# Archiving, Compression & Automation in Linux

Linux gives you powerful tools for managing a lot of files and for scheduling tasks so you don’t have to do them manually. Two major areas you’ll use a lot are:

* **Archiving & Compression:** combining files and making them smaller.
* **Automation:** scheduling commands so they run automatically at certain times.

Think of archiving as **packing many papers into a folder**, compression as **pressing that folder to take up less space**, and automation as **setting a reminder so the packing happens on its own every night**.

### 1. Key Concepts

* **Archiving:** Archiving is simply putting multiple files or folders into a single file. It doesn’t shrink them, it just groups them together. This makes it easier to copy, send or back up.
* *Example:* You have 500 log files in **/var/log**. You put them all into one **logs.tar** file for backup with:
* tar -cvf /home/user/logs.tar /var/log
  + -c means “create a new archive.”
  + -v shows which files are being processed (verbose).
  + -f says “the next word is the file name of the archive.”
* This takes the entire /var/log directory and creates a single file /home/user/logs.tar containing everything inside.
* **Compression:** Compression actually reduces the size of a file using an algorithm. When you compress an archive, you save disk space and speed up transfers.
* *Example:* You compress logs.tar into logs.tar.gz with:
* tar -cvzf /home/user/logs.tar.gz /var/log
  + -z tells tar to use gzip compression.
  + The archive now ends with .tar.gz showing it’s compressed.
* **tar command:** Stands for **T**ape **A**rchive. This is the all-in-one Linux tool for creating, extracting, and sometimes compressing archives.

### 2. tar Command Syntax

Stands for **T**ape **Ar**chive. This is the all-in-one Linux tool for creating, extracting, and sometimes compressing archives

tar [options] [archive\_name] [files\_or\_directories]

Options you’ll use most:

* **c (Create)** → Make a new archive file.
* **v (Verbose)** → Show progress (which files are added/extracted).
* **f (File)** → Name of the archive file (always last in the options).
* **x –** extract (unpack) archive
* **z** – gzip compression
* **j** – bzip2 compression
* **J** – xz compression
* **t** – list contents

### 3. Creating Archives Without Compression

tar -cvf /home/user/backup.tar /etc

* **What happens:** A file named /home/user/backup.tar is created containing everything from /etc. Its size is roughly the same as /etc.
* **Why:** Easier to move, copy, or back up one file instead of thousands of small ones.

### 4. Compressing Archives

When creating archives, tar can also **compress** them in one step, instead of first creating a .tar file and then compressing it separately. Each compression method has a specific switch and results in a different file extension.

|  |  |  |  |
| --- | --- | --- | --- |
| Compression Type | Option | Extension | Notes |
| **Gzip** | -z | .gz | Fastest compression, widely used. Good when speed matters more than file size. |
| **Bzip2** | -j | .bz2 | Slower than gzip, but gives better compression. Balanced choice. |
| **Xz** | -J | .xz | Slowest but provides the **highest compression ratio**. Great for saving space. |

#### Comparison of Formats

|  |  |  |
| --- | --- | --- |
| Algorithm | Size After Compression | Speed |
| **Gzip (-z)** | Good (28 MB → 6.2 MB) | Fast |
| **Bzip2 (-j)** | Better (28 MB → 5.4 MB) | Slower |
| **Xz (-J)** | Best (28 MB → 4.4 MB) | Slowest |

**Full Commands:**

* **Gzip Compression (Fastest):**
* tar -cvzf /home/user/backup.tar.gz /etc
  + -c → create archive
  + -v → verbose (list files being archived)
  + -z → compress with gzip
  + -f → specify filename
* **Bzip2 Compression (Balanced):**
* tar -cvjf /home/user/backup.tar.bz2 /etc
  + -j → compress with bzip2
  + Produces smaller files than gzip, but takes more time.
* **Xz Compression (Maximum Space Saving):**
* tar -cvJf /home/user/backup.tar.xz /etc
  + -J → compress with xz
  + Excellent when storage is limited, though compression/decompression takes longer.

*Tip:* Use gzip if you need speed, xz if you want maximum compression, bzip2 if you want a balance.

### 5. Viewing and Extracting Archives

Once an archive is created (with or without compression), you’ll often need to either **inspect its contents** or **extract files** from it. The tar command makes both tasks easy.

#### Viewing Contents (Without Extraction)

To see what’s inside an archive without actually unpacking it:

tar -tvf /home/user/backup.tar

* -**t** → list contents
* **-v** → verbose (show details like permissions, owner, size, timestamp)
* **-f** → specify archive file

✅ This also works for compressed archives, as long as you use the correct option:

tar -tvzf /home/user/backup.tar.gz # gzip

tar -tvjf /home/user/backup.tar.bz2 # bzip2

tar -tvJf /home/user/backup.tar.xz # xz

This is useful when you only want to **check the files** before extracting.

#### Extract Archive (Current Folder)

To unpack all files into the **current working directory**:

tar -xf /home/user/backup.tar

* -x → extract files
* -f → specify archive file

For compressed files, add the respective option:

tar -xvzf /home/user/backup.tar.gz # gzip  
tar -xvjf /home/user/backup.tar.bz2 # bzip2  
tar -xvJf /home/user/backup.tar.xz # xz

💡 The -v flag is optional, but it helps to see the list of files being extracted.

#### Extract to a Different Location

You don’t always want to clutter your current folder with extracted files. Use the -C option to specify a directory:

mkdir /home/user/extracted\_files  
tar -xf /home/user/backup.tar -C /home/user/extracted\_files

* -C /path/to/folder → extract files into that folder
* If the folder doesn’t exist, create it with mkdir first

#### Extract Specific Files Only

You don’t have to extract everything. You can specify the filename(s) to extract:

tar -xf /home/user/backup.tar etc/passwd etc/hosts

This will only extract passwd and hosts from inside the etc folder of the archive.

💡 **Tips:**

* Use tar -tvf to preview before extracting.
* Use -C for controlled extraction into a safe folder.
* Always check permissions (extracted files retain original permissions).

### 6. Automating Jobs in Linux

In Linux, you can tell the system to **run tasks automatically** without manual intervention. This is very useful for:

* Running **nightly backups**
* Cleaning up **temporary files**
* Scheduling **shutdowns/restarts**
* Running scripts at specific times

Linux provides two main tools for automation:

* **at** → Run a job **once** at a scheduled time
* **cron** → Run a job **repeatedly** at defined intervals

#### 6.1 at Command – One-Time Jobs

The at command is used when you want a task to execute **only once** at a specific time.

**Examples:**

at 22:05  
at> useradd sachin  
at> <Ctrl+D>

👉 Creates a new user named **sachin** at **10:05 PM today**.

at 22:05 tomorrow  
at> tar -cvzf /home/user/nightly\_backup.tar.gz /etc  
at> <Ctrl+D>

👉 Creates a **backup of /etc** at **10:05 PM tomorrow**.

at 4pm 3 days from now  
at> shutdown -r now  
at> <Ctrl+D>

👉 Schedules a **system reboot** at **4:00 PM three days later**.

at 10am December 31  
at> echo "Happy New Year" > /home/user/message.txt  
at> <Ctrl+D>

👉 Writes *Happy New Year* into a file at **10:00 AM on December 31st**.

**Managing at jobs:**

atq # Show all queued jobs  
atrm <job\_id> # Remove a scheduled job

💡 **Tip:** Use at for **one-time tasks** like sending reminders, running a script on a special date, or rebooting after maintenance.

#### 6.2 cron Command – Recurring Jobs

The **cron** service in Linux is used to **schedule tasks that run repeatedly** at specific times. It’s ideal for jobs like:

* Taking **daily/weekly backups**
* Sending **log reports**
* Cleaning up old files every month
* Running scripts at regular intervals

Every user on a Linux system has their own **crontab file** (cron table), which stores their scheduled jobs.

### Editing Your Cron Table

To open and edit your cron table, use:

crontab -e

* This opens your personal cron file in the default text editor (like nano or vim).
* Each line you add here represents **one scheduled job**.
* When you save and exit, the cron service will automatically pick up your changes.

To **view your existing jobs**, run:

crontab -l

### Cron Job Format

Every cron job follows this format:

\* \* \* \* \* command-to-run

The first **five fields** represent **time settings**, and the last part is the **command** you want to execute.

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Meaning | Allowed Values | Example |
| 1st | **Minute** | 0–59 | 30 → at 30 minutes past the hour |
| 2nd | **Hour** | 0–23 | 18 → 6 PM |
| 3rd | **Day of Month** | 1–31 | 10 → on the 10th day of the month |
| 4th | **Month** | 1–12 | 12 → December |
| 5th | **Day of Week** | 0–6 (0 = Sunday, 6 = Saturday) | 6 → Saturday |
| Last | **Command** | Any valid shell command or script | tar -cvzf backup.tar.gz /etc |

### How It Works

* An asterisk (\*) means **“every”** possible value.
  + Example: \* \* \* \* \* → run every minute of every hour, every day.
* You can use **specific numbers** for exact times.
  + Example: 30 18 \* \* \* → run at **6:30 PM every day**.
* You can also use **lists ( , )** or **ranges ( - )**.
  + Example: 0 9,18 \* \* \* → run at **9 AM and 6 PM** daily.
  + Example: 0 9-17 \* \* 1-5 → run every hour from **9 AM to 5 PM, Monday–Friday**.
* You can even use **step values ( / )**.
  + Example: \*/10 \* \* \* \* → run every **10 minutes**.

### Example Cron Jobs

\* \* \* \* \* echo "This runs every minute" >> /home/user/minute.log

👉 Appends a message every minute to minute.log.

30 18 \* \* \* tar -cvzf /home/user/daily\_backup.tar.gz /etc

👉 Runs **daily at 6:30 PM**, creating a backup of /etc.

0 18 \* \* 6 tar -cvzf /home/user/sat\_backup.tar.gz /etc

👉 Runs **every Saturday at 6:00 PM**.

0 0 1 \* \* echo "First day of the month" >> /home/user/monthly.log

👉 Runs **at midnight on the 1st day of every month**.

💡 **Tips for Using Cron Effectively:**

* Always use **absolute paths** in your commands (e.g., /usr/bin/python3 instead of python3).
* Redirect output to a log file so you can check results later. Example:
* 0 2 \* \* \* /usr/bin/python3 /home/user/script.py >> /home/user/script.log 2>&1
* Make sure the **cron service is running**:
* systemctl status cron # Debian/Ubuntu  
  systemctl status crond # RHEL/CentOS