Bash Scripting

1. Introduction to Bash Scripting

- **Shell Script**: A text file containing Unix/Linux commands.
- **Shebang (`#!`)**: Indicates the interpreter to use for the script.
  - Example: `#!/bin/bash`
- **Why write scripts?**
  - Automate repetitive tasks.
  - Simplify complex processes.

---

2. Steps to Write a Bash Script

1. **Create a script** using a text editor (`vi`, `gedit`, etc.).
2. **Add Shebang**: `#!/bin/bash`.
3. **Make script executable**:

   ```
   bash

   chmod +x scriptname.sh
   ```

4. **Run the script**:

   ```
   bash

   ./scriptname.sh
   ```

---

3. Variables in Bash

- **Create a variable**:

  ```
  bash

  varname=value   # No spaces
  ```

- **Access a variable**:

  ```
  bash

  echo $varname
  ```

- **Types of Variables**:
  - **Global Variables**: System-defined, e.g., `PATH`, `HOME`.
  - **Local Variables**: User-defined, limited to a script.
  - **Special Variables**:
    - `$0`: Script name.
    - `$1`, `$2`…: Positional parameters.
    - `$#`: Number of parameters.
    - `$$`: Process ID of script.

---

4. Conditional Statements

- **if Statement**:

```
bash

if [ condition ]; then
    commands
fi
```

- **if-else**:

```
bash

if [ condition ]; then
    commands
else
    other_commands
fi
```

- **elif**:

```
bash

if [ condition ]; then
    commands
elif [ other_condition ]; then
    other_commands
fi
```

- **Comparison Operators**:
    - **Integers**: `-eq`, `-ne`, `-lt`, `-le`, `-gt`, `-ge`.
    - **Strings**: `=`, `!=`, `-z` (is empty), `-n` (is not empty).

---

5. Loops

1. **For Loop**:

```
bash

for i in 1 2 3; do
    echo $i
done
```

2. **While Loop**:

```
bash

i=1
while [ $i -le 5 ]; do
    echo $i
    i=$((i+1))
done
```

3. **Until Loop**:

```bash
bash

i=1
until [ $i -gt 5 ]; do
    echo $i
    i=$((i+1))
done
```

---

6. Functions

- **Define a function**:

```bash
bash

my_function() {
    echo "This is a function"
}
my_function
```

---

7. Input/Output Redirection

- **Redirect Output**:
  - `>`: Overwrite file.
  - `>>`: Append to file.
- **Redirect Input**:
  - `<`: Input from file.
- **Example**:

```bash
bash

ls > output.txt
```

---

8. Special Topics

- **Test Conditions**:
  - File tests: `-e` (exists), `-d` (directory), `-f` (file), `-x` (executable).
- **Quoting**:
  - **Double quotes**: `$variables` are expanded.
  - **Single quotes**: Everything is literal.
  - **Backticks**: Command substitution.

Hands-on Examples

1. **Hello World**:

```bash
bash

echo "Hello World"
```

2. **Factorial of a Number**:

   ```
   bash

   factorial=1
   for i in `seq 1 5`; do
       factorial=$((factorial * i))
   done
   echo "Factorial: $factorial"
   ```

Linux Session, Streams, Pipes & Filters

1. Linux Sessions and Process Groups

- **Session**: A group of process groups controlled by a terminal.
- **Commands**:
  - `ps`: View processes.
  - `jobs`: Show background jobs.

---

2. Standard I/O Streams

- **Standard Input (`stdin`)**: Keyboard, file descriptor `0`.
- **Standard Output (`stdout`)**: Terminal, file descriptor `1`.
- **Standard Error (`stderr`)**: Error messages, file descriptor `2`.

---

3. Redirection

- **Output Redirection**:
  - `>`: Redirect output to a file.
  - `>>`: Append output to a file.
- **Input Redirection**:
  - `<`: Take input from a file.

Example:

```
bash

ls > file.txt     # Write output to file
sort < file.txt   # Read input from file
```

---

4. Pipes

- **Pipe (`|`)**: Connect the output of one command to the input of another.
  - Syntax:

    ```
    bash
    ```

```
command1 | command2
```

- **Example**:

  ```
  bash
  cat file.txt | sort | uniq
  ```

---

5. Filters

- Filters are commands that process input and provide output.

1. **grep**: Search for a pattern.

   ```
   bash
   grep "text" file.txt
   ```

2. **uniq**: Remove duplicates.

   ```
   bash
   uniq file.txt
   ```

3. **sort**: Sort input.

   ```
   bash
   sort file.txt
   ```

4. **wc**: Count words, lines, or characters.

   ```
   bash
   wc -l file.txt   # Count lines
   ```

5. **tr**: Translate characters.

   ```
   bash
   tr 'a-z' 'A-Z' < file.txt
   ```

---

6. Combining Commands

- Use pipelines to combine filters:

  ```
  bash
  grep "error" logfile.txt | sort | uniq -c | sort -nr
  ```

---

7. Here Document (<<)

- Used to redirect multiple lines of input.

```
bash

cat << EOF
Line 1
Line 2
EOF
```

---

8. Special Topics

- **Compression**:
  - `gzip`: Compress files.
- **Email**:

```
bash

cat file.txt | mail -s "Subject" user@example.com
```