SKILLS | DevOps and Cloud Computing

Advanced Bash Scripting

Objective

- Manage and monitor background processes in Linux.
- Automate system administration tasks using Bash scripts.
- Perform file handling, string manipulation, and file compression with Bash commands.
- Use command substitution and process substitution for efficient script execution.

PW SKILLS

Explaining foreground vs. background processes in Linux.

Foreground vs Background

In Linux, foreground processes require direct user interaction and tie up the terminal until they finish. Background processes, on the other hand, run independently, freeing up the terminal for other tasks.

Key differences:

1. Interaction:

- Foreground processes connect to the terminal, allowing communication with the user via the screen and keyboard.
- Background processes disconnect from the terminal and typically don't require user interaction.

Foreground vs Background

2. Execution:

- By default, programs run as foreground processes.
- To run a process in the background, add an ampersand (&) at the end of the command.

3. Management:

- You can suspend a foreground process with Ctrl+Z and move it to the background using the bg command.
- The jobs command lists background processes started from the current shell.

Foreground vs Background

4. Terminal Connection:

- Background processes are not detached from your terminal. Closing the terminal may kill background jobs unless nohup is used.

5. Processing:

- The processor schedules foreground and background processes in the same way. The main difference is whether the shell waits for the process to finish.

Demonstrating running commands in the background using ‘&’

Let’s do it

- To run a command in the background in Linux, append an ampersand (&) to the end of the command. This tells the shell to execute the command as a separate background process, immediately freeing up the terminal.
- **For example:**

```
vim &
```

- This will start the vim text editor in the background. The terminal will display the shell job ID and the process ID (PID) of the background command.

Let’s do it

- If the command produces output, it may interfere with the terminal prompt. To avoid this, redirect the output to a file or to /dev/null to discard it.

```
ping 8.8.8.8 > log.txt 2>&1 &
```

or

```
ping 8.8.8.8 > /dev/null 2>&1 &
```

PW SKILLS

Monitoring background jobs using jobs and bring them back with fg or bg.

How to Monitor background jobs

To monitor background jobs in Linux, use the jobs command.

- This command displays a list of all jobs running in the current shell, including their job number, state (running, stopped, or terminated), and the command that started them.

```
jobs
```

- To bring a background job to the foreground, use the fg command followed by the job number.

```
fg %<job_number>
```

How to Monitor background jobs

- To resume a stopped job in the background, use the bg command followed by the job number.

```
bg %<job_number>
```

- For example, if jobs shows a job with the number 1, you would use fg %1 to bring it to the foreground or bg %1 to run it in the background.

- The + and - symbols next to job numbers in the jobs output indicate the most recent and the previous jobs, respectively.

Quiz 1

Q. What information does the jobs command provide?

A

A list of all processes on the system.

B

The job number, state (running, stopped, or terminated), and the command that started the job

Quiz 1

Q. What information does the jobs command provide?

A

A list of all processes on the system.

B

The job number, state (running, stopped, or terminated), and the command that started the job

Quiz 1

Q. What does the command fg do without a job number?

A

It brings all background jobs to the foreground

B

It brings the most recent job to the foreground.

Quiz 1

Q. What does the command fg do without a job number?

A

It brings all background jobs to the foreground

B

It brings the most recent job to the foreground.

PW SKILLS

Common system administration tasks that can be automated

Let’s Discuss

System administrators can automate many routine tasks to improve efficiency and reduce errors. Common examples include:

- Backup and Recovery
- Patching
- User Management
- Log Management and Analysis
- Server Provisioning
- Password resets
- Restarting services
- Remote shutdowns

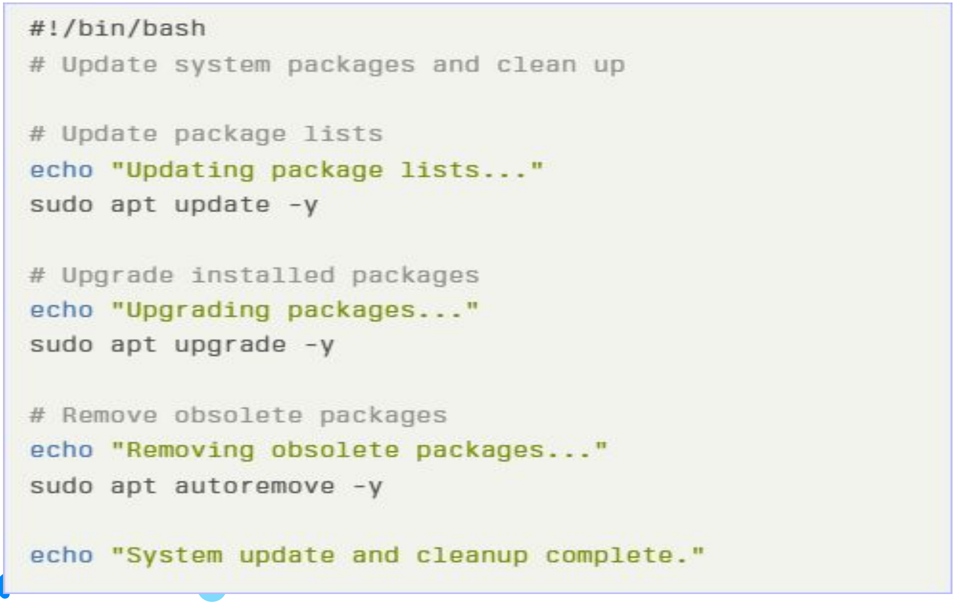
PW SKILLS

Demonstrating writing a bashscript for automating system updates or log rotation

Let’s do it

1. System Update Script

This script automates updating and upgrading packages on Debian/Ubuntu-based systems. It also cleans up obsolete packages.

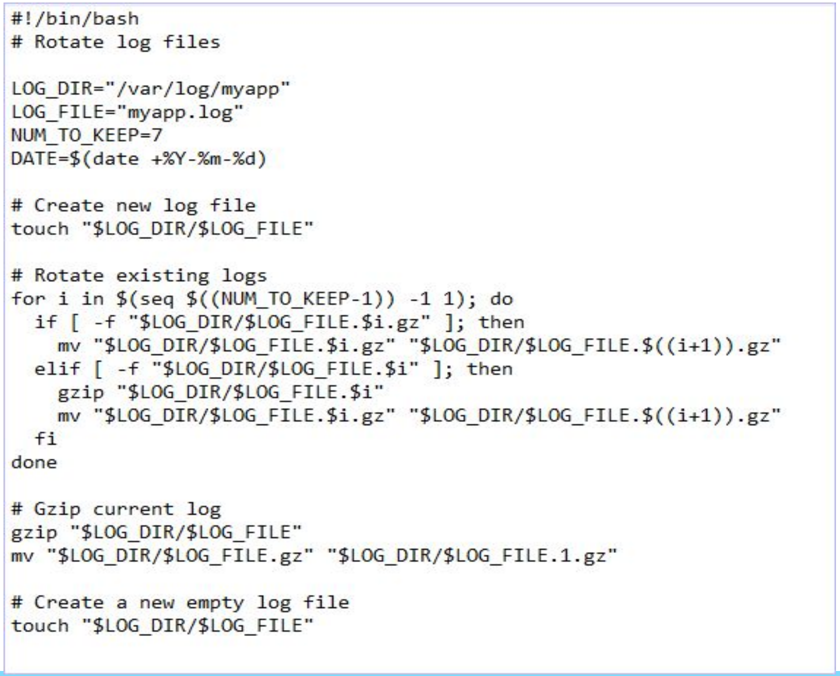


```bash
#!/bin/bash
# Update system packages and clean up

# Update package lists
echo "Updating package lists..."
sudo apt update -y

# Upgrade installed packages
echo "Upgrading packages..."
sudo apt upgrade -y

# Remove obsolete packages
echo "Removing obsolete packages..."
sudo apt autoremove -y

echo "System update and cleanup complete."
```

Let’s do it

2. Log Rotation Script

This script rotates log files, compressing the old ones and deleting the oldest.



```
#!/bin/bash
# Rotate log files

LOG_DIR="/var/log/myapp"
LOG_FILE="myapp.log"
NUM_TO_KEEP=7
DATE=$(date +%Y-%m-%d)

# Create new log file
touch "$LOG_DIR/$LOG_FILE"

# Rotate existing logs
for i in $(seq $((NUM_TO_KEEP-1)) -1 1); do
  if [ -f "$LOG_DIR/$LOG_FILE.$i.gz" ]; then
    mv "$LOG_DIR/$LOG_FILE.$i.gz" "$LOG_DIR/$LOG_FILE.$((i+1)).gz"
  elif [ -f "$LOG_DIR/$LOG_FILE.$i" ]; then
    gzip "$LOG_DIR/$LOG_FILE.$i"
    mv "$LOG_DIR/$LOG_FILE.$i.gz" "$LOG_DIR/$LOG_FILE.$((i+1)).gz"
  fi
done

# Gzip current log
gzip "$LOG_DIR/$LOG_FILE"
mv "$LOG_DIR/$LOG_FILE.gz" "$LOG_DIR/$LOG_FILE.1.gz"

# Create a new empty log file
touch "$LOG_DIR/$LOG_FILE"
```

PW SKILLS

Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS

File Handling Commands

Let’s learn

1. Creating Files:

touch filename: Creates an empty file named "filename“

```
touch my_new_file.txt
```

2. Reading Files:

cat filename: Displays the entire content of "filename" in the terminal

```
cat my_file.txt
```

‘less filename’: Opens "filename" in a pager, allowing you to scroll through the content. Press ‘q’ to exit.

```
less my_file.txt
```

Let’s learn

3. Deleting Files:

‘rm filename’: Deletes the file "filename". Use with caution as this is permanent.

```
rm my_file.txt
```

4. Copying Files:

‘cp source destination’: Copies the content of "source" file to "destination".

```
cp my_file.txt  my_file_copy.txt
cp my_file.txt /path/to/new/directory/
```

Let’s learn

5. Moving/Renaming Files:

‘mv oldname newname’: Renames the file from "oldname" to "newname".

```
mv my_file.txt  new_file_name.txt
```

‘mv source /path/to/destination/’: Moves the "source" file to the specified "destination" directory

```
mv my_file.txt /path/to/existing/directory/
```

String manipulation using tools like awk, sed, and parameter expansion

String Manipulation

1. Parameter Expansion (Bash Built-in):

- Substring Extraction:

```
string='Hello, World!'
echo ${string:7:5}   # Output: World
```

- Pattern Replacement:

```
string='The quick brown fox jumps over the lazy dog.'
replaced_string=${string//quick/slow}
echo $replaced_string # Output: The slow brown fox jumps over the
lazy dog.
```

String Manipulation

2. awk:

awk is a powerful text processing tool that can be used to manipulate strings1. The gsub() function can be used for substitution.

```
 string='Hello, World!'
 echo $string | awk '{gsub(/World/, "Universe"); print}'  # Output:
Hello, Universe!
```

3. sed (Stream Editor):

sed is another tool for string manipulation, often used for substitution.

```
string='Hello, World!'
echo $string | sed 's/World/Universe/g'  # Output: Hello, Universe!
```

String Manipulation

You can modify files in place with the ‘-i’ flag.

```
sed -i 's/universe/Universe/g' quotes.txt
```

Example Custom Functions (using ‘sed’):

```
substitute() {
  echo "$1" | sed "s/$2/$3/g"
}

input="Hello World"
search="World"
replace="Bash"
result=$(substitute "$input" "$search" "$replace")
echo "$result" # Output: Hello Bash
```

Demonstrating file compression using gzip, bzip2, and tar for archiving

Let’s do it

Here's how to use gzip, bzip2, and tar for file compression and archiving:

1. Tar for Archiving:

- To create a tar file, use the following command:

```
tar -cvf archive.tar file1 file2 directory1
```

- To view the contents of the archive:

```
tar -tvf filename.tar
```

- To extract the archive:

```
tar -xvf archive.tar
```

Let’s do it

2. Compressing Archives with Gzip:

- ‘tar’ can use ‘gzip’ to compress archived files. The ‘-z’ switch makes the ‘tar’ command use ‘gzip’.

- To create a Gzip compressed archive file.

```
tar -czvf archive.tar.gz file1 file2 directory1
```

- To extract the contents of a Gzip compressed file.

```
tar -xzvf archive.tar.gz
```

Let’s do it

3. Compressing Archives with Bzip2:

- ‘tar’ supports other compression systems like ‘bzip2’, which may offer better compression rates.

- To create a ‘Bzip2’ compressed file:

```
tar -cjvf archive.tar.bz2 file1 file2 directory1
```

- To extract a Bzip2 compressed file:

```
tar -xjvf archive.tar.bz2
```

Quiz 2

Q. What does the -z switch do when using the tar command?

A

Uses gzip to compress archived files

B

Creates a Bzip2 compressed archive file

Quiz 2

Q. What does the -z switch do when using the tar command?

A

Uses gzip to compress archived files

B

Creates a Bzip2 compressed archive file

PW SKILLS

Explaining command substitution using backticks (‘command’) and $()

Command substitution

Command substitution in Bash allows you to execute a command and insert its output into another command or context.

Backticks (‘command’):

- Encloses the command to be executed.
- The output of the command replaces the backtick expression.
- Backticks are an older syntax and can be harder to read, especially with nested commands, because they require escaping inner backticks.

$() Syntax:

- Encloses the command to be executed.
- Considered more readable and less prone to errors, especially when nesting commands.
- The $( ) syntax is generally preferred over backticks in modern Bash scripting.

Command substitution

For example, to capture the output of ‘ls | wc –l’ (which lists the number of files in the current directory) and embed it in an echo command, you can use either syntax.

```
echo "There are `ls | wc -l` files in this directory"
```

Or

```
echo "There are $(ls | wc -l) files in this directory"
```

SKILLS

Demonstrating how to use command substitution

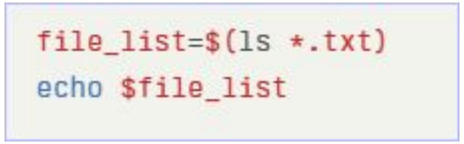
Let’s do it

You can perform command substitution using two primary methods:

1. ‘$()’ Syntax: The recommended and more readable syntax.
2. Backticks (‘command’): An older syntax that is still functional but can be harder to read, especially when nesting commands.

Here are several examples demonstrating command substitution:

- Assigning command output to a variable:



```
file_list=$(ls *.txt)
echo $file_list
```

Let’s do it

- Using command output as an argument:

```
rm $(cat filename)
```

- Embedding command output in a string:

```
echo "Today is $(date +%m-%d-%Y)"
```

Let’s do it

- Setting a variable to the contents of a file:

```
variable1=$(<file1)
# or
variable2=$(cat file2)
```

- These commands set ‘variable1’ and ‘variable2’ to the contents of ‘file1’ and ‘file2’ respectively.

Introducing process substitution with <() and >().

Process substitution

Process substitution in Bash allows you to treat the output (or input) of a command as if it were a file. This is particularly useful for commands that expect file arguments but you want to use the output of another command directly.

Syntax:

- <(command): Treats the output of command as a file for reading (input).
- >(command): Treats the input as a file to be written to by command (output).

Process substitution

Example:

```
diff <(ls *.c | cut -d. -f1) <(ls *.out | cut -d. -f1)
```

This command compares the output of two commands (‘ls *.c | cut -d. -f1’ and ‘ls *.out | cut -d. -f1’) as if they were files, without needing to create temporary files.

PW SKILLS

Providing Examples

Examples

Examples:

1. Compare two files using ‘diff’ with process substitution:

```
diff <(sort file1) <(sort file2)
```

2. Capturing the output of ‘ls’ and passing it as input to another command:

```
wc -l < <(ls input.txt)
```

3. Comparing directory sizes:

```
diff -r <(du -sh dir1) <(du -sh dir2)
```

Time for Case Study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this session
- Review your lecture notes
- Note down questions and queries regarding this session and consult the teaching assistant

Thanks
!