How to Handle Daylight Saving Time Changes on Ubuntu

Manage daylight saving time transitions on Ubuntu servers, understand their impact on logs and scheduled tasks, and configure systems to minimize DST-related disruptions.

@nawazdhandala • Mar 02, 2026 • 8 min read

Ubuntu | Timezone | DST | System Administration | Time Configuration

Daylight saving time (DST) transitions cause real problems in production systems. Twice a year, clocks either skip an hour forward or fall back an hour, and anything that relies on wall clock time can behave unexpectedly. Log timestamps become ambiguous, cron jobs may run twice or not at all, and applications that schedule by local time need to account for the change. This guide covers everything you need to know about DST on Ubuntu servers.

Why DST Causes Problems for Servers

The core issue is that local time is not monotonically increasing. When clocks fall back:

- The local time 2:30 AM occurs twice in the same day
- Log entries at 2:30 AM are ambiguous - which one?
- Cron jobs scheduled at 2:30 AM may run twice

When clocks spring forward:

- The local time 2:30 AM never occurs
- Cron jobs scheduled during the skipped hour don't run

Additional complications:

- Time comparisons in applications that use local time give wrong results
- Database entries with local timestamps are ambiguous
- Monitoring alerts may fire incorrectly around transition times

The Simple Solution: Use UTC on Servers

The most reliable approach is to configure servers to use UTC:

```bash
# Set timezone to UTC
sudo timedatectl set-timezone UTC

# Verify
timedatectl | grep "Time zone"
# Should show: Time zone: UTC (UTC, +0000)
```

UTC has no DST. It never skips hours or falls back. Every moment in time maps to exactly one UTC timestamp. All the DST-related problems listed above disappear.

For servers, there's rarely a good reason to use a local timezone. If your application or users need local time display, convert from UTC in the application layer or at the reporting level.

Checking Your Current DST Configuration

```bash
# Check current timezone and DST status
timedatectl

# Check if current timezone observes DST
# tzselect shows timezone details including DST rules
# Or check zdump output
zdump -v /etc/localtime | grep -E "2026|2027" | head -10

# Shows transition times and resulting offset
# Example output for America/New_York:
# /etc/localtime  Sun Mar  8 06:59:59 2026 UTC = Sun Mar  8 01:59:59 2026 EST isdst=0 gmtoff=-18000
# /etc/localtime  Sun Mar  8 07:00:00 2026 UTC = Sun Mar  8 03:00:00 2026 EDT isdst=1 gmtoff=-14400

# Check DST transition dates for any timezone
zdump -v America/New_York | grep "2026" | head -5
zdump -v Europe/London | grep "2026" | head -5
```

Finding Out When DST Transitions Will Occur

```bash
# Check transitions for the current timezone
zdump -v /etc/localtime | grep "$(date +%Y)" | head -10

# Check transitions for a specific timezone
zdump -v America/New_York | grep "2026"

# Python script for clean DST transition display
python3 << 'EOF'
import pytz
from datetime import datetime

def show_dst_transitions(tz_name, year):
    tz = pytz.timezone(tz_name)
    print(f"DST transitions for {tz_name} in {year}:")

    prev_offset = None
    # Check each month
    for month in range(1, 13):
        for day in range(1, 32):
            try:
                dt = datetime(year, month, day, 0, 0, 0)
                dt_aware = tz.localize(dt)
                offset = dt_aware.utcoffset()
                if prev_offset is not None and offset != prev_offset:
                    print(f"  {dt.strftime('%b %d')}: offset changes from {prev_offset} to {offset}")
                prev_offset = offset
            except ValueError:
                pass

show_dst_transitions("America/New_York", 2026)
show_dst_transitions("Europe/London", 2026)
EOF
```

Cron Jobs and DST

Standard cron runs jobs based on local time. This creates DST issues:

The Fallback Problem (2 AM becomes 1 AM)

A cron job at `30 2 * * *` (2:30 AM) will run twice when clocks fall back because 2:30 AM happens twice.

The Spring Forward Problem (2 AM becomes 3 AM)

A cron job at `30 2 * * *` will not run when clocks spring forward because 2:30 AM never exists.

Solutions for Cron DST Issues

Option 1: Schedule in UTC with the TZ variable:

```bash
sudo crontab -e
```

```bash
# Set crontab to use UTC (ignores DST)
CRON_TZ=UTC

# This job runs at 2:30 AM UTC every day, regardless of DST
30 2 * * * /usr/local/bin/daily-backup.sh

# This job runs at 9 AM UTC (which is 4 AM EST or 5 AM EDT in New York)
0 9 * * * /usr/local/bin/morning-report.sh
```

Option 2: Use systemd timers instead of cron:

systemd timers can be configured to use UTC or to skip DST transitions gracefully:

```bash
sudo nano /etc/systemd/system/daily-backup.timer
```

```bash
[Unit]
Description=Daily Backup Timer

[Timer]
# Use UTC time specification (OnCalendar uses system timezone by default)
OnCalendar=*-*-* 02:30:00 UTC

# Accuracy window - how precisely to hit the scheduled time
AccuracySec=1min

[Install]
WantedBy=timers.target
```

```bash
sudo systemctl enable daily-backup.timer
sudo systemctl start daily-backup.timer

# Verify when it will next fire
systemctl list-timers daily-backup.timer
```

Option 3: Avoid scheduling during transition hours:

If you must use local time cron, avoid scheduling jobs between 1 AM and 3 AM in timezones that observe DST. These are the hours most likely to be affected by transitions.

Log Timestamps and DST

Changing log timestamps is one of the most visible DST impacts:

```bash
# systemd journal stores timestamps in UTC internally
# It always displays in the current system timezone
journalctl --since "2026-03-08 01:00:00" --until "2026-03-08 04:00:00"

# To see journal logs in UTC (removes DST ambiguity)
journalctl --utc --since "2026-03-08"

# syslog format (used by rsyslog) uses local time
# Log entries at 2:30 AM during fallback will appear twice
# Compare:
grep "Nov 1" /var/log/syslog | grep "01:3[0-9]"
# You may see duplicate timestamps

# For precise log correlation, always include timezone offset or use UTC
```

Application-Level DST Handling

Applications that store local timestamps in databases need special attention around DST transitions.

PostgreSQL

```bash
# PostgreSQL timestamp with timezone (recommended)
# SELECT NOW();  -- Returns with timezone info
# Store timestamps as TIMESTAMPTZ (timestamp with time zone)

# Check PostgreSQL timezone
sudo -u postgres psql -c "SHOW timezone;"

# Set PostgreSQL to UTC (recommended for servers)
sudo nano /etc/postgresql/14/main/postgresql.conf
# timezone = 'UTC'

sudo systemctl restart postgresql
```

MySQL

```bash
# Check MySQL timezone
mysql -u root -p -e "SELECT @@global.time_zone, @@session.time_zone;"

# Set MySQL to UTC
sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
# [mysqld]
# default_time_zone = '+00:00'

sudo systemctl restart mysql
```

Monitoring Around DST Transitions

Set up monitoring to watch for DST-related anomalies:

```bash
# Create a script to check for time-related issues
cat << 'EOF' | sudo tee /usr/local/bin/check-dst-readiness
#!/bin/bash
# Check system readiness for DST transitions

echo "=== DST Readiness Check ==="
echo ""

echo "Current timezone:"
timedatectl | grep "Time zone"
echo ""

echo "NTP synchronization:"
timedatectl | grep -E "synchronized|NTP"
echo ""

echo "Next DST transition (if any):"
zdump -v /etc/localtime 2>/dev/null | grep "$(date +%Y)" | \
    awk 'BEGIN{found=0}
  /isdst=1/ && !found {print "Spring forward to DST:", $0; found=1}' | head -2

zdump -v /etc/localtime 2>/dev/null | grep "$(date +%Y)" | \
    awk '/isdst=0/ && NR>1 {print "Fall back from DST:", $0; exit}' | head -2

echo ""
echo "Cron jobs scheduled during DST transition hours (1-3 AM):"
for user in $(cut -d: -f1 /etc/passwd); do
    crontab -l -u $user 2>/dev/null | grep -v "^#" | \
        awk '$2 ~ /^[12]$/ {print "User '$user': "$0}'
done
echo "(Empty = no cron jobs in transition hours)"
EOF

sudo chmod +x /usr/local/bin/check-dst-readiness
sudo /usr/local/bin/check-dst-readiness
```

Keeping Timezone Data Current

DST rules change. Countries occasionally modify their DST observation schedules. Keep timezone data current:

```bash
# Update timezone data package
sudo apt update
sudo apt install --only-upgrade tzdata

# Check when tzdata was last updated
dpkg -l tzdata | tail -1

# The tzdata package version includes the Olson database version
# e.g., tzdata 2024a-0ubuntu1 means Olson 2024a

# Verify timezone files are current
ls -la /usr/share/zoneinfo/ | head -5
zdump -v America/New_York | grep "2026"
```

DST Checklist for Production Servers

Run through this checklist before a DST transition:

```bash
#!/bin/bash
# DST pre-transition checklist

echo "=== DST Pre-Transition Checklist ==="
echo ""

# 1. Are we using UTC? (If yes, no DST concerns)
TZ=$(timedatectl show --property=Timezone --value)
if [ "$TZ" = "UTC" ]; then
    echo "[PASS] Server uses UTC - no DST impact"
else
    echo "[INFO] Server uses $TZ - DST transitions apply"
    echo ""

    # 2. Is NTP synchronized?
    NTP_SYNC=$(timedatectl show --property=NTPSynchronized --value)
    if [ "$NTP_SYNC" = "yes" ]; then
        echo "[PASS] NTP is synchronized"
    else
        echo "[FAIL] NTP is NOT synchronized - fix before DST transition"
    fi

    # 3. Check for cron jobs in transition hours
    echo ""
    echo "[CHECK] Cron jobs near transition hours:"
    crontab -l 2>/dev/null | grep -v "^#" | awk '$2 ~ /^[12]$/'
    sudo crontab -l 2>/dev/null | grep -v "^#" | awk '$2 ~ /^[12]$/'
fi

# 4. Check tzdata is current
TZDATA_VERSION=$(dpkg -l tzdata 2>/dev/null | tail -1 | awk '{print $3}')
echo ""
echo "[INFO] tzdata version: $TZDATA_VERSION"
```

What Happens at UTC Midnight on DST Transition Day

If your server is set to UTC, nothing special happens. The transition is transparent to the server. Applications and databases that store UTC timestamps are unaffected.

The only visible change on a UTC server during DST transitions is:

- The wall clock offset displayed in timestamps changes (e.g., EST -0500 changes to EDT -0400)
- Local-time representations of UTC timestamps shift

```bash
# On a UTC server, verify transition is handled correctly
date -d "2026-03-08 07:00:00 UTC"    # Should show as 2 AM EST (before transition)
date -d "2026-03-08 07:00:01 UTC"    # Should show as 3 AM EDT (after transition)
# This is purely display; internally still UTC
```

Configuring all servers to use UTC is the practical answer to DST complexity. It removes the issue entirely from the system level, letting you handle timezone display in application code where you have full control over the logic. For the servers that genuinely need to operate in a local timezone, audit your cron jobs and scheduled tasks before every DST transition to prevent surprises.

Share this article

Nawaz Dhandala Author
@nawazdhandala • Mar 02, 2026 • 8 min read
Nawaz is building OneUptime with a passion for engineering reliable systems and improving observability.
GitHub