characters. To denote inputs as **String** you must enclose it within the **double quotation(“”)**. Values passed as strings are considered as text rather than a number or variable. Therefore, **Bash** provides an additional set of operators for the **String** data type.

**The syntax for Strings in Shell Scripting is given below:**

| STRING\_NAME="STRING\_VALUE" |
| --- |

**The String operators in Shell Scripting are as follows:**

| **String Operators** |
| --- |
| **<** (Less than) | **==** (Equal) | **+=** (Concatenation) |
| **>** (Greater than) | **!=** (Not equal) |   |

### **Example 1: Find the Length of a String**

You can simply use the **${#STRING}** to find the length of a string:

**Code:**

| #!/bin/bash str="My name is Tom!"len=${#str}echo "The length of the string is: $len" |
| --- |

**Output:**

| **The length of the string is: 15** |
| --- |

### **Example 2: Check if Two Strings are Equal**

Check whether two strings are same or not using the **== (Equal)** operator inside **if** condition:

**Code:**

| #!/bin/bashstring1="hello"string2="world"if [ "$string1" == "$string2" ]; then echo "The strings are equal."else echo "The strings are not equal."fi |
| --- |

**Output:**

| **The strings are not equal.** |
| --- |

### **Example 3: Convert All Uppercase Letters in a String to Lowercase**

Here is a bash script for converting all upper case letters in a string to lower case letters that use the **tr** command with the **[:upper:]** and **[:lower:]** classes for conversion:

**Code:**

| #!/bin/bashread -p "Enter a string: " strecho "Converted String:" $str | tr '[:upper:]' '[:lower:]' |
| --- |

**Output:**

| **Enter a string: ABCDefghconverted string: abcdefgh** |
| --- |

### **Example 4: Remove All Whitespace from a String**

For removing white spaces from a string simply use the **${STRING// /}**:

**Code:**

| **#!/bin/bash str=" Hello from Linuxsimply ! ! "str=${str// /}echo "The resultant string: $str"** |
| --- |

**Output:**

| **The resultant string: HellofromLinuxsimply!!** |
| --- |

### **Example 5: Reverse a String**

To reverse a string use the **rev** command with **echo** and **Pipe(|)**:

**Code:**

| #!/bin/bash str="Linuxsimply"str=$(echo "$str" | rev)echo "The reversed string: $str" |
| --- |

**Output:**

| **The reversed string: ylpmisxuniL** |
| --- |

### **Example 6: Reverse a Sentence**

You can reverse a sentence by reversing the order of words with the **awk** command:

**Code:**

| #!/bin/bashsentence="Hello from LinuxsimplY!!"r\_sentence=$(echo "$sentence" | awk '{ for(i=NF;i>0;i--) printf("%s ",$i); print "" }')echo "The reversed sentence is: $r\_sentence" |
| --- |

**Output:**

| **The reversed sentence is: LinuxsimplY!! from Hello** |
| --- |

### **Example 7: Capitalize the First Letter of a Word**

For capitalizing only the first letter of a word, cut out the first letter to convert it and then concatenate it with the rest of the string:

**Code:**

| #!/bin/bashstr="linuxsimply!!"cap\_str=$(echo "${str:0:1}" | tr '[:lower:]' '[:upper:]')${str:1}echo "The capitalized word is: $cap\_str" |
| --- |

**Output:**

The capitalized word is: Linuxsimply!!

### **Example 8: Replace a Word in a Sentence**

You can replace the first occurrence of a word in a string with a given word using the **$(../../..)**:

**Code:**

| #!/bin/bashread -p "Enter a sentence: " str1read -p "Enter the word to be replaced: " str2read -p "Enter the new word: " str3echo "Modified sentence: ${str1/$str2/$str3}" |
| --- |

**Output:**

| **Enter a sentence: I love LinuxEnter the word to be replaced: LinuxEnter the new word: LinuxsimplyModified sentence: I love Linuxsimply** |
| --- |

## **Loops in Shell Scripting**

Loops are introduced in programming languages to run tasks in a repetitive manner. It iterates a set of statements within a limit depending on conditions. **Bash Scripting** provides **three** types of loops for statement iterations. These are the **for loop**, the **while loop,** and the **until loop**. Syntaxes for each of the loops are listed below.

**Syntaxes for Loops in Bash Scripting:**

| **for** | **while** | **until** |
| --- | --- | --- |
| **for item in item1 item2 ... itemN**OR,**for (( i=initial\_val; i<=terminating\_val; i++ ))****do** *#code to execute***done** | **while [ condition ]****do** *#code to execute***done** | **until [ condition ]****do** *#code to execute***done** |

### **Example 1: Print Numbers from 5 to 1**

You can use the **“until”** loop in bash to print a number sequence. In this case, specify the condition to stop the loop inside “**until [ ]**”:

**Code:**

| #!/bin/bash n=5until [ $n == 0 ]do echo $n n=$((n-1))done |
| --- |

**Output:**

| **54321** |
| --- |

### **Example 2: Print Even Numbers From 1 to 10**

To print the even number in a range, check the even number condition inside the for loop before printing the number:

**Code:**

| #!/bin/bashfor (( i=1; i<=10; i++ ))do if [ $((i%2)) == 0 ] then echo $i fidone |
| --- |

**Output:**

| **246810** |
| --- |

### **Example 3: Print the Multiplication Table of a Number**

Use the simple **echo** command inside a “**for**” loop to display the Multiplication Table of a number:

**Code:**

| #!/bin/bashread -p "Enter a number: " numfor (( i=1; i<=10; i++ ))do echo "$num x $i = $((num\*i))"done |
| --- |

**Output:**

| **Enter a number: 1212 x 1 = 1212 x 2 = 2412 x 3 = 3612 x 4 = 4812 x 5 = 6012 x 6 = 7212 x 7 = 8412 x 8 = 9612 x 9 = 10812 x 10 = 120** |
| --- |

### **Example 4: Calculate the Sum of Digits of a Given Number**

For calculating the sum of digits of a given number, extract each digit using “**%**” operator and store the summation in a fixed variable using the loop:

**Code:**

| #!/bin/bashread -p "Enter a number: " numsum=0while [ $num -gt 0 ]do dig=$((num%10)) sum=$((sum+dig)) num=$((num/10))doneecho "The sum of digits of the given number: $sum" |
| --- |

**Output:**

| **Enter a number: 1567The sum of digits of the given number: 19** |
| --- |

### **Example 5: Calculate the Factorial of a Number**

Calculate the factorial of a number by running multiplications inside a “**for**” loop:

**Code:**

| #!/bin/bashread -p "Enter a number: " numtemp=1for (( i=1; i<=$num; i++ ))do temp=$((temp\*i))doneecho "The factorial of $num is: $temp" |
| --- |

**Output:**

| **Enter a number: 6The factorial of 6 is: 720** |
| --- |

### **Example 6: Calculate the Sum of the First “n” Numbers**

To calculate the sum of the first n numbers run a for loop and addition operation till n:

**Code:**

| #!/bin/bashread -p "Enter a number: " numsum=0for (( i=1; i<=$num; i++ ))do sum=$((sum + i))doneecho "Sum of first $num numbers: $sum" |
| --- |

**Output:**

| Enter a number: 100Sum of first 100 numbers: 5050 |
| --- |

## **Arrays in Shell Scripting**

Arrays, in general, are a set or collection of data of similar types. Bash arrays differ from arrays in other programming languages since bash does not necessarily differentiate between the numbers or string data types. Therefore, an array in **bash** can store both numbers and strings at the same time. Follow the examples below to learn more about array operations in bash scripting.

### **Example 1: Find the Smallest and Largest Elements in an Array**

For finding the smallest and largest element in a given array, first initialize a small and a large number. Then compare the array elements with these numbers inside any loop:

**Code:**

| #!/bin/basharr=(24 27 84 11 99)echo "Given array: ${arr[\*]}"s=100000l=0for num in "${arr[@]}"do if [ $num -lt $s ] then s=$num fi if [ $num -gt $l ] then l=$num fidoneecho "The smallest element: $s"echo "The largest: $l" |
| --- |

**Output:**

| **Given array: 24 27 84 11 99The smallest element: 11The largest: 99** |
| --- |

### **Example 2: Sort an Array of Integers in Ascending Order**

You can sort an array of integers by converting it into a list of integers using “**tr ‘\n’**”. The list of integers is sorted with the “**sort -n**” command and then converted back into an array:

**Code:**

| #!/bin/basharr=(24 27 84 11 99)echo "Given array: ${arr[\*]}"arr=($(echo "${arr[\*]}" | tr ' ' '\n' | sort -n | tr '\n' ' '))echo "Sorted array: ${arr[\*]}" |
| --- |

**Output:**

| **Given array: 24 27 84 11 99Enter an element to remove: 11Resultant array: 24 27 84 99** |
| --- |

### **Example 3: Remove an Element from an Array**

In bash, you can simply remove an element from an array using the pattern substitution concept. The syntax **${arr[@]/$val}** contains all the elements of the original array “**arr**” except for any occurrences of the value **$val**:

**Code:**

| #!/bin/basharr=(24 27 84 11 99)echo "Given array: ${arr[\*]}"read -p "Enter an element to remove: " valarr=("${arr[@]/$val}")echo "Resultant array: ${arr[\*]}" |
| --- |

**Output:**

| **Given array: 24 27 84 11 99Enter an element to remove: 11Resultant array: 24 27 84 99** |
| --- |

### **Example 4: Inserting an Element Into an Array**

For inserting an element into an array, split the array in the given index and insert the element:

**Code:**

| #!/bin/basharr=(24 27 84 11 99)echo "Given array: ${arr[\*]}"read -p "Enter an element to insert: " new\_valread -p "Enter the index to insert the element: " indexarr=("${arr[@]:0:$index}" "$new\_val" "${arr[@]:$index}")echo "The updated array: ${arr[@]}" |
| --- |

**Output:**

| **iven array: 24 27 84 11 99Enter an element to insert: 100Enter the index to insert the element: 3The updated array: 24 27 84 100 11 99** |
| --- |

### **Example 5: Slicing an Array Using Bash Script**

Slice an array in Bash by placing the indices to slice inside the **${arr[@]:$index1:$index2}** pattern:

**Code:**

| #!/bin/basharr=(24 27 84 11 99)echo "Given array: ${arr[\*]}"read -p "Enter 1st index of slice: " index1 read -p "Enter 2nd index of slice: " index2sliced\_arr=("${arr[@]:$index1:$index2}")echo "The sliced array: ${sliced\_arr[@]}" |
| --- |

**Output:**

| **Given array: 24 27 84 11 99Enter 1st index of slice: 1Enter 2nd index of slice: 3The sliced array: 27 84 11** |
| --- |

### **Example 6: Calculate the Average of an Array of Numbers**

Find the sum of array elements using a “**for**” loop and divide it by the number of elements i.e. **${#arr[@]}**:

**Code:**

| #!/bin/bashecho "Enter an array of numbers (separated by space):"read -a arrsum=0for i in "${arr[@]}"do sum=$((sum+i))doneavg=$((sum/${#arr[@]}))echo "Average of the array elements: $avg" |
| --- |

**Output:**

| **Enter an array of numbers (separated by space):23 45 11 99 100Average of the array elements: 55** |
| --- |

### **Example 7: Find the Length of an Array**

To find the length of an array simply use the syntax: **${#arr[@]}**:

**Code:**

| #!/bin/basharr=(24 27 84 11 99)echo "Given array: ${arr[\*]}"len=${#arr[@]}echo "The length of the array: $len" |
| --- |

**Output:**

| Given array: 24 27 84 11 99The length of the array: 5 |
| --- |

## **Functions in Shell Scripting**

Functions are one of the popular concepts of programming languages. It is a piece of code that can be called and executed as many times as you want. Thus, functions offer efficiency, code optimization, and minimization. Functions in **Bash** work in a similar way as functions in other programming languages. However, there are some rules and syntaxes that you must follow while using them in your script.

**The syntax for Function in Shell Scripting:**

| FUNCTION\_NAME () {  #codes to execute  }  |
| --- |

**Or,**

| FUNCTION\_NAME () { #code to execute; }  |
| --- |

**The rules for Function in Shell Scripting are as follows:**

| ● Functions must be defined before using/calling them.● You may pass arguments to functions while calling them.● To access arguments inside the function, use **$1, $2, $3** **…** and so on according to the number and sequence of arguments passed.● The scope of the variables declared inside a function remains within the function. |
| --- |

### **Example 1: Check if a String is a Palindrome**

Write the code to check a palindrome inside the function “**Palindrome()**” and call it by passing the desired string:

**Code:**

| #!/bin/bashPalindrome () { s=$1 if [ "$(echo $s | rev)" == "$str" ] then echo "The string is a Palindrome" else echo "The string is not a palindrome" fi}read -p "Enter a string: " strPalindrome "$str" |
| --- |

**Output:**

| **Enter a string: wowThe string is a Palindrome** |
| --- |

### **Example 2: Check if a Number is Prime**

Create the “**Prime()**” function that returns whether the parameter passed is prime or not:

**Code:**

| #!/bin/bashPrime () { num=$1 if [ $num -lt 2 ] then echo "The number $num is Not Prime" return fi for (( i=2; i<=$num/2; i++ )) do if [ $((num%i)) -eq 0 ] then echo "The number $num is Not Prime" return fi done echo "The number $num is Prime"}read -p "Enter a number: " num Prime "$num" |
| --- |

**Output:**

| **Enter a number: 2The number 2 is Prime** |
| --- |

### **Example 3: Convert Fahrenheit to Celsius**

Here, the function “**Celsius()**” runs the necessary formula on the passed temperature value in Farenheit to convert it into Celsius:

**Code:**

| #!/bin/bashCelsius () { f=$1 c=$((($f-32)\*5/9)) echo "Temperature in Celsius = $c°C"}read -p "Enter temperature in Fahrenheit:" fCelsius $f |
| --- |

**Output:**

| **Enter temperature in Fahrenheit:100Temperature in Celsius = 37°C** |
| --- |

### **Example 4: Calculate the Area of a Rectangle**

Write the formula to calculate the area of a rectangle inside the function “**Area()**” and call it by passing the height and width:

**Code:**

| #!/bin/bashArea() { width=$1 height=$2 area=$(($width \* $height)) echo "Area of the rectangle is: $area"}read -p "Enter height and width of the ractangle:" h wArea $h $w |
| --- |

**Output:**

| **Enter height and width of the ractangle:10 4"Area of the rectangle is: 40"** |
| --- |

### **Example 5: Calculate the Area of a Circle**

Write the formula to calculate the area of a circle inside the function “**Area()**” and call it by passing the given radius:

**Code:**

| #!/bin/bashArea () { radius=$1 area=$(echo "scale=2; 3.14 \* $radius \* $radius" | bc) echo "Area of a circle with radius $radius is $area."}read -p "Enter radius of the circle:" rArea $r |
| --- |

**Output:**

| **Enter radius of the circle:4Area of a circle with radius 4 is 50.24.** |
| --- |

### **Example 6: Grading System**

The function “**Grade()**” runs the necessary conditions to divide the number ranges into grades and returns the resultant grade:

**Code:**

| #!/bin/bashGrade() { score=$1 if (( $score >= 80 )); then grade="A+" elif (( $score >= 70 )); then grade="A" elif (( $score >= 60 )); then grade="B" elif (( $score >= 50 )); then grade="C" elif (( $score >= 40 )); then grade="D" else grade="F" fi echo "The grade for mark $s is $grade"}read -p "Enter a score between 1-100:" sGrade $s |
| --- |

**Output:**

| Enter a score between 1-100:76"The grade for mark 76 is A" |
| --- |

## **Task-Specific Bash Scripts**

In addition to the conceptual bash scripts, in this section, you will find some task-specific script examples. These scripts are mostly related to the regular process that you run on your system. Hence, explore the examples below to get more hands-on experience with **Shell Scripting**.

### **Regular Expression Based Scripts**

#### **1. Search for a Pattern inside a File**

The script given below will take a filename and a pattern as user input and search it within the file. If the pattern is found then the lines having the pattern will be displayed on the screen along with line numbers. Otherwise, it will print a message saying the pattern did not match:

**Code:**

| #!/bin/bashread -p "Enter filename: " filenameread -p "Enter a pattern to search for: " patterngrep -w -n $pattern $filenameif [ $? -eq 1 ]; thenecho "Pattern did not match."fi |
| --- |

**Output:**

| **Enter filename: poem.txtEnter a pattern to search for: daffodils4:A host, of golden daffodils;27:And dances with the daffodils.** |
| --- |

#### **2. Replace a Pattern in a Fille**

The following script will take a file name and a pattern from the user to replace it with a new pattern. Finally, it will display the updated lines on the terminal. If the pattern to replace does not exist, then it will show an error message:

**Code:**

| #!/bin/bashread -p "Enter filename: " filenameread -p "Enter a pattern to replace: " patternread -p "Enter new pattern: " new\_patterngrep -q $pattern $filenameif [ $? -eq 0 ]; thensed -i "s/$pattern/$new\_pattern/g" $filenameecho "Updated Lines: "grep -w -n $new\_pattern $filenameelseecho "Error! Pattern did not match."fi |
| --- |

**Output:**

| Enter filename: poem.txtEnter a pattern to replace: daffodilsEnter new pattern: dandelionsUpdated Lines: 4:A host, of golden dandelions;27:And dances with the dandelions. |
| --- |

### **File Operations with Shell Scripts**

#### **3. Take Multiple Filenames and Prints their Contents**

The below script is for reading the contents of multiple files. It will take the file names as user input and display their contents on the screen. If any filename does not exist, it will show a separate error message for that file:

**Code:**

| #!/bin/bashread -p "Enter the file names: " filesIFS=' ' read -ra array <<< "$files"for file in "${array[@]}"doif [ -e "$file" ]; then echo "Contents of $file:" cat "$file" else echo "Error: $file does not exist"fidone |
| --- |

**Output:**

| **Enter the file names: message.txt passage.txtContents of message.txt:"Merry Christmas! May your happiness be large and your bills be small."Contents of passage.txt:The students told the headmaster that they wanted to celebrate the victory of the National Debate Competition.** |
| --- |

#### **4. Copy a File to a New Location**

You can copy a file to another location using the bash script below. It will read the filename and destination path from the terminal and copy the file if it exists in the current directory. If the file is not there, the script will return an error message.

**Code:**

| #!/bin/bashread -p "Enter the file name: " fileread -p "Enter destination path:" destif [ -e "$file" ]; then cp $file $dest file\_location=$(readlink -f $dest) echo "A copy of $file is now located att: $file\_location" else echo "Error: $file does not exist"fi |
| --- |

**Output:**

| **Enter the file name: poem.txtEnter destination path:/home/susmit/DocumentsA copy of poem.txt is now located at: /home/susmit/Documents** |
| --- |

#### **5. Create a New File and Write Text Inside**

The script given below is for creating a new file and writing text inside the file. You will be able to write into the file from the command line. Upon completion, it will show a message saying the file has been created.

**Code:**

| #!/bin/bashread -p "Enter the file name: " fileecho "Enter text to write:"read textecho "$text" > "$file"echo "-----------------------------------"echo "The File $file is created!" |
| --- |

**Output:**

| **Enter the file name: text\_file1.txtEnter text to write:In English, there are three articles: a, an, and the. Articles are used before nouns or noun equivalents and are a type of adjective. The definite article (the) is used before a noun to indicate that the identity of the noun is known to the reader. -----------------------------------The File text\_file1.txt is created!** |
| --- |

#### **6. Compare the Contents of Two Given Files**

The following bash script takes two file names as user input and compares there contents. If one or either of the files does not exist in the current directory it shows an error to the user. Otherwise prints the result if the files are identical or not.

**Code:**

| #!/bin/bashread -p "Enter the 1st file name: " file1read -p "Enter the 2nd file name: " file2if [ ! -f $file1 ] || [ ! -f $file2 ]then echo "Error! One of the files does not exists." exit 1fiif cmp -s "$file1" "$file2"then echo "The Files $file1 and $file2 are identical."else echo "The Files $file1 and $file2 are different."fi |
| --- |

**Output:**

| **Enter the 1st file name: article1.txtEnter the 2nd file name: text\_file1.txtThe Files article1.txt and text\_file1.txt are identical.** |
| --- |

#### **7. Delete a Given File If It Exists**

This is a script for checking a file's existence before running deleting the file. The script will take the file’s name from the user and delete it if it is found in the current directory. Otherwise, it will display an error.

**Code:**

| #!/bin/bashread -p "Enter the file name for deletion: " fileif [ -f $file ]then rm $file echo "The file $file deleted successfully!"else echo "Error! The file $file does not exist."fi |
| --- |

**Output:**

| **Enter the file name for deletion: article1.txtThe file article1.txt deleted successfully!** |
| --- |

#### **8. Renames a File from Script**

You can rename an existing file using the script below. All you have to do is enter the old filename and the new filename. The script will rename the file if it is available in the directory. If the file is not in the path, then it will display an error message.

**Code:**

| #!/bin/bashread -p "Enter the file name: " fileread -p "Enter new file name: " new\_fileif [ -f $file ]then mv "$file" "$new\_file" echo "The file $file has been renamed as $new\_file!"else echo "Error! The file $file does not exist."fi |
| --- |

**Output:**

| Enter the file name: poem.txtEnter new file name: daffodils.txtThe file poem.txt has been renamed as daffodils.txt! |
| --- |

### **File Permission Based Shell Scripts**

#### **9. Check the Permissions of a file**

The script below checks permissions for the given filename and lists the active permissions of the current user. If there does not exist any file of the input file name, then it displays an error message.

**Code:**

| #!/bin/bashread -p "Enter the file name: " fileif [ -f $file ]; thenif [ -r "$file" ]; then echo "Readable"fiif [ -w "$file" ]; then echo "Writable"fiif [ -x "$file" ]; then echo "Executable"fielse echo "Error! The file $file does not exist."fi |
| --- |

**Output:**

| **Enter the file name: daffodils.txtReadableWritable** |
| --- |

#### **10. Sets the Permissions of a Directory for the Owner**

The following script the give current user read, write, and execute permissions of a directory. The directory name is taken as user input and if the directory does not exist, it displays an error message.

**Code:**

| #!/bin/bashread -p "Enter the directory name: " dirif [ -d $dir ]; then chmod u+rwx $dir echo "Directory permissions have been updated!"else echo "Error! The directory $dir does not exist."fi |
| --- |

**Output:**

| **Enter the file name: daffodils.txtEnter file owner name: tom[sudo] password for susmit: Changed file owner to tom!** |
| --- |

#### **11. Change the File Owner**

The script here changes the owner of a file if the file exists in the directory. Since changing ownership requires administrator permissions, you will need to give the **sudo** password while running the script. Upon completion of the task, the script will show a success message.

**Code:**

| #!/bin/bashread -p "Enter the file name: " fileread -p "Enter file owner name: " ownerif [ -f $file ]; then sudo chown $owner $file echo "Changed file owner to $owner!"else echo "Error! The file $file does not exist."fi |
| --- |

**Output:**

| **Enter the file name: daffodils.txtEnter file owner name: tom[sudo] password for susmit: Changed file owner to tom!** |
| --- |

#### **12. File Permissions: Change the Overall Permissions of a File**

You can change the permissions of an existing file using the script below. All you have to do is enter the filename, the permissions in **absolute mode,** and the **sudo** password to activate administrative privileges. The script will update the file permissions if it is available in the directory. If the file is not in the path, then it will display an error message.

**Code:**

| #!/bin/bashread -p "Enter the file name: " fileread -p "Enter new permissions in Absolute Mode: " permissionsif [ -f $file ]; then sudo chmod $permissions $file echo "Permissions for the file $file has been changed!"else echo "Error! The file $file does not exist."fi |
| --- |

**Output:**

| **Enter remote host IP address:192.168.0.6PING 192.168.0.6 (192.168.0.6) 56(84) bytes of data.64 bytes from 192.168.0.6: icmp\_seq=1 ttl=64 time=4.10 ms--- 192.168.0.6 ping statistics ---1 packets transmitted, 1 received, 0% packet loss, time 0msrtt min/avg/max/mdev = 4.095/4.095/4.095/0.000 ms-----------------------Host is up!-----------------------** |
| --- |

### **Network Connection Based Shell Scripts**

#### **13. Check a Remote Host for its Availability**

The following script checks the network status of a remote host. You will need to enter the host IP address as input and it will return a message saying if the host is up or down.

**Code:**

| #!/bin/bashread -p "Enter remote host IP address:" ipping -c 1 $ipif [ $? -eq 0 ]then echo "-----------------------" echo "Host is up!" echo "-----------------------"else echo "-----------------------" echo "Host is down!" echo "-----------------------"fi |
| --- |

**Output:**

| Enter remote host IP address:192.168.0.6PING 192.168.0.6 (192.168.0.6) 56(84) bytes of data.64 bytes from 192.168.0.6: icmp\_seq=1 ttl=64 time=4.10 ms--- 192.168.0.6 ping statistics ---1 packets transmitted, 1 received, 0% packet loss, time 0msrtt min/avg/max/mdev = 4.095/4.095/4.095/0.000 ms-----------------------Host is up!----------------------- |
| --- |

#### **14. Test if a Remote Port is Open**

The script below checks the network connection in a system port. It takes a host address and port number as the input. If the connection to the host through the port number is successful then it verifies that the port is open. Otherwise, it returns a message saying the port is closed.

**Code:**

| #!/bin/bashread -p "Enter host address:" HOSTread -p "Enter port number:" PORTnc -z -v -w5 "$HOST" "$PORT"if [ $? -eq 0 ]; then echo "----------------------------------------------" echo "Port $PORT on $HOST is open" echo "----------------------------------------------"else echo "----------------------------------------------" echo "Port $PORT on $HOST is closed" echo "----------------------------------------------"fi |
| --- |

**Output:**

| **Enter host address:192.168.0.107Enter port number:80Connection to 192.168.0.107 80 port [tcp/http-alt] succeeded!----------------------------------------------Port 80 on 192.168.0.107 is open----------------------------------------------** |
| --- |

#### **15. Checking Network Connectivity**

The below script checks network connectivity to a remote host via the internet. If there is a successful connection then it returns the status “internet connection is up”. Otherwise, returns “Internet connection is down”.

**Code:**

| #!/bin/bashread -p "Enter a host address:" HOSTif ping -q -c 1 -W 1 $HOST >/dev/null; then echo "----------------------------------------------" echo "Internet connection is up" echo "----------------------------------------------"else echo "----------------------------------------------" echo "Internet connection is down" echo "----------------------------------------------"fi |
| --- |

**Output:**

| **Enter a host address:192.168.0.107----------------------------------------------Internet connection is up----------------------------------------------** |
| --- |

#### **16. Automating Network Configuration**

The following bash script configures a network IP address and subnet mask. Upon configuration, it sets up the gateway and DNS server at the given addresses. All four IP addresses are passed as user input. It will return an error message if it is unsuccessful at running any of the commands.

**Code:**

| **#!/bin/bashecho "Enter network configuration variables:"read -p "Enter an IP address: " ip\_addrread -p "Enter a subnet mask: " subnet\_maskread -p "Enter a Gateway address: " gatewayread -p "Enter a DNS address: " dns# Configure network interfacesudo ifconfig eth0 $ip\_addr netmask $subnet\_mask upif [ $? -eq 0 ]; then # Set default gateway sudo route add default gw $gateway if [ $? -eq 0 ]; then # Set DNS servers sudo echo "nameserver $dns" > /etc/resolv.conf if [ $? -eq 0 ]; then echo "----------------------------------------------" echo "Network configuration completed" echo "----------------------------------------------" else echo "----------------------------------------------" echo "Error while setting the DNS server." fi else echo "----------------------------------------------" echo "Error while setting the default gateway." fielse echo "----------------------------------------------" echo "Network Configuration Failed."fi** |
| --- |

**Syntax to run the Script: sudo bash bin/adv\_example16.sh**

**Requirement:** [ifconfig must be installed.](https://linuxsimply.com/ifconfig-command-in-linux/#Installing_the_ifconfig_Command_in_Linux)

**Output:**

| Enter network configuration variables:Enter an IP address: 192.168.0.108Enter a subnet mask: 255.255.255.0Enter a Gateway address: 192.168.0.1Enter a DNS address: 8.8.8.8----------------------------------------------Network configuration completed---------------------------------------------- |
| --- |

#### **17. Check if a Process is Running**

The given script can check if a process is currently running on your system or not. You will need to enter your desired process name and the script will display the process’s current status.

**Code:**

| #!/bin/bashread -p "Enter process name: " processif pgrep $process > /dev/nullthen echo "Process is running."else echo "Process is not running."fi |
| --- |

**Output:**

| **Enter process name: bashProcess is running.** |
| --- |

### **Process Management Based Shell Scripts**

#### **18. Start a Process if It's Not Already Running**

You can use the script given below to start a process. The process name is passed as user input to the script. If the process is already running then it will return a message saying “The Process is already running”. Otherwise, It will start the desired process.

**Code:**

| #!/bin/bashread -p "Enter process name: " processif ! pgrep $process > /dev/nullthen /path/to/process\_name & echo "The Process $process has now started."else echo "The Process is already running."fi |
| --- |

**Output:**

| **Enter process name: bashThe Process is already running.** |
| --- |

#### **19. Stop a Process**

The script below can stop a process if it runs in the system. The user has to enter a process name as the script input. If the process is currently running then the script will terminate that process. Otherwise, it says, “The process is not running”.

**Code:**

| #!/bin/bashread -p "Enter process name: " processif pgrep $process > /dev/nullthen pkill $process echo "The Process $process has stopped."else echo "The Process $process is not running."fi |
| --- |

**Output:**

| **Enter process name: nslookupThe Process nslookup has stopped.** |
| --- |

#### **20. Restart a Process**

The following script aims to take a process name as input and then restart it. If the process is already running then the script kills the process and starts over. After the first kill command, it waits for 5 seconds. If by then the process does not terminate then it will force kill that process before restarting.

**Code:**

| #!/bin/bashread -p "Enter process name: " processpid=$(pgrep -f $process)if [ -n "$pid" ]; then kill $pid sleep 5 if pgrep -f $process> /dev/null; then echo "Process did not exit properly, force killing..." kill -9 $pid fifiprocess\_path=$(which $process)$process\_path & echo "Process restarted." |
| --- |

**Output:**

| **Enter process name: firefoxThe Process firefox is running.The Process firefox is running.** |
| --- |

#### **21. Monitor a Process and Restart It If Crashes**

The script here, takes a process name as input from the user and checks for its status every 5 seconds. If the process is running without any issues then it shows a message saying “The process is running”. Otherwise, it restarts the process and continues to check its status again.

**Code:**

| #!/bin/bashread -p "Enter process name: " processprocess\_path=$(which $process)while truedo if pgrep $process > /dev/null then echo "The Process $process is running." sleep 5 else $process\_path & echo "The Process $process restarted." continue fidone |
| --- |

**Output:**

| **Enter process name: firefoxThe Process firefox is running.The Process firefox is running.** |
| --- |

#### **22. Display the Top 10 CPU-Consuming Processes**

The script below lists the top 10 CPU-consuming processes. It prints the Process ID, the percentage of CPU usage along with the command that runs each process.

**Code:**

| #!/bin/bashecho "The current top 10 CPU-consuming processes: "ps -eo pid,%cpu,args | sort -k 2 -r | head -n 11 |
| --- |

**Output:**

| **The current top 10 CPU-consuming processes:  PID %CPU COMMAND 2161 0.6 /usr/bin/gnome-shell 1126 0.5 /usr/sbin/mysqld 7593 0.5 /usr/libexec/gnome-terminal-server 832 0.2 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080 668 0.1 /usr/bin/vmtoolsd 5498 0.1 gjs /usr/share/gnome-shell/extensions/ding@rastersoft.com/ding.js -E -P /usr/share/gnome-shell/extensions/ding@rastersoft.com -M 0 -D 0:0:1918:878:1:34:0:0:0:0 104 0.0 [zswap-shrink] 86 0.0 [xenbus\_probe] 26 0.0 [writeback] 39 0.0 [watchdogd]** |
| --- |

#### **23. Display the Top 10 Memory-Consuming Processes**

The given script lists the top 10 memory-consuming processes. It prints the Process ID, percentage of memory usage as well as the commands for running each process.

**Code:**

| #!/bin/bashecho "The current top 10 Memory-consuming processes: "ps -eo pid,%mem,args | sort -k 2 -r | head -n 11 |
| --- |

**Output:**

| **The current top 10 Memory-consuming processes:  PID %MEM COMMAND 1126 9.7 /usr/sbin/mysqld 832 6.8 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080 2161 6.7 /usr/bin/gnome-shell 2516 2.1 /usr/bin/Xwayland :0 -rootless -noreset -accessx -core -auth /run/user/1000/.mutter-Xwaylandauth.G8UR41 -listen 4 -listen 5 -displayfd 6 -initfd 7 2585 1.9 /usr/libexec/gsd-xsettings 1209 1.5 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock 5498 1.5 gjs /usr/share/gnome-shell/extensions/ding@rastersoft.com/ding.js -E -P /usr/share/gnome-shell/extensions/ding@rastersoft.com -M 0 -D 0:0:1918:878:1:34:0:0:0:0 2966 1.4 /usr/bin/gedit --gapplication-service 7593 1.3 /usr/libexec/gnome-terminal-server 2381 1.3 /usr/libexec/evolution-data-server/evolution-alarm-notify** |
| --- |

#### **24. Kill Processes of a Specific User**

The following script is created to kill all the processes of a specific user. The Specified username is taken as user input. After receiving the username, all the running processes of that user are terminated.

**Code:**

| #!/bin/bashread -p "Enter username: " usersudo pkill -u $userecho "All processes of user $user have been terminated." |
| --- |

**Output:**

| **Enter username: tom[sudo] password for susmit: All processes of user tom have been terminated.** |
| --- |

#### **25. Kill All Processes That are Consuming More Than a Certain Amount of CPU**

This script takes a CPU usage percentage as user input and terminates all the running processes that are consuming more than the entered CPU threshold. If there is no process above that threshold, then it returns a message saying there are no such processes.

**Code:**

| #!/bin/bashread -p "Enter CPU usage threshold: " thresholdif [ "$(ps -eo pid,%cpu | awk -v t=$threshold '$2 > t {print $1}' | wc -c)" -gt 0 ]; thenfor pid in $(ps -eo pid,%cpu | awk -v t=$threshold '$2 > t {print $1}')do kill $piddoneecho "All processes consuming more than $threshold% CPU killed."elseecho "There are no process consuming more than $threshold% CPU."fi |
| --- |

**Output:**

| **Enter CPU usage threshold: 10There are no process consuming more than 10% CPU.** |
| --- |

#### **26. Kill All Processes That are Consuming More Than a Certain Amount of Memory**

This script takes a memory space percentage as user input and terminates all the running processes that are consuming more than the entered space threshold. If there is no process above that threshold, then it returns a message saying there are no such processes.

**Code:**

| #!/bin/bashread -p "Enter CPU usage threshold: " thresholdif [ "$(ps -eo pid,%cpu | awk -v t=$threshold '$2 > t {print $1}' | wc -c)" -gt 0 ]; thenfor pid in $(ps -eo pid,%cpu | awk -v t=$threshold '$2 > t {print $1}')do kill $piddoneecho "All processes consuming more than $threshold% CPU killed."elseecho "There are no process consuming more than $threshold% CPU."fi |
| --- |

**Output:**

| Enter memory usage threshold (in KB): 10There are no process consuming more than 10 KB memory. |
| --- |

### **System Information Based Shell Scripts**

#### **27. Check the Number of Logged-in Users**

You view the find the number of logged-in users in your system with the script below. It counts the users that are logged in only at the current time.

**Code:**

| #!/bin/bashusers=$(who | wc -l)echo "Number of currently logged-in users: $users" |
| --- |

**Output:**

| **Number of currently logged-in users: 2** |
| --- |

#### **28. Check the Operating System Information**

The following script displays information regarding the machine’s operating system. It retrieves and lists the os name, release, version as well as system architecture.

**Code:**

| #!/bin/bashos\_name=$(uname -s)os\_release=$(uname -r)os\_version=$(cat /etc/\*-release | grep VERSION\_ID | cut -d '"' -f 2)os\_arch=$(uname -m)echo "OS Name: $os\_name"echo "OS Release: $os\_release"echo "OS Version: $os\_version"echo "OS Architecture: $os\_arch" |
| --- |

**Output:**

| **OS Name: LinuxOS Release: 5.19.0-38-genericOS Version: 22.04OS Architecture: x86\_64** |
| --- |

#### **29. Check the System’s Memory Usage**

The script given below calculates the percentage of memory being used. The “**$3\*100/$2**” expression converts the usage into percentages and displays the output with two decimal places.

**Code:**

| #!/bin/bashmem=$(free -m | awk 'NR==2{printf "%.2f%%", $3\*100/$2}')echo "Current Memory Usage: $mem" |
| --- |

**Output:**

| **Current Memory Usage: 72.48%** |
| --- |

#### **30. Check the System’s Disk Usage**

The following script displays the percentage of disk space used on the root (/) file system. It gets the file system’s disk space usage in a human-readable format and prints only the used percentage.

**Code:**

| #!/bin/bashdisk=$(df -h | awk '$NF=="/"{printf "%s", $5}')echo "Current Disk Usage: $disk" |
| --- |

**Output:**

| **System's network information:-IP Address: 192.168.0.109 Gateway: 192.168.0.1DNS Server: 127.0.0.53** |
| --- |

#### **31. Check the System’s Network Information**

Use the script below to get the network information of your system. It lists the system’s IP address, Gateway address, and DNS server address.

**Code:**

| #!/bin/bashecho " System's network information:-"ip=$(hostname -I)echo "IP Address: $ip"gw=$(ip route | awk '/default/ { print $3 }')echo "Gateway: $gw"dns=$(grep "nameserver" /etc/resolv.conf | awk '{print $2}')echo "DNS Server: $dns" |
| --- |

**Output:**

| **System's network information:-IP Address: 192.168.0.109 Gateway: 192.168.0.1DNS Server: 127.0.0.53** |
| --- |

#### **32. Check the Uptime**

The given script can be used to find out the uptime of the system. It will return two values. The first one is the current time, and the second one is the uptime i.e. for how long the system has been running. In this example, “**up 16:19**”indicates that the system has been up for 16 hours and 19 minutes.

**Code:**

| #!/bin/bashuptime | awk '{print $1,$2,$3}' | sed 's/,//'echo "Uptime: $uptime" |
| --- |

**Output:**

| **Uptime: 00:16:38 up 16:19** |
| --- |

#### **33. Check the System Load Average**

The following script returns the system’s Load Average. It will extract the load averages for the past 1, 5, and 15 minutes from the system’s uptime and display their average on the screen.

**Code:**

| #!/bin/bashloadavg=$(uptime | awk '{print $10,$11,$12}')echo "Load Average: $loadavg"  |
| --- |

**Output:**

| **Load Average: 0.36** |
| --- |

#### **34. Check the System Architecture**

To determine your current machine's architecture you can run the following script. It returns the system’s architecture. In this example, **x86\_64** indicates that the machine is using the 64-bit version of the x86 architecture.

**Code:**

| #!/bin/basharch=$(uname -m)echo "System Architecture: $arch" |
| --- |

**Output:**

| **System Architecture: x86\_64** |
| --- |

#### **35. Count the Number of Files in the System**

You can use the script below to find the available number of files on your machine. It runs the find command to check every file on the system and returns the total file count.

**Code:**

| #!/bin/bashcount=$(find / -type f | wc -l)echo "Number of files in the system: $count." |
| --- |

**Output:**

| **Number of files in the system: 500090.** |
| --- |

### **Advanced Tasks with Shell Scripts**

#### **36. Automated Backup**

The following script creates a backup file of a given directory. The source directory path and the destination directory path are user inputs. The backup file is named along with the current date for keeping track. Upon completion of the task, it returns the path where the backup archive resides.

**Code:**

| #!/bin/bashread -p "Enter path of the directory to backup: " source\_dirread -p "Enter destination path for backup: " backup\_dirdate=$(date +%Y-%m-%d)backup\_file="backup-$date.tar.gz"# Create backup directory if it doesn't existif [ ! -d "$backup\_dir" ]; then mkdir -p "$backup\_dir"fi# Create backup archivetar -czf "$backup\_dir/$backup\_file" "$source\_dir"echo "Completed Creating backup at: $backup\_dir." |
| --- |

**Output:**

| **Enter filename to write alert: alert.logEnter disk space threshold: 70Alert for "/dev/sda3: Almost out of disk space 80% as on Thu May 11 01:54:50 AM EDT 2023** |
| --- |

#### **37. Generate Alert if Disk Space Usage Goes Over a Threshold**

The script below generates an alert if the disk space usage goes over a threshold. It takes the threshold and a filename from the user. The alert is then generated in that file along with the disk space usage. If the space consumed is less than the threshold than the file remains empty.

**Code:**

| #!/bin/bashread -p "Enter filename to write alert: " filetouch $fileread -p "Enter disk space threshold: " thresholddf -H | grep -vE "^Filesystem|tmpfs|cdrom" | awk '{ print $5 " " $1 }' | while read output;do usage=$(echo $output | awk '{ print $1}' | cut -d'%' -f1) if [ $usage -ge $threshold ]; then partition=$(echo $output | awk '{ print $2 }') echo "Alert for \"$partition: Almost out of disk space $usage% as on $(date) " >> $file break  fidonecat $file |
| --- |

**Output:**

| Enter filename to write alert: alert.logEnter disk space threshold: 70Alert for "/dev/sda3: Almost out of disk space 80% as on Thu May 11 01:54:50 AM EDT 2023 |
| --- |

#### **38. Create a New User and Add to Sudo Group**

You can use the following script to create a new sudo user in your Linux system. The script will take the username and password as input to create the user. It will also create a home directory for the user besides adding the account to the sudo group.

**Code:**

| #!/bin/bashread -p "Enter username: " usernameread -p "Enter password: " passworduseradd -m -s /bin/bash -p $(openssl passwd -1 $password) $usernameif [ $? -eq 0 ]; thenusermod -a -G sudo $usernamemkdir /home/$username/mydirchown -R $username:$username /home/$username/mydirusermod -d /home/$username/mydir $usernameecho "$username ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoersecho "User $username created successfully!"echo "User $username added to sudo group!"elseecho "Error while creating user!"fi |
| --- |

**Syntax to run the Script: sudo bash bin/adv\_example38.sh**

**Output:**

| **Enter username: susmitEnter password: linuxsimplyUser susmit created successfully!User susmit added to sudo group!** |
| --- |

#### **39. Monitor Network Traffic**

The following script monitors the receiving (RX) and transmitting(TX) packets over a network. User needs to enter the interface name which they want to monitor. Then in every 10 seconds it will display the total packet received and transmitted and their size in KB.

**Code:**

| #!/bin/bashread -p "Enter network interface to monitor traffic (ex. eth0): " netwhile truedo rx=$(ifconfig $net | grep "RX packets" | awk '{print $3 $6 $7}') tx=$(ifconfig $net | grep "TX packets" | awk '{print $3 $6 $7}') echo "$(date) RX: $rx, TX: $tx" sleep 10done |
| --- |

**Output:**

| **Enter network interface to monitor traffic (ex. eth0): ens33Wed May 10 16:55:40 +06 2023 RX: 342(40.4KB), TX: 171(18.4KB)Wed May 10 16:55:51 +06 2023 RX: 355(41.6KB), TX: 178(19.0KB)Wed May 10 16:56:01 +06 2023 RX: 361(42.0KB), TX: 178(19.0KB)Wed May 10 16:56:11 +06 2023 RX: 361(42.0KB), TX: 178(19.0KB)** |
| --- |

#### **40. Monitor CPU and Memory Usage**

The script below can be used to monitor the CPU and Memory usage of a system. It extracts the CPU and Memory usage information every 10 seconds and converts them into a percentage for displaying on the screen.

**Code:**

| #!/bin/bashwhile truedo cpu=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1"%"}') mem=$(free -m | awk 'NR==2{printf "%.2f%%", $3\*100/$2 }') echo "$(date) CPU Usage: $cpu, Memory Usage: $mem" sleep 10done |
| --- |

**Output:**

| **Sun May 7 02:19:49 AM EDT 2023 CPU Usage: 29.4%, Memory Usage: 68.78%Sun May 7 02:19:59 AM EDT 2023 CPU Usage: 7.1%, Memory Usage: 68.78%Sun May 7 02:20:10 AM EDT 2023 CPU Usage: 25%, Memory Usage: 68.72%Sun May 7 02:20:20 AM EDT 2023 CPU Usage: 17.6%, Memory Usage: 68.72%Sun May 7 02:20:30 AM EDT 2023 CPU Usage: 6.2%, Memory Usage: 68.70%** |
| --- |

#### **41. Creating a Script and Adding It to PATH**

You can use the script below to customize another script and make it runnable. The script here will take another script name and the commands to write within this new script as user inputs. After receiving the input values, it will update the permission modes of the desired script and add it to the **$PATH** variable to make the new script runnable. After creation, you can run this new script with the **bash** keyword.

**Code:**

| #!/bin/bashread -p "Enter a name for the command: " my\_commecho "Enter commands to write on script:"read commread -p "Enter path to the directory containing the command: " comm\_path# Create script for custom commandecho "#!/bin/bash" > $my\_comm.shecho "$comm" >> $my\_comm.sh# Make script executablechmod +x $my\_comm.sh# Add script to PATHexport PATH="$PATH$comm\_path/$my\_comm.sh"echo "A script called $my\_comm has been created." |
| --- |

**Output:**

| Enter a name for the command: echo\_helloEnter commands to write on script:echo "Hello from custom command!!"Enter path to the directory containing the command: /home/susmit/binA script called echo\_hello has been created. |
| --- |

#### **42. Running a Command At Regular Intervals**

The script given below runs a command at a regular time interval. To achieve this task the user has to enter the desired command and the interval for running that command. The interval passed as input must be in the following format: m h dom mon dow.

**Code:**

| #!/bin/bashread -p "Enter command to run: " comcommand\_to\_run=$(which $com)read -p "Enter interval for running the command (m h dom mon dow): " interval# Add command to crontab(crontab -l ; echo "$interval $command\_to\_run") | sort - | uniq - | crontab -echo "Command added to crontab and will run at $interval" |
| --- |

**Output:**

| Enter command to run: echo "1 Minute passed!" >> time.logEnter interval for running the command (m h dom mon dow): \* \* \* \* \*Command added to crontab and will run at \* \* \* \* \* |
| --- |

#### **43. Downloading Files from a List of URLs**

The following script takes a filename as input where a list of URLs should be stored. The script will iterate through the list of URLs and download the available contents on the link. It displays each download information on the terminal along with the “Completed Download” message. Upon downloading files from all the URLs, it shows another message saying “All files downloaded successfully!”.

**Code:**

| #!/bin/bashread -p "Enter the filename containing URLs: " url\_filewhile read -r url; do filename=$(basename "$url") curl -o "$filename" "$url" echo "Completed Download $filename"done < "$url\_file"echo "--------------------------------------------------------------------------------------------"echo "All files downloaded successfully!" |
| --- |

**Output:**

| **Enter the filename containing URLs: urls.txt % Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left Speed 0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (6) Could not resolve host: linuxsimply.comCompleted Download Emacs-Keybindings-or-Shortcuts-in-Linux.pdfcurl: (3) URL using bad/illegal format or missing URLDownloaded  % Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left Speed 0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (6) Could not resolve host: linuxsimply.comCompleted Download Bash-Terminal-Keyboard-Shortcuts-for-Information.pdf--------------------------------------------------------------------------------------------All files downloaded successfully!** |
| --- |

#### **44. Organizes Files in a Directory Based on Their File Types**

The script given below organizes files in a directory depending on their type. The user needs to give a destination directory path to organize the files along with the source directory path.

This script will create five directories: 1) Documents, 2) Images, 3) Music, 4) Videos, and 5) Others only if they do not already exist on the destination path. Then, it will check all the files and their extension and move them to the corresponding directory. If there is any unknown file extension, then the script will move the file to the Others Directory.

**Code:**

| #!/bin/bash# Specify the source and destination directoriesread -p "Enter path to the source directory: " source\_dirread -p "Enter path to the destination directory: " dest\_dir# Create the destination directories if they don't existmkdir -p "${dest\_dir}/Documents"mkdir -p "${dest\_dir}/Images"mkdir -p "${dest\_dir}/Music"mkdir -p "${dest\_dir}/Videos"mkdir -p "${dest\_dir}/Others"# Move files to the appropriate directories based on their extensionsfor file in "${source\_dir}"/\*; do if [ -f "${file}" ]; then extension="${file##\*.}" case "${extension}" in txt|pdf|doc|docx|odt|rtf) mv "${file}" "${dest\_dir}/Documents" ;; jpg|jpeg|png|gif|bmp) mv "${file}" "${dest\_dir}/Images" ;; mp3|wav|ogg|flac) mv "${file}" "${dest\_dir}/Music" ;; mp4|avi|wmv|mkv|mov) mv "${file}" "${dest\_dir}/Videos" ;; \*) mv "${file}" "${dest\_dir}/Others" ;; esac fidoneecho "Files organized successfully!" |
| --- |

**Output:**

| **Enter path to the source directory: /home/susmit/DownloadsEnter path to the destination directory: /home/susmit/Downloads\_OrganizedFiles organized successfully!** |
| --- |

## **Conclusion**

From complex task automation to efficient data manipulation, you now possess the ability to tackle real-world challenges with confidence. Embrace the power of advanced shell scripting and unlock a world of automation and efficiency.

|  | **Prepared By:** [**Susmit Das Gupta**](https://linuxsimply.com/susmit-das-gupta/) | **Web View:** [**Advanced Shell Script With Examples [Free Downloads]**](https://linuxsimply.com/advanced-shell-script-examples/)**Copyright ©2024 linuxsimply.com| All rights reserved** |
| --- | --- | --- |