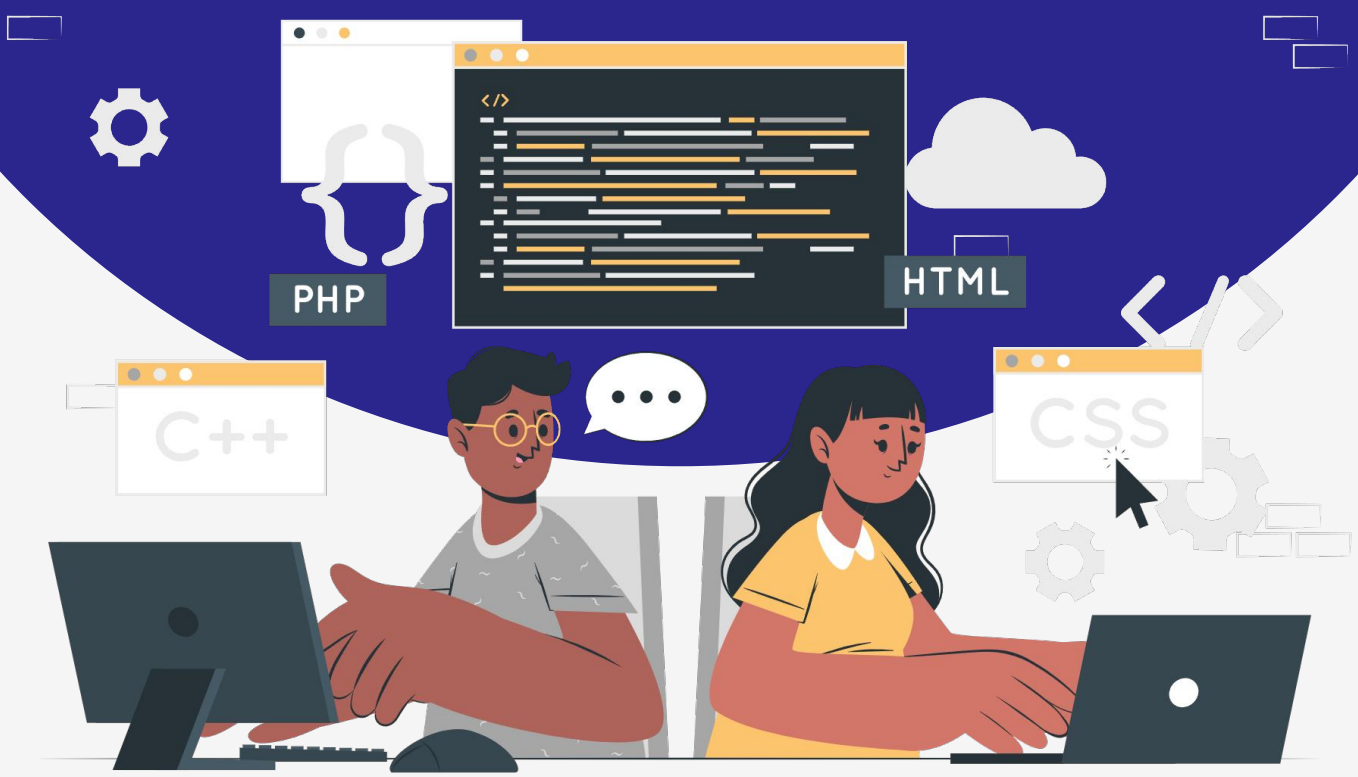
Lesson Plan

Automating Tasks with Scripts for system administration



PW SKILLS

Automating Tasks with Scripts for system administration

Automating tasks with scripts in Bash is a powerful way to manage system administration tasks. Bash scripting can help automate repetitive tasks, manage system configurations, perform backups, and monitor systems. Here's an overview of how you can use Bash scripting for system administration:

Basics of Bash Scripting

- **Shebang:** Start your script with #!/bin/bash to specify that the script should be run in the Bash shell.
- **Variables:** Assign values to variables using = without spaces (e.g., VARIABLE=value).
- **Comments:** Use # for single-line comments.

1. Automating Repetitive Tasks

- Loops: Use for, while, or until loops to perform repetitive tasks.

```
for i in {1..10}; do
  echo "This is loop number $i"
done
```

- Conditionals: Use if, elif, and else for decision-making.

```
if [ -f /path/to/file ]; then
  echo "File exists"
else
  echo "File does not exist"
fi
```

2. System Monitoring

- Check Disk Usage: Monitor disk usage and send alerts if it exceeds a threshold.

```
THRESHOLD=80
USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')
if [ $USAGE -gt $THRESHOLD ]; then
  echo "Disk usage is above $THRESHOLD%"
fi
```

- Process Monitoring: Check if a specific process is running and restart it if it's not.

```
if ! pgrep -x "process_name" > /dev/null; then
  /path/to/process &
fi
```

3. Backup Automation

- Automated Backups: Create a backup of important files or directories.

```
BACKUP_DIR="/backup"
SOURCE_DIR="/important/files"
DATE=$(date +%F)
tar -czf "$BACKUP_DIR/backup-$DATE.tar.gz" "$SOURCE_DIR"
```

4. User Management

- Create Users in Bulk: Add multiple users from a file.

```
while IFS=, read -r username password; do
  useradd -m -p "$(openssl passwd -1 $password)" "$username"
done < users.csv
```

5. System Updates

- Automate System Updates: Automate the process of updating the system.

```
sudo apt-get update -y
sudo apt-get upgrade -y
```

6. Logging & Notifications

- Logging: Redirect script output to a log file for later review.

```
exec > /var/log/script.log 2>&1
```

- Email Alerts: Send an email alert if something goes wrong.

```
if ! pgrep -x "process_name" > /dev/null; then
  echo "Process not running!" | mail -s "Alert: Process down" admin@example.com
fi
```

7. Scheduling with Cron

Use cron to schedule your Bash scripts to run at specific intervals.

- Edit cron jobs with crontab -e.
- Example of a cron job that runs a script daily at midnight:

```
0 0 * * * /path/to/script.sh
```

8. Security Considerations

- Permissions: Ensure scripts have the correct permissions (e.g., chmod +x script.sh).
- Environment Variables: Be cautious with sensitive data in scripts, like passwords.

9. Error Handling

- Exit Status: Check the exit status of commands to detect errors.

```
cp /source/file /destination/ || { echo "Copy failed"; exit 1; }
```

- Trap Errors: Use trap to catch errors and execute cleanup code.

```
trap 'echo "An error occurred"; exit 1' ERR
```

PW SKILLS

THANK YOU !