Lesson Plan

Background running processes

Background running processes

Running background processes in scripting is a common practice to improve efficiency and manage long-running or concurrent tasks without blocking the main script. Here's a deeper look into handling background processes in Bash scripting:

Basic Concepts:

1. Starting a Background Process:

- Use & at the end of a command to run it in the background.

Example:

```
command &
```

- **The script will immediately proceed to the next line without waiting for 'command' to finish.**

2. Process IDs (PIDs) and Job Control:

- When you run a command in the background, it gets a unique Process ID (PID).
- You can retrieve the PID of the last background process using $!.

Example:

```
command &
pid=$!
echo "Background process PID: $pid"
```

3. Using jobs, fg, and bg:

- **jobs: Lists all background jobs with their job IDs.**
- **fg: Brings a background job to the foreground.**
- **bg: Resumes a paused background job.**

Example:

```
command &
jobs      # List background jobs
fg %1     # Bring job 1 to the foreground
```

4. Redirecting Output:

- To prevent the background process from cluttering the terminal with output, redirect its output to a file or /dev/null.

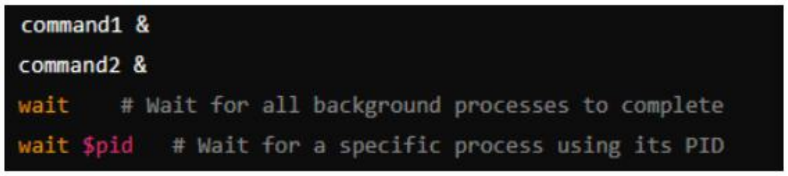
Example:

```
command > output.log 2>&1 &
```

5. Using wait:

- The wait command pauses the script until all background processes have finished.
- You can also wait for a specific background process by passing its PID to wait.

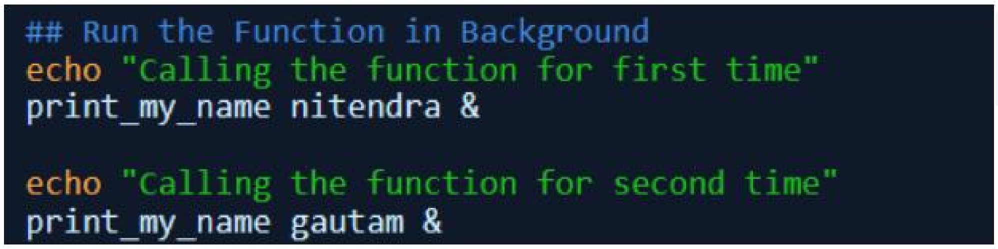
Example:



```
command1 &
command2 &
wait     # Wait for all background processes to complete
wait $pid    # Wait for a specific process using its PID
```

Running the function or command in background

We can execute a command in the background by placing an ampersand (&) symbol at the end of the command. When we place a job in the background, a user job number and system process number or ID are displayed in the terminal.

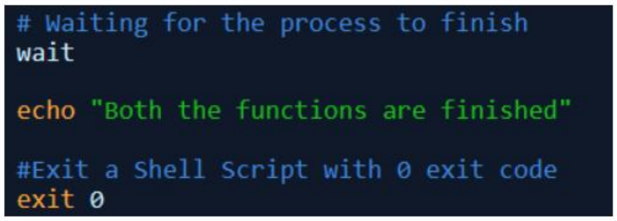


```
## Run the Function in Background
echo "Calling the function for first time"
print_my_name nitendra &

echo "Calling the function for second time"
print_my_name gautam &
```

The "&" symbol at the end of the function instructs bash to run print_my_name function in the background.

Wait for the Background function to finish

We use the wait command to wait for the background process to finish. Let's wait for both the process to finish and print the exit code.



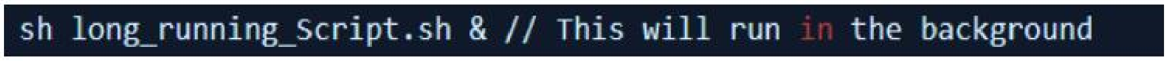
```
# Waiting for the process to finish
wait

echo "Both the functions are finished"

#Exit a Shell Script with 0 exit code
exit 0
```

Running the Shell script in the background

We can use the ampersand(&) or nohup to run a shell script in the background.

ampersand(&)

The "&" symbol at the end of the script instructs bash to run that script in the background.



```
sh long_running_Script.sh & // This will run in the background
```

Nohup Command

Nohup is a command used to run a long-running process (job) on a server. This command executes another command and instructs the system to continue running it even if the session is disconnected.

It is present on all the Unix compute servers. Its syntax is given below.

```
nohup command [command-argument ...]
nohup <command> &
```

Replace < command > with the name of your executable or that of the script we want to execute.

```
nohup long_running_Script.sh &
```

When we use &, we will see the bash job ID in brackets, and the process ID (PID) listed after.

```
~ % nohup ls -ltr &
[3] 27631
```

We can use the PID to terminate the process prematurely. We can use the kill command to terminate this process if needed.

```
~ % kill -9 27631
```

This is the advantage nohup has over the direct use of ampersand(&).

PW SKILLS

THANK YOU !