DAY 4x/14 LEARNING ABOUT LINUX

AUTOMATING TASKS WITH CRON AND CRONTAB

Cron, a time-based job scheduler in Unix-like computer operating systems, plays a crucial role in automating the execution of repetitive tasks. This functionality is not just a convenience but a necessity for efficient system management, ensuring tasks like backups, updates, and custom scripts are executed at predefined intervals without manual intervention.

I. MASTERING CRON JOBS IN LINUX

Cron jobs, scheduled tasks in the Linux environment, are orchestrated by the cron daemon. These jobs can be programmed to run at intervals of minutes, hours, days, months, or even specified days of the week.

PRACTICAL APPLICATIONS OF CRON JOBS:

- **System Maintenance**: Automating routine tasks, from database backups to system updates and cache clearing.
- **Scheduled Reporting**: Generating and sending reports or data snapshots at regular intervals.
- **Automated Alerts**: Checking system health or application status and triggering alerts or actions.

DISCOVERING SCHEDULED TASKS:

User-specific cron jobs are organized within crontab files, named after the user and located variably across different Linux distributions. For instance, Red Hat distributions like CentOS store these files in /var/spool/cron, while Debian and Ubuntu prefer /var/spool/cron/crontabs.

- **Viewing User Cron Jobs**: Utilize crontab -l to list the current user's scheduled tasks.
- **Editing User Cron Jobs**: Editing or creating cron jobs for different users can be achieved with sudo crontab -u username -e.
- **Locating User Cron Jobs**: To uncover which users have scheduled cron jobs, a peek into the spool directory with sudo ls /var/spool/cron/crontabs is insightful.

UNDERSTANDING SYSTEM-WIDE CRON JOBS:

Beyond individual user tasks, Linux supports system-wide cron jobs, editable only by system administrators and located in /etc/crontab and /etc/cron.d. These directories, along with /etc/cron.{hourly,daily,weekly,monthly}, facilitate the scheduling of tasks to run across the entire system at regular intervals.

Integration with Systemd Timers:

In modern Linux distributions using systemd, systemd timers offer an alternative to cron, providing more flexible and powerful means of scheduling tasks with dependencies on other system events or conditions.

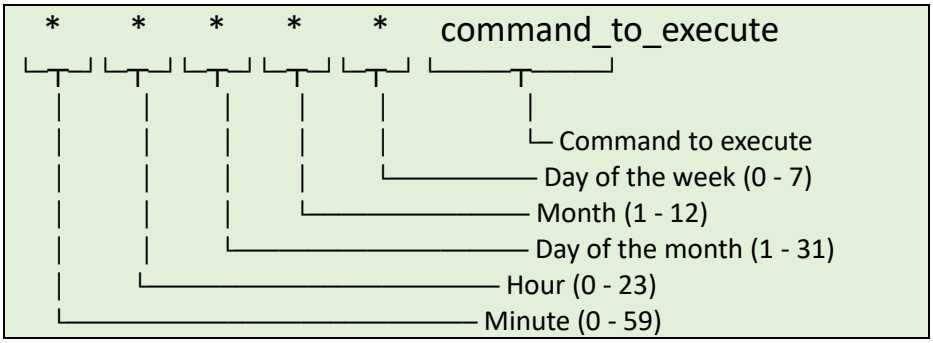
II. SCHEDULING WITH CRONTAB

Crontab, short for "cron table," is a configuration file that specifies schedules for cron jobs. There are two flavors of crontab files:

- **User Crontabs**: Unique to each user, allowing the scheduling of personal tasks.
- **System-wide Crontabs**: Managed by system administrators for tasks affecting the whole system.

CRAFTING A CRONTAB ENTRY:

A crontab entry consists of six fields: five time and date fields, followed by the command to execute, delineated by spaces.



OPERATORS FOR FLEXIBILITY:

- *** (Asterisk):** Represents "every" possible value for a field.
- **- (Hyphen):** Specifies ranges.
- **, (Comma):** Lists multiple discrete values.
- **/ (Slash):** Indicates step values for incrementing within a specified range.

SPECIAL TIME SCHEDULES WITH MACROS:

Crontab supports shorthand macros like @yearly, @monthly, @weekly, @daily, @hourly, and @reboot, simplifying the scheduling of common interval tasks.

EDITING AND MANAGING CRONTAB FILES:

The crontab command facilitates the creation, viewing, and editing of crontab files with options like -e for editing, -l for listing, and -r for removing a user's crontab.

CRONTAB ENVIRONMENT VARIABLES:

It's possible to set environment variables within a crontab file, such as PATH, SHELL, and MAILTO, to customize the execution environment for cron jobs.

ACCESS RESTRICTIONS:

Control over who can schedule cron jobs is managed through /etc/cron.allow and /etc/cron.deny files, determining user access to crontab command functionality.

III. ADVANCED SCHEDULING: CRON JOBS EVERY 5, 10, OR 15 MINUTES

For tasks requiring frequent execution, cron facilitates scheduling at intervals of 5, 10, or 15 minutes using the slash operator for simplicity, e.g., */5, */10, */15 in the minute field of a crontab entry.

PRACTICAL EXAMPLES:

1. **Backup Every 10 Minutes**: */10 * * * * /path/to/backup_script.sh
2. **Update Checks Every 15 Minutes**: */15 * * * * /usr/bin/check_updates
3. **Log Monitoring Every 5 Minutes**: */5 * * * * /path/to/log_monitor.sh

CONCLUSION

Cron and crontab offer a robust framework for automating routine tasks, essential for system health, security, and efficiency. Mastery of cron scheduling not only optimizes system performance but also ensures critical tasks are performed reliably, making it an indispensable tool in the Linux administrator's toolkit.

HOMEWORK 4.x

HOMEWORK SET 1/4: BASIC UNDERSTANDING AND CREATION

1. **Create a Simple Cron Job**: Write a Bash script that appends the current date and time to a file named date_log.txt in the user's home directory every day at noon.

   Solution:

   ```
   echo "0 12 * * * echo `date` >> ~/date_log.txt" | crontab -
   ```

2. **Backup Cron Job:** Schedule a cron job to create a compressed archive of the /etc directory weekly on Sunday at 3 AM and save it to /backup/etc_backup.tar.gz.

   Solution:

   ```
   echo "0 3 * * 0 tar -czf /backup/etc_backup.tar.gz /etc" | crontab -
   ```

HOMEWORK SET 2/4: INTERMEDIATE SCRIPTING WITH CRON

3. **Disk Usage Alert**: Write a cron job that checks the disk usage of the / partition every day at 8 PM. If the usage exceeds 80%, it should write a warning message to /var/log/disk_usage.log.

   Solution:

```
echo "0 20 * * * df / | awk '0+$5 >= 80 {print \"Disk usage warning: \", $5}' >> /var/log/disk_usage.log" | crontab -
```

4. **Monitor HTTP Service:** Create a cron job that pings localhost on port 80 every hour. If the ping fails, it should attempt to restart the httpd service and log this action to /var/log/httpd_monitor.log.

   Solution:

```
echo "0 * * * * curl -s http://localhost > /dev/null || (systemctl restart httpd && echo \"httpd restarted at `date`\" >> /var/log/httpd_monitor.log)" | crontab -
```

HOMEWORK SET 3/4: ADVANCED CRON SCHEDULING

5. **Database Backup Rotation**: Schedule a cron job to backup the mydb database at 2 AM daily and keep only the last 7 backups by removing older files in the /backup/db directory.

   Solution:

```
echo "0 2 * * * mysqldump -u root mydb > /backup/db/mydb_$(date +\%Y\%m\%d).sql && find /backup/db/ -type f -mtime +7 -delete" | crontab -
```

6. **System Update and Reboot:** Write a cron job that runs system updates using apt-get every Saturday at midnight. If updates are successfully applied, it should reboot the system.

   Solution:

```
echo "0 0 * * 6 apt-get update && apt-get upgrade -y && reboot" | crontab -
```

HOMEWORK SET 4/4:INTEGRATION WITH SYSTEMD TIMERS

7. **Convert to Systemd Timer**: Given a cron job that echoes "Hello World" to /tmp/hello.txt every minute, convert this job into a systemd timer and service.

   Solution:

   1. **Create the service file (hello.service)**:

```
[Unit]
Description=Hello World Service

[Service]
Type=oneshot
ExecStart=/bin/bash -c 'echo "Hello World" > /tmp/hello.txt'
```

   2. **Create the timer file (hello.timer):**

```
[Unit]
Description=Runs hello every minute

[Timer]
OnUnitActiveSec=1min
Unit=hello.service
```

```
[Install]
WantedBy=timers.target
```

3. **Enable and start the timer:**

```
systemctl enable hello.timer && systemctl start hello.timer
```

These homework tasks are designed to reinforce practical cron scheduling concepts, script creation, and system administration tasks that can be resolved directly within the Bash shell environment.

Submission Guidelines

- *Ensure your scripts are well-commented to explain your logic and approach.*
- *Test your scripts with sample data to verify correctness.*
- *Submit (in the forum) your AWK scripts along with a brief report describing your solution strategy for each task. Also export history and upload it as usual.*