shall scripting basics:

Detailed Answers to 100 Shell Scripting Basics

1. **What is a shell script?**
   - A shell script is a text file containing a series of commands that are interpreted and executed by a shell, such as Bash or Sh.
2. **How do you run a shell script?**
   - Make the script executable using `chmod +x script.sh` and then run it using `./script.sh`.
3. **What is the role of `#!` (shebang) in a script?**
   - The `#!` at the beginning of a script specifies the path to the interpreter that should execute the script (e.g., `#!/bin/bash`).
4. **Basic structure of a shell script**
   - A typical shell script starts with the shebang line, followed by comments and commands in a logical sequence.
5. **How to write comments in shell scripts?**
   - Use `#` for single-line comments. Multi-line comments can be simulated with `<<COMMENT ... COMMENT`.
6. **Declaring variables in shell scripts**
   - Variables are declared as `variable_name=value` without spaces. To use them, prefix with `$` (e.g., `echo $variable_name`).
7. **What is the difference between `$var` and `${var}`?**
   - `${var}` helps avoid ambiguity when variables are next to characters or words.
8. **Using `echo` command for output**
   - The `echo` command prints text or variable values to the terminal (e.g., `echo "Hello World"`).
9. **Reading input from the user**
   - Use the `read` command (e.g., `read name`).
10. **Basic arithmetic operations**
    - Use `expr` or `$(( ))` for arithmetic (e.g., `result=$(expr 3 + 4)` or `result=$((3 + 4))`).
11. **Conditional `if` statements**

-
  ```
  if [ condition ]; then
  # Commands if true
  fi
  ```

12. **Using `elif` and `else` in conditions**

-
  ```
  if [ condition ]; then
  # Commands if true
  elif [ condition ]; then
  # Another condition
  else
  # Commands if false
  fi
  ```

13. **`case` statements for pattern matching**

-
  ```
  case $var in
  pattern1)
  # Commands
  ;;
  pattern2)
  # Commands
  ;;
  esac
  ```

14. **Looping with `for`**

-
  ```
  for i in {1..5}; do
  echo "Iteration $i"
  done
  ```

15. **Using `while` loops**

-
  ```
  while [ condition ]; do
  # Commands
  done
  ```

16. **Using `until` loops**

-
  ```
  until [ condition ]; do
  # Commands
  done
  ```

17. **Infinite loops and their control**
    - Infinite loops run with `while true; do ... done` and can be controlled with `break`.

18. **`break` and `continue` statements**
    - `break` exits a loop, while `continue` skips to the next iteration.

19. **Defining functions in a shell script**

-
  ```
  my_function() {
  echo "Hello from the function"
  }
  ```

20. **Calling functions in scripts**
    - Call a function by its name (e.g., `my_function`).
21. **Passing arguments to functions**

-
  ```
  my_function() {
  echo "Argument 1: $1"
  }
  my_function "Hello"
  ```

22. **File test operators (`-e`, `-d`, `-f`)**
    - `-e` checks if a file exists, `-d` checks if it's a directory, `-f` checks if it's a regular file.
23. **String comparison**
    - Use `=` and `!=` for equality checks (e.g., `[ "$str1" = "$str2" ]`).
24. **Numeric comparison (`-eq`, `-ne`)**
    - Use `-eq` for equality, `-ne` for not equal (e.g., `[ "$num1" -eq "$num2" ]`).
25. **Logical operators (`&&`, `||`)**
    - `&&` executes the second command if the first succeeds; `||` if it fails.
26. **Command substitution (`$()` or `` ` ``)**
    - Capture output with `$()` (preferred) or backticks (e.g., `output=$(ls)`).
27. **Redirecting output to files**
    - Use `>` to overwrite and `>>` to append (e.g., `echo "text" > file.txt`).
28. **Redirecting input from files**
    - Use `<` (e.g., `command < file.txt`).
29. **Pipes (`|`)**
    - Combine commands by passing output as input to the next (e.g., `ls | grep "txt"`).
30. **Using `grep` for pattern matching**
    - `grep "pattern" file.txt` searches for matching lines in a file.