Scheduling Tasks with Cron

1. Introduction to Cron

Cron is a time-based job scheduler in Unix-like operating systems. It enables users to schedule scripts or commands to run automatically at specified times, dates, or intervals.

What is Cron?

- **Cron Daemon (crond)**: Background service that executes scheduled tasks
- **Crontab**: Table file containing the list of commands to be executed
- **Cron Job**: A scheduled task/command

Common Use Cases

- Daily backups
- System maintenance tasks
- Automated reports
- Database cleanups
- Log rotation
- Website monitoring
- Email reminders
- Data synchronization

Checking if Cron is Running

```
# Check cron service status
sudo systemctl status cron

# Or on older systems
sudo service cron status

# Start cron if not running
sudo systemctl start cron
```

---

2. Understanding Cron Syntax

Basic Cron Job Format

```
* * * * * command_to_execute
| | | | |
| | | | └── Day of Week (0-7, where 0 and 7 are Sunday)
| | | └──── Month (1-12)
| | └────── Day of Month (1-31)
| └──────── Hour (0-23)
└────────── Minute (0-59)
```

Field Values

- **Minute**: 0-59
- **Hour**: 0-23 (0 = midnight, 23 = 11 PM)
- **Day of Month**: 1-31
- **Month**: 1-12 (or JAN-DEC)
- **Day of Week**: 0-7 (0 and 7 = Sunday, 1 = Monday, or MON-SUN)

Special Characters

- `*` (asterisk): Any value / every
- `,` (comma): List of values (e.g., 1,3,5)
- `-` (dash): Range of values (e.g., 1-5)
- `/` (slash): Step values (e.g., */5 means every 5)
- `?` (question mark): No specific value (not always supported)

Special Strings (Shortcuts)

```
@reboot        # Run once at startup
@yearly        # Run once a year (0 0 1 1 *)
@annually      # Same as @yearly
@monthly       # Run once a month (0 0 1 * *)
@weekly        # Run once a week (0 0 * * 0)
@daily         # Run once a day (0 0 * * *)
@midnight      # Same as @daily
@hourly        # Run once an hour (0 * * * *)
```

---

3. Managing Cron Jobs

Viewing Crontab

```
# View your crontab
crontab -l

# View another user's crontab (requires sudo)
sudo crontab -u username -l

# View system-wide crontab
cat /etc/crontab
```

Editing Crontab

```
# Edit your crontab
crontab -e

# Edit another user's crontab (requires sudo)
sudo crontab -u username -e

# First time, you'll be asked to choose an editor
# Select nano (easiest for beginners) or vim
```

Creating Your First Cron Job

```
# Step 1: Open crontab editor
crontab -e

# Step 2: Add this line to run a script every day at 2 AM
0 2 * * * /path/to/your/script.sh

# Step 3: Save and exit
# For nano: Ctrl+O, Enter, Ctrl+X
# For vim: Press Esc, type :wq, press Enter
```

Removing Cron Jobs

```
# Remove all your cron jobs
crontab -r

# Remove a specific job: edit crontab and delete the line
crontab -e
```

Listing All Users' Cron Jobs

```
# List all crontab files
sudo ls -la /var/spool/cron/crontabs/

# View all users' cron jobs
for user in $(cut -f1 -d: /etc/passwd); do
    echo "Crontab for $user:"
    sudo crontab -u $user -l 2>/dev/null
done
```

---

4. Cron Expressions Explained

Simple Examples with Explanation

Every Minute

```
* * * * * /path/to/script.sh
# Runs every minute of every hour of every day
```

Every 5 Minutes

```
*/5 * * * * /path/to/script.sh
# Runs at minute 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55
```

Every Hour

```
0 * * * * /path/to/script.sh
# Runs at the start of every hour (XX:00)
```

Every Day at Midnight

```
0 0 * * * /path/to/script.sh
# Runs at 00:00 (midnight) every day
```

Every Day at 3:30 PM

```
30 15 * * * /path/to/script.sh
# Runs at 15:30 (3:30 PM) every day
```

Every Monday at 9:00 AM

```
0 9 * * 1 /path/to/script.sh
# Runs at 09:00 every Monday
```

First Day of Every Month

```
0 0 1 * * /path/to/script.sh
# Runs at midnight on the 1st of each month
```

Every Weekday at 8:00 AM

```
0 8 * * 1-5 /path/to/script.sh
# Runs Monday through Friday at 08:00
```

Every 6 Hours

```
0 */6 * * * /path/to/script.sh
# Runs at 00:00, 06:00, 12:00, 18:00
```

Twice Daily

```
0 9,17 * * * /path/to/script.sh
# Runs at 09:00 and 17:00 every day
```

---

5. Common Scheduling Patterns

Time-Based Patterns

Multiple Specific Times

```
# At 6 AM, 12 PM, and 6 PM every day
0 6,12,18 * * * /path/to/script.sh

# Every 15 minutes during business hours (9 AM - 5 PM)
*/15 9-17 * * * /path/to/script.sh

# Every hour from 9 AM to 5 PM on weekdays
0 9-17 * * 1-5 /path/to/script.sh
```

Day-Based Patterns

```
# Every Sunday at 2 AM
0 2 * * 0 /path/to/script.sh

# Every weekend day at noon
0 12 * * 0,6 /path/to/script.sh

# Last day of every month at 11:59 PM (approximation)
59 23 28-31 * * [ "$(date +\%d -d tomorrow)" = "01" ] && /path/to/script.sh

# Every 15th and 30th of the month
0 0 15,30 * * /path/to/script.sh
```

Week-Based Patterns

```
# First Monday of every month at 10 AM
0 10 1-7 * 1 /path/to/script.sh

# Every other week on Monday
0 9 * * 1 [ $(expr $(date +\%U) \% 2) -eq 0 ] && /path/to/script.sh
```

Month-Based Patterns

```
# First day of January, April, July, October
0 0 1 1,4,7,10 * /path/to/script.sh

# Every quarter (Jan 1, Apr 1, Jul 1, Oct 1)
0 0 1 */3 * /path/to/script.sh

# Summer months only (June, July, August)
0 0 * 6-8 * /path/to/script.sh
```

Year-Based Patterns

```
# New Year's Day at midnight
0 0 1 1 * /path/to/script.sh

# Christmas Day at 9 AM
0 9 25 12 * /path/to/script.sh
```

---

7. Practical Examples

Example 1: Daily Backup Script

```
# Crontab entry: Backup every day at 2 AM
0 2 * * * /home/user/scripts/daily_backup.sh >> /var/log/backup.log 2>&1

# Script: daily_backup.sh
#!/bin/bash

BACKUP_DIR="/backup"
SOURCE_DIR="/home/user/important_files"
DATE=$(date +%Y%m%d_%H%M%S)
LOGFILE="/var/log/backup.log"

log() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOGFILE"
}

log "Starting backup"

# Create backup directory if it doesn't exist
mkdir -p "$BACKUP_DIR"

# Create compressed backup
tar -czf "$BACKUP_DIR/backup_$DATE.tar.gz" "$SOURCE_DIR"

if [ $? -eq 0 ]; then
    log "Backup completed successfully: backup_$DATE.tar.gz"

    # Remove backups older than 7 days
    find "$BACKUP_DIR" -name "backup_*.tar.gz" -mtime +7 -delete
    log "Old backups cleaned up"
else
    log "ERROR: Backup failed!"
    exit 1
fi
```

Example 2: Website Monitoring

```
# Crontab: Check website every 5 minutes
*/5 * * * * /home/user/scripts/monitor_website.sh

# Script: monitor_website.sh
#!/bin/bash

WEBSITE="https://example.com"
LOGFILE="/var/log/website_monitor.log"
EMAIL="admin@example.com"

response=$(curl -s -o /dev/null -w "%{http_code}" "$WEBSITE")

if [ "$response" != "200" ]; then
    echo "$(date): Website down! Status: $response" >> "$LOGFILE"
    echo "Website $WEBSITE is down. HTTP Status: $response" | mail -s "Website
Alert" "$EMAIL"
else
    echo "$(date): Website OK" >> "$LOGFILE"
fi
```

Example 3: Database Backup

```
# Crontab: Backup database every day at 3 AM
0 3 * * * /home/user/scripts/db_backup.sh

# Script: db_backup.sh
#!/bin/bash

DB_NAME="myapp_db"
DB_USER="db_admin"
DB_PASS="secure_password"
BACKUP_DIR="/backup/database"
DATE=$(date +%Y%m%d_%H%M%S)
RETENTION_DAYS=14

# Create backup directory
mkdir -p "$BACKUP_DIR"

# Dump database
mysqldump -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" | gzip >
"$BACKUP_DIR/${DB_NAME}_$DATE.sql.gz"

if [ $? -eq 0 ]; then
    echo "$(date): Database backup successful" >> /var/log/db_backup.log

    # Remove old backups
    find "$BACKUP_DIR" -name "${DB_NAME}_*.sql.gz" -mtime +$RETENTION_DAYS -
delete
else
    echo "$(date): Database backup FAILED" >> /var/log/db_backup.log
    exit 1
fi
```

Example 4: Disk Space Alert

```
# Crontab: Check disk space every hour
0 * * * * /home/user/scripts/disk_alert.sh

# Script: disk_alert.sh
#!/bin/bash

THRESHOLD=80
EMAIL="admin@example.com"

df -H | grep -vE '^Filesystem|tmpfs|cdrom' | awk '{ print $5 " " $1 }' | while
read output;
do
    usage=$(echo $output | awk '{ print $1}' | sed 's/%//g')
    partition=$(echo $output | awk '{ print $2 }')

    if [ $usage -ge $THRESHOLD ]; then
        echo "Running out of space on $partition ($usage%)" | \
        mail -s "Disk Space Alert: $partition at ${usage}%" "$EMAIL"
    fi
done
```

Example 5: Log Rotation

```
# Crontab: Rotate logs weekly on Sunday at midnight
0 0 * * 0 /home/user/scripts/rotate_logs.sh

# Script: rotate_logs.sh
#!/bin/bash

LOG_DIR="/var/log/myapp"
ARCHIVE_DIR="/var/log/myapp/archive"
DATE=$(date +%Y%m%d)

mkdir -p "$ARCHIVE_DIR"

# Find logs older than 1 day and compress them
find "$LOG_DIR" -maxdepth 1 -name "*.log" -mtime +1 -exec gzip {} \;

# Move compressed logs to archive
find "$LOG_DIR" -maxdepth 1 -name "*.log.gz" -exec mv {} "$ARCHIVE_DIR/" \;

# Delete archives older than 90 days
find "$ARCHIVE_DIR" -name "*.log.gz" -mtime +90 -delete

echo "$(date): Log rotation completed" >> "$LOG_DIR/rotation.log"
```

Example 6: System Maintenance

```
# Crontab: System cleanup every Sunday at 1 AM
0 1 * * 0 /home/user/scripts/system_cleanup.sh

# Script: system_cleanup.sh
#!/bin/bash

LOG="/var/log/system_cleanup.log"

log() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" >> "$LOG"
}

log "Starting system cleanup"

# Update package lists
apt-get update >> "$LOG" 2>&1
log "Package lists updated"

# Clean package cache
apt-get clean >> "$LOG" 2>&1
apt-get autoclean >> "$LOG" 2>&1
log "Package cache cleaned"

# Remove old kernels (keep current and one previous)
apt-get autoremove --purge -y >> "$LOG" 2>&1
log "Old packages removed"

# Clear thumbnail cache
rm -rf /home/*/.cache/thumbnails/*/* >> "$LOG" 2>&1
log "Thumbnail cache cleared"

# Clear temporary files older than 7 days
find /tmp -type f -atime +7 -delete >> "$LOG" 2>&1
log "Old temporary files removed"

log "System cleanup completed"
```

Example 7: Automated Report Generation

```
# Crontab: Generate weekly report every Monday at 8 AM
0 8 * * 1 /home/user/scripts/weekly_report.sh

# Script: weekly_report.sh
#!/bin/bash

REPORT_DIR="/reports"
DATE=$(date +%Y%m%d)
REPORT_FILE="$REPORT_DIR/weekly_report_$DATE.txt"
EMAIL="manager@example.com"

mkdir -p "$REPORT_DIR"

{
    echo "Weekly System Report - $(date '+%Y-%m-%d')"
    echo "==================================="
    echo ""
    echo "System Uptime:"
    uptime
    echo ""
    echo "Disk Usage:"
    df -h
    echo ""
    echo "Memory Usage:"
    free -h
    echo ""
    echo "Top 10 Processes by CPU:"
    ps aux --sort=-%cpu | head -11
    echo ""
    echo "Top 10 Processes by Memory:"
    ps aux --sort=-%mem | head -11
    echo ""
    echo "Failed Login Attempts:"
    grep "Failed password" /var/log/auth.log | tail -20
} > "$REPORT_FILE"

# Email the report
cat "$REPORT_FILE" | mail -s "Weekly System Report - $(date '+%Y-%m-%d')"
"$EMAIL"

echo "$(date): Report generated and sent" >> /var/log/reports.log
```

---

9. Advanced Techniques

Conditional Execution

```
# Run only if a file exists
0 2 * * * [ -f /tmp/trigger.txt ] && /path/to/script.sh

# Run only on specific date
0 0 25 12 * /path/to/christmas_script.sh

# Run only if previous command succeeds
0 2 * * * /path/to/backup.sh && /path/to/cleanup.sh

# Run even if previous command fails
0 2 * * * /path/to/backup.sh || /path/to/fallback.sh
```

Parallel Execution

```
# Run multiple tasks in parallel
0 2 * * * /path/to/task1.sh & /path/to/task2.sh & /path/to/task3.sh &
```

Essential Commands

```
crontab -e      # Edit crontab
crontab -l      # List cron jobs
crontab -r      # Remove all cron jobs
crontab -v      # Display last edit time

sudo systemctl status cron  # Check cron status
sudo tail -f /var/log/cron  # View cron log
```

Conclusion

Cron is a powerful tool for automating tasks on Linux systems. With proper understanding of its syntax and best practices, you can create reliable automated workflows that save time and reduce manual intervention.

Key Takeaways:

- Always use absolute paths in cron jobs

- Log your output for debugging
- Test scripts manually before scheduling
- Handle errors and edge cases
- Monitor your cron jobs regularly
- Keep your crontab documented and backed up

Next Steps:

- Create your first automated backup script
- Set up monitoring for critical services
- Implement log rotation for your applications
- Experiment with different scheduling patterns

Happy automating!