Basic Scripting in Bash

Bash scripting allows you to automate tasks and write powerful scripts using the Linux command line.

Creating and Running a Script

1. **Create a Script File:** Use a text editor to create a file with the `.sh` extension.

   ```
   nano script.sh
   ```

2. **Add the Shebang Line:** At the top of the file, include the shebang (`#!`) to specify the script interpreter:

   ```
   #!/bin/bash
   ```

3. **Make the Script Executable:** Use the `chmod` command to give execute permissions:

   ```
   chmod +x script.sh
   ```

4. **Run the Script:** Execute the script by specifying its path:

   ```
   ./script.sh
   ```

Basic Script Structure

Here's an example of a simple script:

```
#!/bin/bash

# Print a message
echo "Hello, World!"

# Define a variable
NAME="User"
echo "Welcome, $NAME!"

# Use a conditional statement
if [ "$NAME" = "User" ]; then
    echo "Default user detected."
else
    echo "Custom user: $NAME"
fi

# Loop through numbers
for i in {1..5}; do
    echo "Number: $i"
done
```

Common Features

Variables

Assign values to variables and reference them using `$`:

```
VAR="value"
echo $VAR
```

Conditionals

Use `if`, `elif`, and `else` for conditional logic:

```
if [ condition ]; then
    # Commands
elif [ other_condition ]; then
    # Commands
else
    # Commands
fi
```

Example:

```
if [ -f "file.txt" ]; then
    echo "file.txt exists"
else
    echo "file.txt does not exist"
fi
```

Loops

`for` Loop:

```
for item in list; do
    # Commands
done
```

Example:

```
for file in *.txt; do
    echo "Processing $file"
done
```

`while` Loop:

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

Example:

```
COUNT=1
while [ $COUNT -le 5 ]; do
    echo "Count: $COUNT"
    ((COUNT++))
done
```

Functions

Define reusable blocks of code:

```
function_name() {
    # Commands
}
```

Example:

```
greet() {
    echo "Hello, $1!"
}
```

```
greet "World"
```

Debugging

Run the script with debugging enabled:

```
bash -x script.sh
```

Add `set -x` in the script to enable debugging for specific sections:

```
set -x
# Debugging commands
set +x
```

Comments

Use `#` to add comments:

```
# This is a comment
```

Useful Commands

- **`read`**: Get user input.

  ```
  echo "Enter your name:"
  read NAME
  echo "Hello, $NAME!"
  ```

- **`exit`**: Exit the script with a status code.

  ```
  exit 0
  ```

- **`date`**: Get the current date and time.

  ```
  echo "Today is $(date)"
  ```

With these basics, you can start creating powerful Bash scripts to automate repetitive tasks and enhance your workflow.