Shell Scripting Commands

Command	Description	Example	Output
echo	Prints text to the terminal.	`echo "Hello, world!"`	Hello, world!
if	Conditional statement.	`if [ "$VAR" -eq 10 ]; then echo "VAR is 10"; fi`	(Output depends on the value of **$VAR**)
for	Looping construct.	`bash for i in {1..5}; do echo "Iteration: $i"; done`	Iteration: 1 Iteration: 2 Iteration: 3 Iteration: 4 Iteration: 5
while	Looping construct.	`bash i=0; while [ $i -lt 5 ]; do echo "Iteration: $i"; ((i++)); done`	Iteration: 0 Iteration: 1 Iteration: 2 Iteration: 3 Iteration: 4
case	Conditional construct.	`bash case "$VAR" in   1) echo "VAR is 1";;   2) echo "VAR is 2";;   *) echo "VAR is neither 1 nor 2";;   esac`	(Output depends on the value of **$VAR**)

read	Reads input from the user.	`bash echo "Enter your name: "; read name; echo "Hello, $name!";`	(Depends on user input)
grep	Searches for patterns in files.	`grep "pattern" file.txt`	(Output depends on matches)
sed	Stream editor for filtering and transforming text.	`sed 's/old_text/new_text/' file.txt`	(Output depends on substitutions made)
awk	Text processing tool for pattern scanning and processing.	`awk '{print $1}' file.txt`	(Output depends on text processing)
cut	Extracts sections from each line of files.	`cut -d',' -f1 file.csv`	(Output depends on sections extracted)
chmod	Changes file permissions.	`chmod +x script.sh`	(No output if successful)
chown	Changes file owner and group.	`chown user:group file.txt`	(No output if successful)
mv	Moves or renames files/directories.	`mv file.txt new_location/`	(No output if successful)
cp	Copies files/directories.	`cp file.txt copy_of_file.txt`	(No output if successful)
rm	Removes files/directories.	`rm file.txt`	(No output if successful)
mkdir	Creates directories.	`mkdir new_directory`	(No output if successful)
rmdir	Removes directories.	`rmdir directory_to_remove`	(No output if successful)
touch	Creates an empty file.	`touch new_file.txt`	(No output if successful)

ln	Creates links between files.	`ln -s /path/to/file /path/to/link`	(No output if successful)
find	Searches for files and directories.	`find . -name "*.txt"`	(Output lists found files and directories)
wc	Counts lines, words, or characters in a file.	`wc -l file.txt`	(Output shows the number of lines)
sort	Sorts lines of text files.	`sort file.txt`	(Output displays sorted lines)
uniq	Removes duplicate lines from a sorted file.	`uniq file.txt`	(Output displays unique lines)
diff	Compares files line by line.	`diff file1.txt file2.txt`	(Output shows the differences between files)
head	Outputs the first part of files.	`head -n 5 file.txt`	(Output displays the first 5 lines)
tail	Outputs the last part of files.	`tail -n 5 file.txt`	(Output displays the last 5 lines)
tar	Manipulates archive files.	`tar -czvf archive.tar.gz directory_to_compress/`	(No output if successful)
unzip	Extracts files from a ZIP archive.	`unzip archive.zip`	(No output if successful)
wget	Downloads files from the internet.	`wget https://example.com/file.txt`	(No output if successful)

curl	Transfers data from or to a server.	`curl -O https://example.com/file.txt`	(No output if successful)
ps	Displays information about running processes.	`ps aux`	(Output lists running processes)
kill	Terminates processes by ID or name.	`kill PID`	(No output if successful)
df	Displays disk space usage.	`df -h`	(Output shows disk space usage)
du	Displays disk usage for files and directories.	`du -sh directory`	(Output shows disk usage summary)
date	Displays or sets the system date and time.	`date`	(Output shows current date and time)
uname	Displays system information.	`uname -a`	(Output shows system information)
uptime	Displays system uptime.	`uptime`	(Output shows system uptime)
hostname	Displays or sets the system's hostname.	`hostname`	(Output shows system hostname)

Sed Commands:			
Command	**Description**	**Example**	**Output**
s	Substitutes text with another.	`sed 's/old_text/new_text/g' file.txt`	(Output with substitutions made)
d	Deletes lines matching a pattern.	`sed '/pattern/d' file.txt`	(Output with matching lines deleted)
p	Prints lines matching a pattern.	`sed -n '/pattern/p' file.txt`	(Output with matching lines printed)
i	Inserts text before a line.	`sed '/pattern/i¥new_line' file.txt`	(Output with new line inserted before pattern)
a	Appends text after a line.	`sed '/pattern/a¥new_line' file.txt`	(Output with new line appended after pattern)
y	Translates characters.	`sed 'y/abcd/ABCD/' file.txt`	(Output with characters translated)
/^pattern/	Matches lines starting with pattern.	`sed '/^pattern/' file.txt`	(Output with lines starting with pattern matched)
/pattern$/	Matches lines ending with pattern.	`sed '/pattern$/' file.txt`	(Output with lines ending with pattern matched)

Awk commands:			
Command	**Description**	**Example**	**Output**
{print}	Prints entire line.	`awk '{print}' file.txt`	(Output with entire lines printed)
{print $n}	Prints nth field of each line.	`awk '{print $2}' file.txt`	(Output with second field of each line printed)
{print NF}	Prints number of fields in each line.	`awk '{print NF}' file.txt`	(Output with number of fields printed)
{printf}	Prints formatted output.	`awk '{printf "%s %s¥n", $1, $2}' file.txt`	(Output with formatted output)
{if}	Conditional statement.	`awk '{if($1 > 10) print $0}' file.txt`	(Output with lines where first field > 10)
{for}	Looping construct.	`awk '{for(i=1;i<=NF;i++) print $i}' file.txt`	(Output with each field printed on separate line)
{split}	Splits a string into an array.	`awk '{split($0, arr, ":"); print arr[1]}' file.txt`	(Output with first element of split string)
{gsub}	Global substitution of text.	`awk '{gsub("old_text", "new_text", $0); print}' file.txt`	(Output with global substitutions made)
{length}	Returns length of a string.	`awk '{print length($0)}' file.txt`	(Output with length of each line printed)

Interview Question & Answers

1. **What is Shell scripting?**
    - Shell scripting is scripting in any shell language, which is a command-line interpreter. It is a sequence of commands written for execution by a shell.
2. **Differentiate between a shell variable and an environment variable.**
    - A shell variable is a variable that is valid only in the current shell session, while an environment variable is available to any child processes of the shell.
3. **What is the shebang (#!) in a shell script, and why is it used?**
    - The shebang is a special character sequence (#!) at the beginning of a script that tells the system which interpreter to use to execute the script. For example, #!/bin/bash indicates that the script should be executed using the Bash shell.
4. **How do you comment a line in a shell script?**
    - In shell scripting, a line can be commented using the # symbol. Everything after # on a line is considered a comment.
5. **What is the purpose of the echo command in shell scripting?**
    - The echo command is used to print text or variables to the terminal.
6. **How do you assign a value to a variable in shell scripting?**
    - Variables are assigned values using the syntax variable_name=value. For example, x=10.
7. **What is the difference between single quotes (' ') and double quotes (" ") in shell scripting?**

- Single quotes preserve the literal value of each character enclosed within the quotes, whereas double quotes allow for variable expansion and certain character substitutions.

8. **Explain the significance of the $ symbol in shell scripting.**
    - In shell scripting, the $ symbol is used to access the value of a variable. For example, $x represents the value of the variable x.
9. **How do you read user input in a shell script?**
    - User input can be read using the read command followed by the variable name. For example, read name.
10. **What is the purpose of the if statement in shell scripting?**
    - The if statement is used for conditional execution in shell scripting. It allows you to execute a set of commands based on whether a certain condition is true or false.
11. **Explain the difference between && and || operators in shell scripting.**
    - && (AND) executes the command following it only if the preceding command succeeds (returns a status code of 0). || (OR) executes the command following it only if the preceding command fails (returns a status code other than 0).
12. **How do you loop through a set of values in shell scripting?**
    - You can use the for loop to iterate over a set of values. For example, for i in {1..5}; do echo $i; done will print numbers from 1 to 5.
13. **What is the purpose of the case statement in shell scripting?**
    - The case statement is used for multiway branching in shell scripting. It allows you to match the value of a variable against various patterns and execute different commands based on the match.
14. **How do you define a function in shell scripting?**
    - Functions are defined using the syntax function_name() { ... }. For example, hello() { echo "Hello, world!"; }.
15. **What is the purpose of the exit command in shell scripting?**

- The exit command is used to exit the shell script with a specified exit status. It is often used to indicate success or failure of the script execution.

16. Explain the role of the shift command in shell scripting.

- The shift command is used to shift command line arguments one position to the left. This is useful when processing command line arguments in a loop.

17. How do you perform arithmetic operations in shell scripting?

- Arithmetic operations can be performed using the expr command or the $((...)) syntax. For example, sum=$(($a + $b)) or sum=$(expr $a + $b).

18. What is a here document in shell scripting?

- A here document is a way to redirect input to a command or a shell script from a block of text within the script itself. It is specified using << followed by a delimiter.

19. How do you check if a file exists in shell scripting?

- You can use the -e option with the test command or [ ] to check if a file exists. For example, if [-e file.txt]; then echo "File exists"; fi.

20. Explain the purpose of the grep command in shell scripting.

- The grep command is used to search for patterns in files. It prints lines that match the specified pattern.

21. How do you redirect standard output to a file in shell scripting?

- Standard output can be redirected to a file using the > operator. For example, echo "Hello" > output.txt.

22. What is the purpose of the awk command in shell scripting?

- The awk command is a powerful text processing tool used for pattern scanning and processing. It allows you to manipulate text data based on patterns and columns.

23. Explain the difference between sed and awk commands.

- sed is primarily used for text substitution and transformation, while awk is more versatile and can perform complex text processing operations including pattern matching, data extraction, and reporting.

24. How do you find the length of a string in shell scripting?

- You can find the length of a string using the ${#string} syntax. For example, len=${#str} will store the length of the string str in the variable len.

25. What is the purpose of the dirname command in shell scripting?

- The dirname command extracts the directory portion of a pathname and prints it.

26. How do you list all files in a directory in shell scripting?

- You can list all files in a directory using the ls command. For example, ls -l will list all files in long format.

27. Explain the purpose of the cut command in shell scripting.

- The cut command is used to extract sections from each line of files. It is often used to extract specific columns from text files.

28. How do you remove leading and trailing whitespace from a string in shell scripting?

- Leading and trailing whitespace can be removed using the trim() function with awk. For example, trimmed=$(echo $str | awk '{gsub(/^ +| +$/,"")} {print $0}').

29. What is the purpose of the tr command in shell scripting?

- The tr command is used for translating or deleting characters. It can be used to perform character-based substitutions.

30. How do you check if a directory exists in shell scripting?

- You can use the -d option with the test command or [ ] to check if a directory exists. For example, if [-d directory]; then echo "Directory exists"; fi.

31. Explain the significance of the grep -v command in shell scripting.

- The grep -v command is used to invert the match, i.e., it prints lines that do not match the specified pattern.

32. How do you concatenate two strings in shell scripting?

- Strings can be concatenated using the concatenation operator . or by simply placing them adjacent to each other. For example, result=$str1$str2 or result=$str1$str2.

33. What is the purpose of the basename command in shell scripting?

- The basename command strips directory and suffix from filenames. It extracts the last component of a pathname.

34. How do you check if a string contains a substring in shell scripting?

- You can use the grep command with the -q option to check if a string contains a substring. For example, if echo "$str" | grep -q "substring"; then echo "Substring found"; fi.

35. What is the significance of the tee command in shell scripting?

- The tee command reads from standard input and writes to standard output and files simultaneously. It is often used to display output and write it to a file at the same time.

36. Explain the purpose of the uniq command in shell scripting.

- The uniq command is used to remove duplicate lines from a sorted file. It compares adjacent lines and removes duplicates.

37. How do you convert uppercase to lowercase characters in shell scripting?

- You can convert uppercase to lowercase characters using the tr command. For example, lowercase=$(echo "$uppercase" | tr '[:upper:]' '[:lower:]').

38. What is the purpose of the dirname command in shell scripting?

- The dirname command extracts the directory portion of a pathname and prints it.

39. How do you perform substring extraction in shell scripting?

- Substring extraction can be done using parameter expansion. For example, substring=${string:position:length} extracts length characters starting from position in string.

40. What is process substitution in shell scripting?

- Process substitution is a form of substitution that allows you to use the output of a command or a shell script as input to another command.

41. Explain the use of the mktemp command in shell scripting.

- The mktemp command is used to create temporary files and directories securely. It generates a unique filename based on a template.

42. How do you rename a file in shell scripting?

- You can rename a file using the mv command followed by the old and new filenames. For example, mv old_file.txt new_file.txt.

43. What is the purpose of the awk '{print $NF}' command in shell scripting?

- The awk '{print $NF}' command prints the last field of each line in a file. $NF represents the last field.

44. Explain the use of the head and tail commands in shell scripting.

- The head command is used to display the first few lines of a file, while the tail command is used to display the last few lines of a file.

45. How do you check if a command exists in shell scripting?

- You can use the command -v command followed by the command name to check if a command exists. For example, if command -v command_name >/dev/null 2>&1; then echo "Command exists"; fi.

46. What is the purpose of the read -p command in shell scripting?

- The read -p command is used to prompt the user for input with a message. For example, read -p "Enter your name: " name.

47. How do you perform arithmetic operations with floating-point numbers in shell scripting?

- Shell scripting does not support floating-point arithmetic natively. However, you can use external programs like bc or awk for floating-point arithmetic.

48. Explain the significance of the tee -a command in shell scripting.

- The tee -a command is used to append output to a file instead of overwriting it. It is useful for logging output to a file without losing existing content.

49. What is the purpose of the shift command in shell scripting?

- The shift command is used to shift command line arguments one position to the left. This is useful when processing command line arguments in a loop.

50. How do you find the process ID of a running script in shell scripting?

- You can use the $$ variable to find the process ID of the current shell script. For example, echo $$ will print the process ID.

51. Explain the purpose of the trap command in shell scripting.

- The trap command is used to intercept and respond to signals received by the shell. It allows you to define actions to be taken when specific signals are received.

52. Differentiate between errors and exceptions in shell scripting.

- In shell scripting, errors typically refer to conditions that prevent a command or script from executing successfully, such as syntax errors or command not found errors. Exceptions, on the other hand, are unexpected conditions that arise during the execution of a script, often requiring special handling to prevent termination.

53. What is the significance of the set -e command in shell scripting?

- The set -e command enables the shell's errexit option, which causes the script to exit immediately if any command exits with a non-zero status.

54. Explain the purpose of the eval command in shell scripting.

- The eval command evaluates and executes the arguments as a shell command. It is often used for dynamic code execution and variable expansion.

55. How do you handle command line arguments in shell scripting?

- Command line arguments can be accessed using variables like $1, $2, etc., or by using the shift command to iterate over them in a loop.

56. What is the purpose of the getopts command in shell scripting?

- The getopts command is used to parse command line options and arguments. It allows you to specify options with arguments and handles error checking.

57. Explain the purpose of the readonly command in shell scripting.

- The readonly command is used to create variables that cannot be modified or unset later in the script. They are similar to constants.

58. What is process substitution in shell scripting?

- Process substitution is a form of substitution that allows you to use the output of a command or a shell script as input to another command.

59. Explain the purpose of the sleep command in shell scripting.

- The sleep command is used to suspend execution for a specified amount of time. It is often used to introduce delays in scripts.

60. What is the purpose of the declare command in shell scripting?

- The declare command is used to declare variables and give them attributes. It is used for defining variables with specific characteristics such as readonly, integer, etc.

61. Write a shell script to find the sum of all numbers in a given list.

```
#!/bin/bash
sum=0
for num in "$@"; do
    sum=$((sum + num))
done
echo "Sum: $sum"
```

Example Usage:

```
./sum.sh 10 20 30
```

Output:

```
Sum: 60
```

62. **Write a shell script to check if a number is prime.**

```
#!/bin/bash
is_prime() {
    n=$1
    if [ $n -le 1 ]; then
        echo "$n is not prime"
        return
    fi
    for ((i=2; i*i<=n; i++)); do
        if [ $((n % i)) -eq 0 ]; then
```

```
            echo "$n is not prime"
            return
        fi
    done
    echo "$n is prime"
}
is_prime "$1"
```

Example Usage:

./prime.sh 17

Output:

17 is prime

63. **Write a shell script to find the factorial of a number.**

```
#!/bin/bash
factorial() {
    n=$1
    if [ $n -eq 0 ]; then
        echo "1"
        return
    fi
    fact=1
```

```
    for ((i=1; i<=n; i++)); do
        fact=$((fact * i))
    done
    echo "$fact"
}
factorial "$1"
```

Example Usage:

./factorial.sh 5

Output:

120

64. **Write a shell script to calculate the average of numbers in a given list.**

```
#!/bin/bash
calculate_average() {
    sum=0
    count=0
    for num in "$@"; do
        sum=$((sum + num))
        ((count++))
    done
    average=$(bc <<< "scale=2; $sum / $count")
    echo "Average: $average"
```

```
}
calculate_average "$@"
```

Example Usage:

./average.sh 10 20 30 40 50

Output:

Average: 30.00

65.Write a shell script to count the number of files and directories in a given directory.

```
#!/bin/bash
count_files_directories() {
    directory="$1"
    files=0
    directories=0
    for item in "$directory"/*; do
        if [ -f "$item" ]; then
            ((files++))
        elif [ -d "$item" ]; then
            ((directories++))
        fi
    done
    echo "Files: $files"
    echo "Directories: $directories"
```

```
}
count_files_directories "$1"
```

Example Usage:

./count.sh /path/to/directory

Output:

Files: 10 Directories: 5

66.Write a shell script to find the second largest number in a given list.

```
#!/bin/bash
find_second_largest() {
    largest=$1
    second_largest=$1
    for num in "$@"; do
        if [ $num -gt $largest ]; then
            second_largest=$largest
            largest=$num
        elif [ $num -gt $second_largest ] && [ $num -lt $largest ]; then
            second_largest=$num
        fi
    done
    echo "Second largest number: $second_largest"
}
```

```
find_second_largest "$@"
```

Example Usage:

./second_largest.sh 10 20 30 15 25

Output:

Second largest number: 25

67.Write a shell script to reverse a given string.

```
#!/bin/bash
reverse_string() {
    str="$1"
    reversed=""
    for ((i=${#str}-1; i>=0; i--)); do
        reversed="$reversed${str:$i:1}"
    done
    echo "$reversed"
}
reverse_string "$1"
```

Example Usage:

./reverse.sh "hello"

Output:

olleh

68.Write a shell script to sort numbers in descending order.

```
#!/bin/bash
 sort_descending() {
 printf "%s\n" "$@" | sort -nr
 }
sort_descending "$@"
```

Example Usage:

./sort_descending.sh 30 10 20 40 15

Output:

40 30 20 15 10

69.Write a shell script to find the factorial of a number using recursion.

```
#!/bin/bash
factorial() {
    n=$1
    if [ $n -eq 0 ] || [ $n -eq 1 ]; then
        echo "1"
    else
        echo "$((n * $(factorial $((n - 1)))))"
    fi
}
factorial "$1
```

Example Usage:

./factorial_recursive.sh 5

Output:

120

70.Write a shell script to print the Fibonacci series up to a given number.

```
#!/bin/bash
fibonacci() {
    n=$1
    a=0
    b=1
    echo "Fibonacci series up to $n:"
    echo -n "$a "
    while [ $b -le $n ]; do
        echo -n "$b "
        temp=$((a + b))
        a=$b
        b=$temp
    done
    echo
}
fibonacci "$1"
```

Example Usage:

./fibonacci.sh 20

Output:

Fibonacci series up to 20: 0 1 1 2 3 5 8 13

71. Write a shell script to count the number of lines, words, and characters in a file.

```
#!/bin/bash
count_lines_words_chars() {
    lines=$(wc -l < "$1")
    words=$(wc -w < "$1")
    characters=$(wc -m < "$1")
    echo "Lines: $lines"
    echo "Words: $words"
    echo "Characters: $characters"
}
count_lines_words_chars "$1"
```

Example Usage:

./count.sh file.txt

Output:

makefile

Lines: 10 Words: 50 Characters: 300

72.Write a shell script to check if a file is executable.

```
#!/bin/bash
check_executable() {
    file="$1"
    if [ -x "$file" ]; then
        echo "$file is executable"
    else
        echo "$file is not executable"
    fi
}
check_executable "$1"
```

Example Usage:

```
./check_executable.sh script.sh
```

Output:

```
script.sh is executable
```

73.Write a shell script to find the largest number in a given list.

```
#!/bin/bash
find_largest() {
    largest=$1
    for num in "$@"; do
        if [ $num -gt $largest ]; then
```

```
            largest=$num
        fi
    done
    echo "Largest number: $largest"
}
find_largest "$@"
```

Example Usage:

./find_largest.sh 10 20 30 15 25

Output:

Largest number: 30

74. Write a shell script to find the square root of a given number.

```
#!/bin/bash
find_square_root() {
    awk "BEGIN { printf \"Square root: %.2f\\n\", sqrt($1) }"
}
find_square_root "$1"
```

Example Usage:

./square_root.sh 16

Output:

Copy code

Square root: 4.00

75.Write a shell script to find the maximum and minimum of numbers in a given list.

```
#!/bin/bash
find_max_min() {
    max=$1
    min=$1
    for num in "$@"; do
        if [ $num -gt $max ]; then
            max=$num
        fi
        if [ $num -lt $min ]; then
            min=$num
        fi
    done
    echo "Maximum: $max"
    echo "Minimum: $min"
}
find_max_min "$@"
```

Example Usage:

./find_max_min.sh 10 20 30 15 25

Output:

Maximum: 30 Minimum: 10

76.Write a shell script to find the ASCII value of a given character.

```
#!/bin/bash
find_ascii_value() {
    echo -n "ASCII value of '$1': "
    printf '%d\n' "'$1"
}
find_ascii_value "$1"
```

Example Usage:

./ascii_value.sh A

Output:

ASCII value of 'A': 65

77. Write a shell script to convert a decimal number to binary.

```
#!/bin/bash
decimal_to_binary() {
    echo "Binary representation of $1: $(bc <<< "obase=2; $1")"
}
decimal_to_binary "$1"
```

Example Usage:

./decimal_to_binary.sh 10

Output:

Binary representation of 10: 1010

78.Write a shell script to find the common elements between two arrays.

```
#!/bin/bash
find_common_elements() {
    declare -a array1=("${!1}")
    declare -a array2=("${!2}")
    common=()
    for element in "${array1[@]}"; do
        if [[ " ${array2[@]} " =~ " $element " ]]; then
            common+=("$element")
        fi
    done
    echo "Common elements: ${common[@]}"
}
array1=("apple" "banana" "orange")
array2=("banana" "orange" "grape")
find_common_elements array1[@] array2[@]
```

Example Usage:

./common_elements.sh

Output:

Common elements: banana orange

79.Write a shell script to find the difference between two arrays.

```
#!/bin/bash
find_array_difference() {
    declare -a array1=("${!1}")
    declare -a array2=("${!2}")
    difference=()
    for element in "${array1[@]}"; do
        if [[ ! " ${array2[@]} " =~ " $element " ]]; then
            difference+=("$element")
        fi
    done
    echo "Difference: ${difference[@]}"
}
array1=("apple" "banana" "orange")
array2=("banana" "orange" "grape")
find_array_difference array1[@] array2[@]
```

Example Usage:

./array_difference.sh

Output:

Difference: apple

80.Write a shell script to print the multiplication table of a given number.

```
#!/bin/bash
multiplication_table() {
    num=$1
    for ((i=1; i<=10; i++)); do
        echo "$num x $i = $((num * i))"
    done
}
multiplication_table "$1"
```

Example Usage:

./multiplication_table.sh 7

Output:

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

7 x 4 = 28

7 x 5 = 35

7 x 6 = 42

7 x 7 = 49

7 x 8 = 56

7 x 9 = 63

7 x 10 = 70

81.Write a shell script to count the number of occurrences of a word in a file.

```
#!/bin/bash
count_occurrences() {
    word="$1"
    file="$2"
    occurrences=$(grep -o "\b$word\b" "$file" | wc -l)
    echo "Occurrences of '$word': $occurrences"
}
count_occurrences "$1" "$2"
```

Example Usage:

./count_occurrences.sh "apple" file.txt

Output:

Occurrences of 'apple': 3

82.Write a shell script to find the ASCII value of a given character.

```
#!/bin/bash
find_ascii_value() {
    echo -n "ASCII value of '$1': "
    printf '%d\n' "'$1"
}
find_ascii_value "$1"
```

Example Usage:

./ascii_value.sh A

Output:

ASCII value of 'A': 65

83.Write a shell script to check if a given year is a leap year or not.

```
#!/bin/bash
is_leap_year() {
    local year=$1
    if [ $((year % 4)) -eq 0 ] && [ $((year % 100)) -ne 0 ] || [ $((year % 400)) -eq 0 ]; then
        echo "$year is a leap year"
    else
        echo "$year is not a leap year"
    fi
}
is_leap_year "$1"
```

Example Usage:

./leap_year.sh 2024

Output:

2024 is a leap year

84.Write a shell script to calculate the area of a circle.

```
#!/bin/bash
calculate_circle_area() {
    local radius=$1
    local area=$(bc <<< "scale=2; 3.14159 * $radius * $radius")
    echo "Area of circle with radius $radius: $area"
}
calculate_circle_area "$1"
```

Example Usage:

```
./circle_area.sh 5
```

Output:

```
Area of circle with radius 5: 78.54
```

85.Write a shell script to count the number of vowels and consonants in a given string.

```
#!/bin/bash
count_vowels_consonants() {
    local string="$1"
    local vowels=$(echo "$string" | tr -dc 'aeiouAEIOU' | wc -c)
    local consonants=$(echo "$string" | tr -dc 'bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ' | wc -c)
    echo "Vowels: $vowels"
    echo "Consonants: $consonants"
}
```

```
count_vowels_consonants "$1"
```

Example Usage:

```
./vowels_consonants.sh "hello world"
```

Output:

```
Vowels: 3 Consonants: 7
```

86.Write a shell script to check if a given string is palindrome.

```
#!/bin/bash
is_palindrome_string() {
    local string="$1"
    local reversed=$(echo "$string" | rev)
    if [ "$string" == "$reversed" ]; then
        echo "$string is a palindrome string"
    else
        echo "$string is not a palindrome string"
    fi
}
is_palindrome_string "$1"
```

Example Usage:

```
./palindrome_string.sh "madam"
```

Output:

```
madam is a palindrome string
```

87.Write a shell script to generate a random password.

```
#!/bin/bash
generate_password() {
    local length=$1
    tr -dc 'A-Za-z0-9' < /dev/urandom | head -c $length
    echo
}
generate_password "$1"
```

Example Usage:

./random_password.sh 8

Output:

qD3Lz9Wx

88.Write a shell script to check if a given string is an anagram of another string.

```
#!/bin/bash
is_anagram() {
    local str1="$1"
    local str2="$2"
    local sorted_str1=$(echo "$str1" | grep -o . | sort | tr -d '\n')
    local sorted_str2=$(echo "$str2" | grep -o . | sort | tr -d '\n')
    if [ "$sorted_str1" == "$sorted_str2" ]; then
        echo "$str1 and $str2 are anagrams"
```

```
    else
        echo "$str1 and $str2 are not anagrams"
    fi
}
is_anagram "$1" "$2"
```

Example Usage:

./anagram.sh "listen" "silent"

Output:

listen and silent are anagrams

89.Write a shell script to generate a random number within a specified range.

```
#!/bin/bash
generate_random_number() {
    local min=$1
    local max=$2
    echo $((RANDOM % (max - min + 1) + min))
}
generate_random_number "$1" "$2"
```

Example Usage:

./random_number.sh 1 100

Output:

(random number within the range)

90.Write a shell script to find the sum of all even numbers and the sum of all odd numbers in a given list.

```
#!/bin/bash
sum_even_odd() {
    local sum_even=0
    local sum_odd=0
    for num in "$@"; do
        if [ $((num % 2)) -eq 0 ]; then
            sum_even=$((sum_even + num))
        else
            sum_odd=$((sum_odd + num))
        fi
    done
    echo "Sum of even numbers: $sum_even"
    echo "Sum of odd numbers: $sum_odd"
}
sum_even_odd "$@"
```

Example Usage:

```bash
```

./sum_even_odd.sh 1 2 3 4 5
```

**Output:**
```
Sum of even numbers: 6
Sum of odd numbers: 9

91. Write a shell script to find the area of a rectangle.

```
#!/bin/bash
calculate_rectangle_area() {
    local length=$1
    local width=$2
    local area=$((length * width))
    echo "Area of rectangle: $area"
}
calculate_rectangle_area "$1" "$2"
```
****Example Usage:****
./rectangle_area.sh 5 8

****Output**:**
Area of rectangle: 40

92.Write a shell script to find the area of a triangle.

```
#!/bin/bash
calculate_triangle_area() {
    local base=$1
    local height=$2
    local area=$(bc <<< "scale=2; 0.5 * $base * $height")
    echo "Area of triangle: $area"
}
calculate_triangle_area "$1" "$2"
```

****Example Usage:****

```
./triangle_area.sh 6 8
```

****Output:****

```
Area of triangle: 24.00
```

93.Write a shell script to print the elements of an array in reverse order.

```
#!/bin/bash
print_array_reverse() {
    local array=("$@")
    for ((i=${#array[@]}-1; i>=0; i--)); do
        echo -n "${array[i]} "
    done
    echo
}
```

print_array_reverse "$@"

****Example Usage:****

./array_reverse.sh 1 2 3 4 5

****Output:***

5 4 3 2 1

94. Explain the purpose of the shift command in shell scripting.

- The **shift** command is used to shift command line arguments one position to the left. This is useful when processing command line arguments in a loop. After shifting, **$2** becomes **$1**, **$3** becomes **$2**, and so on, while **$1** is discarded.

95.What is the purpose of the exec command in shell scripting?

- The **exec** command is used to execute a command in place of the current shell without creating a new process. It is often used for file redirection and changing process environments. Once **exec** is used, the current shell process is replaced by the specified command.

96. Explain the purpose of the continue statement in shell scripting.

- The **continue** statement is used to skip the current iteration of a loop and continue with the next iteration. It is often used to skip certain iterations based on specific conditions, without terminating the loop.

97. Explain the purpose of the break statement in shell scripting.

- The **break** statement is used to exit from a loop prematurely. It is often used to terminate a loop based on a certain condition, allowing the script to continue executing from the statement following the loop.

98.What is the purpose of the case statement in shell scripting?

- The **case** statement is used for multiway branching in shell scripting. It allows you to match the value of a variable against various patterns and execute different commands based on the match. It provides an alternative to nested **if-elif-else** statements for handling multiple conditions.

99.Explain the significance of the here documents in shell scripting.

- Here documents (heredocs) are a way of passing multiple lines of input to a command in a script. They allow you to embed a block of text within a script without having to use echo or cat commands. Heredocs are often used for providing input to commands or for creating configuration files dynamically within scripts.

100.What are the differences between single quotes (' ') and double quotes (" ") in shell scripting?

- In shell scripting, single quotes and double quotes have different behaviors:
    - Single quotes preserve the literal value of each character enclosed within them. Variables and command substitutions within single quotes are not expanded.
    - Double quotes allow for variables and command substitutions to be expanded. However, they preserve the whitespace within them.

101.Explain the purpose of the basename command in shell scripting.

- The **basename** command strips directory and suffix from filenames. It extracts the last component of a pathname. It is often used to extract filenames from paths.

102.Explain the purpose of the dirname command in shell scripting.

- The **dirname** command extracts the directory portion of a pathname and prints it. It removes the last component of a pathname, leaving only the directory part.

103.What are shell variables?

- Shell variables are placeholders used to store data or values. They are used to store strings, numbers, filenames, or any other data. Shell variables can be created, assigned values, and accessed throughout a shell script.

104.What are environment variables?

- Environment variables are special shell variables that are inherited by all child processes of the shell. They are typically used to pass information from the shell to programs invoked by the shell. Environment variables are accessible by all processes running under the shell.

105.Explain the purpose of the printf command in shell scripting.

- The **printf** command is used to format and print data to standard output. It provides more control over the formatting compared to the **echo** command. **printf** allows you to specify the format of the output using format specifiers such as **%s**, **%d**, **%f**, etc.

106.What is process substitution in shell scripting?

- Process substitution is a form of substitution that allows you to use the output of a command or a shell script as input to another command. It is denoted by **<()** or **>()** syntax. Process substitution is useful for cases where input redirection or pipes are not sufficient.

107.How to create a Softlink?

```
    # Create a symbolic link to a file
target_file="/path/to/target_file"
link_name="/path/to/link_name"

# Check if link already exists
if [ ! -e "$link_name" ]; then
    ln -s "$target_file" "$link_name"
    echo "Symbolic link created."
else
    echo "Symbolic link already exists."
fi
```

Sample Output:

Symbolic link created.

or

Symbolic link already exists.

108. How to create Hardlink?

```
# Create a hard link to an existing file
existing_file="/path/to/existing_file"
new_link="/path/to/new_link"

# Check if link already exists
if [ ! -e "$new_link" ]; then
    ln "$existing_file" "$new_link"
    echo "Hard link created."
else
    echo "Hard link already exists."
fi
```

Sample Output:

Hard link created.

or

Hard link already exists.

109.How to create a cron job?

```
# Add a cron job to run a script every day at 2:00 AM
cronjob_exists=$(crontab -l | grep "/path/to/your_script.sh")

if [ -z "$cronjob_exists" ]; then
    echo "0 2 * * * /path/to/your_script.sh" | crontab -
    echo "Cron job added."
else
    echo "Cron job already exists."
fi
```

Sample Output:

Cron job added.

or

Cron job already exists.