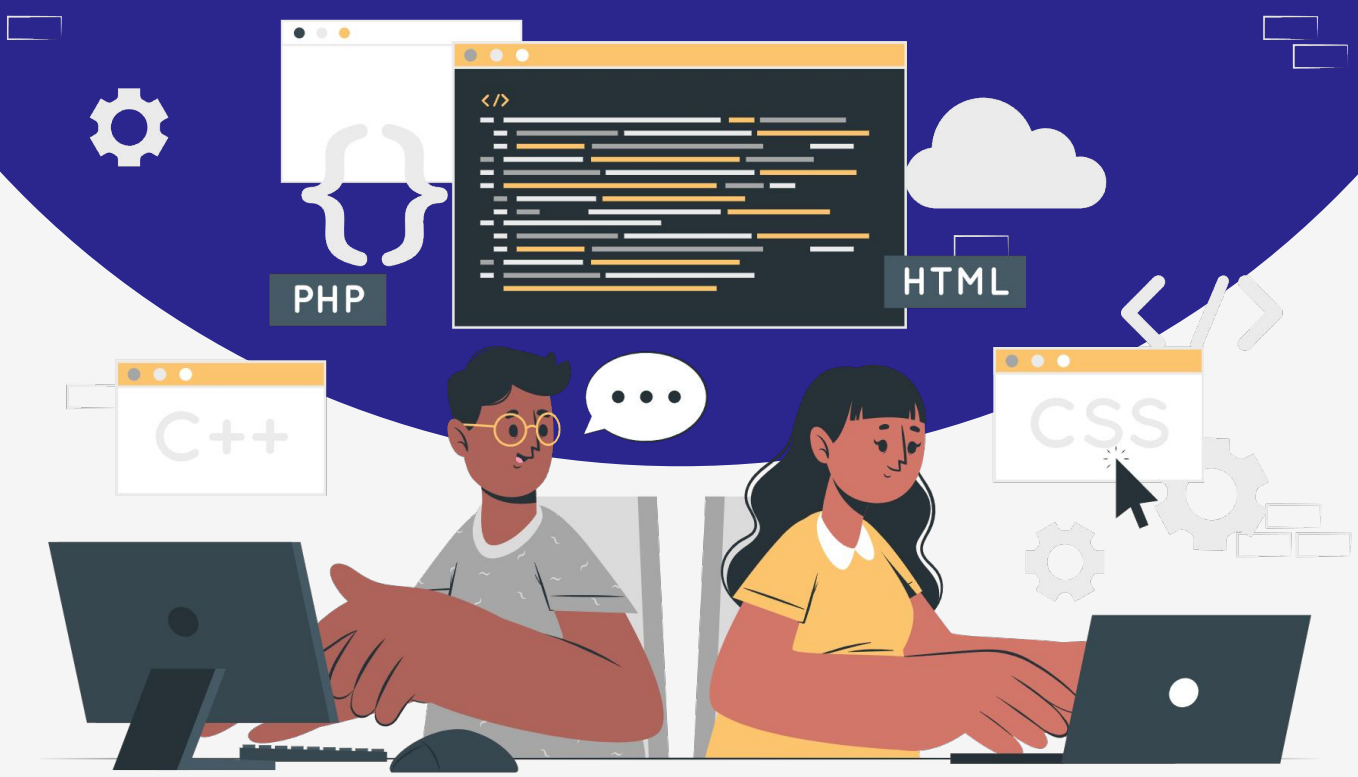
Lesson Plan

Best practices for Bash Scripting



PW SKILLS

1. Use Descriptive Variable Names

- Choose clear, meaningful names for variables to make your script easier to understand.

Example:

```
backup_directory="/backup/data"
```

2. Comment your Code

- Add comments to explain what your script or specific commands are doing, especially for complex logic.

Example:

```
# Check if the backup directory exists
```

3. Use set -e for Error Handling

- Use set -e to make your script exit immediately if a command fails, preventing unintended consequences.

Example:

```
set -e  # Exit on any error
```

4. Check for Dependencies

- Ensure that any external commands your script relies on are available. This avoids runtime errors.

Example:

```
if ! command -v curl &> /dev/null; then
  echo "curl could not be found"
  exit 1
fi
```

5. Use Functions to Organize code

- Break your script into functions to make it modular and easier to manage or reuse.

Example:

```
backup_files() {
  # Code to backup files
}

restore_files() {
  # Code to restore files
}
```

6. Handle Input and Output Carefully

- Use read for user input and validate it.
- Redirect outputs to log files or handle errors with 2>&1 to avoid cluttered terminal output.

Example:

```
read -p "Enter a filename: " filename
echo "Processing file: $filename" >> script.log
```

7. Use Exit Status Codes

- Return appropriate exit status codes (0 for success, non-zero for errors) to indicate the script's success or failure.

Example:

```
if [ some_error_condition ]; then
  echo "Error occurred"
  exit 1
fi
exit 0  # Script executed successfully
```

8. Make Your Script Portable

- Write your script to be as portable as possible, avoiding platform-specific features unless necessary.
- Use #!/bin/bash at the beginning of your script to ensure it runs in Bash.