

Linux & Docker

Admin Scripts

Practical shell scripts and how-tos for running a small Linux and Docker home lab: monitoring, backups, scheduled jobs, certificates and media-library housekeeping.

- [Monitoring & Server Setup](#)
 - [Monitor server load, disk, CPU and memory with email alerts](#)
 - [Auto-create home folders for Active Directory users on RHEL / Rocky](#)
 - [A production warning banner, safer aliases and a colour prompt for every login](#)
 - [List listening ports and the programs using them](#)
 - [Email yourself an internet speed test report](#)
- [Backups](#)
 - [Full operating-system tar backup of a Linux server](#)
 - [MySQL / MariaDB backups to a NAS](#)
 - [Store MySQL / MariaDB backup credentials safely](#)
 - [Back up every database on several PostgreSQL versions](#)
 - [Set up .pgpass for PostgreSQL backups](#)
 - [Back up an OpenProject \(package install\) to a NAS](#)
 - [Back up Docker app SQLite databases without stopping containers](#)
 - [Back up Docker container data folders \(stop, archive, restart\)](#)
 - [Pull server backups onto a Synology NAS with rsync over SSH](#)
- [Scheduling & Certificates](#)
 - [Deploy cron jobs to many servers from one script folder](#)
 - [Reliable Let's Encrypt renewals on RHEL / Rocky](#)
- [Docker & Media Library](#)

- [A menu script for Docker media containers, with a VPN routing check](#)
- [Clean up leftover {tmdb-...} folders in a media library](#)
- [Remove movie folders that have no video files \(safely\)](#)
- [Photos, Files & Raspberry Pi](#)
 - [Set the "date taken" on photos with ExifTool](#)
 - [Convert images in bulk \(convert_images.sh\)](#)
 - [Delete HEIC files that already have a JPG copy](#)
 - [Rename all files in a folder to upper case](#)
 - [Shrink Raspberry Pi SD-card images with PiShrink](#)

Monitoring & Server Setup

Monitor server load, disk, CPU and memory with email alerts

A small shell script, run from cron every 5 minutes, that checks four health numbers and emails you when one goes over its limit. It doesn't need an agent or a monitoring server.

What it checks

Check	How it is read	Default limit
1-minute load average	<code>/proc/loadavg</code>	10.00
CPU use	<code>vmstat 1 2</code> (100 minus idle)	98 %
Root disk use	<code>df --output=pcent /</code>	98 %
Memory use	<code>free</code> (used ÷ total)	98 %

Each limit can be overridden with an environment variable, so one script works on small and large machines:

```
LOAD_THRESHOLD=4 DISK_THRESHOLD=90 RECIPIENTS="you@example.com" /usr/local/bin/loadmon.sh
```

Requirements

`awk`, `df`, `free`, `vmstat` (procps), `bc`, `logger`, a `mail` command (s-nail) with a working mail relay, and optionally `sar` (sysstat) to include recent history in the alert email.

Check for every tool at start-up, including `bc`, so a missing tool fails loudly instead of silently skipping alerts:

```
for c in awk df free vmstat bc mail logger; do
  command -v "$c" >/dev/null || { logger -t loadmon "missing $c"; exit 1; }
done
```

Reading the numbers

```
load=$(cut -d' ' -f1 /proc/loadavg)
cpu=$(vmstat 1 2 | awk 'END {print 100 - $15}')
disk=$(df --output=pcent / | tail -1 | tr -dc '0-9')
mem=$(free | awk '/^Mem:/ {printf "%d", $3 * 100 / $2}')
if (( $(echo "$load > $LOAD_THRESHOLD" | bc -l) )); then ... fi
```

Avoid parsing `top -bn1` for CPU. Its first frame is an average since boot, and its column positions shift.

Schedule it

```
/etc/cron.d/loadmon:
```

```
*/* * * * * root /usr/local/bin/loadmon.sh >/dev/null 2>&1
```

Tips

- **Throttle alerts.** Write a timestamp file per check (for example `/var/lib/loadmon/disk.last`), and only email again after an hour. Otherwise a full disk sends 12 emails an hour.
- **Rotate the log.** One line every 5 minutes is about 100,000 lines a year. Add `/etc/logrotate.d/loadmon`.
- **Check more than `/`.** Loop over `/var`, `/home` and data volumes too.
- Use `hostname -s` in the subject line so you can tell servers apart.

Auto-create home folders for Active Directory users on RHEL / Rocky

When Linux servers are joined to Active Directory (SSSD or winbind), domain users who log in for the first time have no `/home/<user>`.

Recommended: use the built-in feature

RHEL, Rocky and Alma 8/9 ship this already:

```
sudo dnf install oddjob-mkhomedir
sudo authselect enable-feature with-mkhomedir
sudo systemctl enable --now oddjobd
```

The home folder is then created from `/etc/skel` at first login, with the right owner and SELinux context.

Manual fallback (one user)

If you need to create a home folder by hand, quote everything. Group names like `domain users` contain a space.

```
#!/bin/bash
set -euo pipefail
u=${1:?usage: mkhome <user>}
```

```
id "$u" >/dev/null          # make sure the user resolves
install -d -m 700 -o "$u" -g "domain users" "/home/$u"
cp -rT /etc/skel "/home/$u"
chown -R "$u":"domain users" "/home/$u"
restorecon -R "/home/$u"
```

Gotcha

Copying commands from Word or email can turn " into curly "smart quotes". The shell treats them as part of the text, so `chown` gets a group name that doesn't exist. Retype the quotes if a command fails mysteriously.

A production warning banner, safer aliases and a colour prompt for every login

Drop-in files in `/etc/profile.d/` apply to every user's login shell. Here are three small ones.

1. Safer aliases: `/etc/profile.d/custom-aliases.sh`

```
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias vi='vim'
```

These only affect interactive shells, not scripts.

2. Production warning: `/etc/profile.d/custom-prod-warning.sh`

Show it only on hosts that really are production. If every box shows it, people learn to ignore it.

```
[[ $- == *j* && -t 0 && -t 1 ]] || return
grep -qx "$(hostname -s)" /etc/production-hosts 2>/dev/null || return
```

```

RED='\033[1;31m'; NC='\033[0m'
if command -v figlet >/dev/null; then
    figlet -f big PRODUCTION | while IFS= read -r l; do printf "${RED}%s${NC}\n" "$l"; done
else
    echo -e "${RED}***** PRODUCTION SERVER *****${NC}"
fi
echo -e "${RED}This is a LIVE system. Think twice before running commands.${NC}"

```

3. Two-line colour prompt:

/etc/profile.d/custom-prompt-and-colors.sh

```

if [[ $- == *i* && -t 0 && -t 1 && -z ${SSH_ORIGINAL_COMMAND:-} ]]; then
    R=$(tput sgr0); B=$(tput bold); G=$(tput setab 8)
    RED=$(tput setaf 1); MAG=$(tput setaf 5); GRN=$(tput setaf 2); CYN=$(tput setaf 6)
    PS1="\n[${B}${G}${RED}\u[${MAG}]@[${RED}]\h[${GRN}] : \d \t[${R}]\n[${CYN}][\w][${R}${B}] \\\$ \[${R}]"
    export PS1
fi

```

- Wrap every colour code in `\[ \]`, or line editing gets confused.
- The `-t` and `SSH_ORIGINAL_COMMAND` checks keep `scp`, `rsync` and Ansible sessions clean. Any output from profile scripts breaks them.

Notes

- `tailf` was removed from util-linux; use `tail -f`.
- Files are read in alphabetical order; prefix them (`zz-`) if order matters.

List listening ports and the programs using them

Before you publish a new container port, check what is already listening. `ss` shows everything; this tidies the output into one line per protocol and port.

Quick one-liner

```
sudo ss -Htulnp | awk '{split($5,a,"."); port=a[length(a)]; print $1, port, $7}' | sort -k2,2n -u
```

Example output:

```
tcp 22 users:(("sshd",pid=812,fd=3))
tcp 443 users:(("docker-proxy",pid=1450,fd=4))
udp 53 users:(("dnsmasq",pid=990,fd=5))
```

- `-H` drops the header. `-t`/`-u` select TCP and UDP, `-l` listening sockets only, `-n` numeric ports, and `-p` the process (needs sudo).
- `sort -u` on the port column removes the duplicate IPv4 and IPv6 lines.

Older systems and NAS devices

Where `ss` is missing, use `netstat -tulnp` (net-tools). The columns differ: the local address is field 4 and the program is field 7.

Colour (optional)

If you add colour, **sort first and colour last**. Sorting text that already contains ANSI codes gives the wrong order. Also, `match(str, re, arr)` in awk is a GNU awk (gawk) feature; busybox and mawk don't have it.

Docker-specific view

```
docker ps --format '{{.Names}}\t{{.Ports}}' | column -t
```

Email yourself an internet speed test report

Run Ookla's speed test on a Linux box and email the result, for example to keep a record of your ISP's performance.

Install the Ookla CLI

Install it from Ookla's package repository, rather than piping a web script into `sudo bash`:

```
# Download the repo setup script, read it, then run it
curl -fsSLo /tmp/ookla.sh https://packagecloud.io/install/repositories/ookla/speedtest-cli/script.rpm.sh
less /tmp/ookla.sh && sudo bash /tmp/ookla.sh
sudo dnf install speedtest
```

To check whether a package is installed, use `rpm -q speedtest`. `rpm -qa speedtest` always returns success, even when the package is missing.

Run and send

```
#!/bin/bash
set -euo pipefail
tmp=$(mktemp -d); trap 'rm -rf "$tmp"' EXIT
speedtest --accept-license --accept-gdpr > "$tmp/speedtest.txt"
enscript -q -p - "$tmp/speedtest.txt" | ps2pdf - "$tmp/speedtest.pdf"
mutt -s "Speed test $(hostname -s) $(date +%F)" -a "$tmp/speedtest.pdf" -- you@example.com <
"$tmp/speedtest.txt"
```

Points to note:

- Run `mutt` in the **foreground**. If you background it with `&` and then delete the attachment, the email can go out without it.
- Use `mktemp` rather than fixed names in `/tmp`.

Requirements

`speedtest`, `enscript`, `ghostscript` (`ps2pdf`), `mutt` and a working mail relay.

Schedule it (optional)

```
0 7 * * * root /usr/local/bin/speedtest-report.sh
```

Backups

Full operating-system tar backup of a Linux server

A simple bare-bones backup: one compressed tarball of the whole root filesystem, minus the folders that are virtual, temporary or backed up separately.

Exclude list

`/etc/backup.exclude` :

```
/proc/*  
/sys/*  
/dev/*  
/run/*  
/tmp/*  
/var/tmp/*  
/usr/tmp/*  
/mnt/*  
/media/*  
/lost+found  
/var/cache/*  
/swapfile
```

Keep `/etc/fstab` in the backup; you will want it for a restore.

The backup

```
#!/bin/bash  
set -euo pipefail  
umask 077 # the archive contains /etc/shadow and keys  
dst=/mnt/backup/$(hostname -s)/system
```

```
mountpoint -q /mnt/backup
mkdir -p "$dst"
nice -n 19 tar --one-file-system -cpjf "$dst/${date +%F}.system.tar.bz2" \
  --exclude-from=/etc/backup.exclude -C / .
find "$dst" -type f -name '*.tar.bz2' -mtime +3 -delete
```

- Use `--one-file-system` to stay on the root filesystem. In modern GNU tar, `-l` means `--check-links`, **not** "one filesystem".
- `umask 077` matters. The archive holds password hashes, SSH host keys and database credential files, so it must not be readable by everyone on the NAS share.
- `zstd` (`tar -I zstd`) is much faster than `bzip2` if it is available.

Schedule

```
0 0 * * 0 root /usr/local/bin/backup_system.sh >/dev/null 2>&1
```

Restore

Boot a rescue system, mount the new root at `/mnt/sysimage`, then run `tar -xpf <archive> -C /mnt/sysimage`. Reinstall the boot loader, and fix `/etc/fstab` UUIDs if the disks changed.

MySQL / MariaDB backups to a NAS

Dump every database to its own compressed file, keep a few days, and fail loudly when something is wrong.

Requirements

- `mysqldump` or `mariadb-dump`. Newer MariaDB installs may only have `mariadb-dump`, so detect which one exists.
- `gzip`.
- A stored login for root (see *Store MySQL backup credentials safely*).
- A mounted backup share, for example `/mnt/backup`.

The core loop

```
DUMP=$(command -v mysqldump || command -v mariadb-dump) || exit 1
CLI=$(command -v mysql || command -v mariadb) || exit 1
mountpoint -q /mnt/backup || { logger -t mysql-backup "backup share not mounted"; exit 1; }

dbs=$(("$CLI" -N -e 'SHOW DATABASES') || { logger -t mysql-backup "cannot connect"; exit 1; })
for db in $dbs; do
    case $db in information_schema|performance_schema|sys) continue;; esac
    "$DUMP" --single-transaction --routines --triggers --complete-insert "$db" \
        | gzip > "/mnt/backup/${hostname -s}/${date +%Y.%m.%d_%H}.${db}.sql.gz" \
        || logger -t mysql-backup "dump of $db failed"
done
find "/mnt/backup/${hostname -s}" -maxdepth 1 -type f -name '*.sql.gz' -mtime +7 -delete
```

- `--single-transaction` gives a consistent dump of InnoDB tables without locking them.

- Check the `SHOW DATABASES` step. If the login fails and you don't check, the loop silently backs up nothing.
- Add `set -o pipefail` so a failed dump isn't hidden by a successful `gzip`.

Schedule

Run it only on servers that actually run MySQL:

```
0 * * * * root /usr/local/bin/backup_mysql.sh >/dev/null 2>&1
```

Hourly dumps kept for 7 days is 168 copies of each database. Daily, or hourly with fewer days kept, is usually plenty.

Restore

```
gunzip -c 2026.01.01_03.mydb.sql.gz | mysql mydb
```

Store MySQL / MariaDB backup credentials safely

Backup scripts shouldn't contain a database password. Store the login once for the account that runs the backups (usually root), and let `mysql` and `mysqldump` read it automatically.

MySQL: `mysql_config_editor`

```
sudo mysql_config_editor set --login-path=client --host=localhost --user=backup --password
sudo mysql_config_editor print --all      # passwords are shown masked
```

This writes an obfuscated `~/.mylogin.cnf` with mode 600. `mysql` and `mysqldump` use the `client` login path by default.

MariaDB: `~/.my.cnf`

MariaDB doesn't ship `mysql_config_editor`, so use an option file instead:

```
sudo install -m 600 /dev/null /root/.my.cnf
sudo tee /root/.my.cnf >/dev/null <<'CNF'
[client]
user=backup
password="your password here"
CNF
```

Quote the password if it contains `#` or `;`.

Create a least-privilege backup account

Send the SQL on stdin rather than with `mysql -e "..."`, so the password doesn't show up in `ps`:

```
sudo mysql <<'SQL'
CREATE USER 'backup'@'localhost' IDENTIFIED BY '...';
GRANT SELECT, SHOW VIEW, TRIGGER, LOCK TABLES, EVENT, PROCESS, RELOAD ON *.* TO
'backup'@'localhost';
SQL
```

Test

```
sudo mysql -e 'SELECT CURRENT_USER();'
```

Back up every database on several PostgreSQL versions

On RHEL, Rocky and Alma, PGDG packages let PostgreSQL 16 and 17 (for example) run side by side, each on its own port with its own binaries in `/usr/pgsql-<ver>/bin`. One script can back up both.

Describe your instances

```
PG_INSTANCES=(  
  "5432|usr/pgsql-16/bin|pg16"  
  "5433|usr/pgsql-17/bin|pg17"  
)
```

Use each instance's own `pg_dump`. A newer `pg_dump` can dump an older server, but not the other way round.

The loop

Run it as the `postgres` user so it connects over the local socket with peer authentication. No password is needed.

```
for inst in "${PG_INSTANCES[@]}"; do  
  IFS='|' read -r port bin name <<<"$inst"  
  out=/mnt/backup/$(hostname -s)/$name; mkdir -p "$out"  
  "$bin/pg_dumpall" -p "$port" --globals-only | gzip > "$out/$(date +%F_%H).globals.sql.gz"  
  while read -r db; do  
    "$bin/pg_dump" -p "$port" "$db" | gzip > "$out/$(date +%F_%H).$db.sql.gz"  
  done < <("$bin/psql" -p "$port" -Atc \  
    "SELECT datname FROM pg_database WHERE NOT datistemplate AND datname <> 'postgres'")  
done  
find /mnt/backup/$(hostname -s) -type f -name '*.sql.gz' -mtime +7 -delete
```

- **Back up the globals.** `pg_dump` doesn't include roles, passwords or grants. Without `pg_dumpall --globals-only`, a restore to a new server has no logins.
- Check that the backup share is mounted (`mountpoint -q`), and that the `postgres` user can write to it.
- Use `flock` to stop runs from overlapping.

Schedule

`/etc/cron.d/backup-postgres`:

```
0 * * * * postgres /usr/local/bin/backup_postgres.sh >/dev/null 2>&1
```

Restore

```
gunzip -c 2026-01-01_03.globals.sql.gz | psql -p 5433 postgres
createdb -p 5433 mydb && gunzip -c 2026-01-01_03.mydb.sql.gz | psql -p 5433 mydb
```

Set up .pgpass for PostgreSQL backups

If your backups connect over TCP or as a user other than `postgres`, `pg_dump` needs a password. Put it in a `.pgpass` file **in the home folder of the user that runs the backup**, not in the script.

Peer authentication first

If the backup runs as the `postgres` OS user over the local socket, peer authentication already works and you don't need `.pgpass` at all.

Format

One line per server, port, database and user:

```
hostname:port:database:username:password
localhost:5432:*:backup:secret
localhost:5433:*:backup:secret
```

Create it safely

Set a restrictive umask first, so the file is never briefly readable by others:

```
sudo -u postgres bash -c 'umask 077; cat >> ~/.pgpass' <<'EOF2'
localhost:5432:*:backup:secret
EOF2
sudo -u postgres chmod 600 ~/.pgpass
```

libpq ignores the file if its mode is wider than 0600.

Common mistake

Writing `/root/.pgpass` while the cron job runs as `postgres` (home folder `/var/lib/pgsql`). The backup never sees that file. Check which user and home folder your job actually uses.

Test

```
sudo -u postgres psql -h localhost -p 5433 -U backup -c 'select 1' postgres
```

Back up an OpenProject (package install) to a NAS

The packaged OpenProject (DEB/RPM) has its own backup command. It writes a database dump plus archives of attachments, configuration and repositories.

Run the built-in backup

```
sudo openproject run backup
ls /var/db/openproject/backup/
# a database dump (*.pgdump) plus *.tar.gz archives for attachments, config and repositories
```

Copy it off the server and tidy up

```
#!/bin/bash
set -euo pipefail
SRC=/var/db/openproject/backup
DST=/mnt/backup/${hostname -s}/openproject
mountpoint -q /mnt/backup
openproject run backup
mkdir -p "$DST"
rsync -a "$SRC/" "$DST/"
find "$DST" -type f -mtime +7 -delete
# Clean the local copies too, including the big attachment archives
find "$SRC" -type f \( -name '*.pgdump' -o -name '*.tar.gz' \) -mtime +1 -delete
```

Make sure the local clean-up includes the `*.tar.gz` files. Otherwise the attachment archives build up in `/var/db` until the disk is full.

How often?

A full backup includes every attachment, so daily is usually right:

```
0 2 * * * root /usr/local/bin/backup_openproject.sh >/dev/null 2>&1
```

Restore

Follow the OpenProject documentation for "Restoring an installation". In short: stop the services, restore the `.pgdump` with `pg_restore`, unpack the attachments and configuration archives, then run `openproject configure`.

Back up Docker app SQLite databases without stopping containers

Most self-hosted media apps (Plex, Jellyfin, Sonarr, Radarr, Lidarr, Prowlarr, Bazarr, Tautulli, Overseerr/Jellyseerr, SABnzbd, Calibre and others) keep their state in SQLite files under their config volume. Copying those files with `cp` or `tar` while the app is running can give you a corrupt copy. SQLite's **online backup** API makes a consistent copy while the app keeps running.

The core idea

```
sqlite3 -readonly /data/sonarr/config/sonarr.db ".timeout 60000" ".backup '/tmp/sonarr.db'"
sqlite3 /tmp/sonarr.db "PRAGMA quick_check;"      # should print: ok
gzip -c /tmp/sonarr.db > /mnt/backup/$(hostname -s)/sonarr/$(date +%Y.%m.%d_%H).sonarr.db.gz
```

`.timeout` makes sqlite wait for the app's write locks instead of failing.

A script around it

Keep a simple list of `app|path|check` entries at the top, for example:

```
DBS=(
  "plex|$DATA/plex/Library/Application Support/Plex Media Server/Plug-in
  Support/Databases/com.plexapp.plugins.library.db|0"
  "sonarr|$DATA/sonarr/config/sonarr.db|1"
  "radarr|$DATA/radarr/config/radarr.db|1"
)
```

The last field turns the integrity check on or off. Plex's database uses custom tokenizers that make `quick_check` report false errors, so set it to 0 for Plex.

Good habits worth building in:

- **Refuse to run if the backup share isn't mounted:** `mountpoint -q /mnt/backup || exit 1`. Otherwise backups fill the local disk.
- **Check free space** before starting (for example, require 10 GB).
- **Run gently** with `nice -n 19 ionice -c3`.
- **Prevent overlapping runs** with `flock -n`.
- **Write to `file.part`, then `mv`**, so a half-written backup never looks complete.
- **Also copy the app's own backup zips** (the *arr apps write them to `config/Backups`).
- **Add a `--list` option** that shows what would be backed up and writes nothing.
- **Keep N days and delete older copies:** `find ... -name '*.gz' -mtime +14 -delete`.

Schedule

```
30 1 * * * root /usr/local/bin/backup_docker_dbs.sh >/dev/null 2>&1
```

Restore

1. `docker stop <container>`
2. Move the live `app.db` aside, and remove `app.db-wal` and `app.db-shm`.
3. `gunzip -c <backup>.gz > app.db`
4. `chown <puid>:<pgid> app.db` (the IDs the container runs as).
5. `docker start <container>`

Back up Docker container data folders (stop, archive, restart)

For apps whose data can't be copied safely while they are running, the simple, reliable method is: stop the container, tar its data folder, then start it again. Do this one container at a time, so each app is only down for a moment.

Assumptions

Each container's data lives in `<base>/<container-name>/` (for example `/srv/docker/sonarr`), and backups go to a mounted share at `/mnt/backup`.

The script

```
#!/bin/bash
set -uo pipefail          # no -e: we must always restart the container
BASE=/srv/docker
DEST=/mnt/backup/docker/$(hostname -s)/$(date +%F)
KEEP=14
mountpoint -q /mnt/backup || { echo "backup share not mounted"; exit 1; }
mkdir -p "$DEST"
for c in $(docker ps --format '{{.Names}}'); do
    [[ -d $BASE/$c ]] || { echo "skip $c (no data folder)"; continue; }
    docker stop -t 120 "$c" >/dev/null || { echo "could not stop $c"; continue; }
    if tar -czf "$DEST/$c.tar.gz.part" -C "$BASE" "$c"; then
        mv "$DEST/$c.tar.gz.part" "$DEST/$c.tar.gz"
    else
```

```
echo "backup of $c failed"; rm -f "$DEST/$c.tar.gz.part"
fi
docker start "$c" >/dev/null || echo "WARNING: $c did not restart"
done
find "${DEST%/*}" -type f -name '*.tar.gz' -mtime +$KEEP -delete
find "${DEST%/*}" -mindepth 1 -type d -empty -delete
```

Why no `set -e`

With `set -e`, a failing `tar` (or `tar`'s exit code 1 for "file changed while reading") ends the script **before** `docker start`, leaving the app down. Handle errors explicitly, or add a `trap` that restarts the current container.

Skip containers whose folder isn't there

Check that the data folder exists **before** stopping the container. Otherwise a wrong `BASE` path means every container is stopped and started for nothing.

Stacks with databases

If an app and its database are separate containers, stop the app first, then the database, back up both, and start them in reverse order. Or use a proper database dump; see *Back up Docker app SQLite databases without stopping containers*.

Schedule

```
0 3 * * * root /usr/local/bin/docker_backup.sh >> /var/log/docker-backup.log 2>&1
```

Restore

`docker stop app` , then `tar -xzf app.tar.gz -C /srv/docker` , then `docker start app` .

Pull server backups onto a Synology NAS with rsync over SSH

Instead of letting servers write to the NAS, have the NAS **pull** from them. If a server is compromised, it then has no access to the backups.

One-time SSH key setup

1. On the NAS, enable SSH (Control Panel → Terminal & SNMP).
2. Create a key for the account that runs the task:

```
ssh-keygen -t ed25519 -f ~/.ssh/backup_pull -N ''
```

3. On the server, add the public key for a backup user. Restrict it if you can, for example `command="rrsync -ro /"` (rrsync ships with rsync).

The script (on the NAS)

```
#!/bin/bash
set -uo pipefail
SRC=backup@<server>
DST=/volume1/<backup-share>/<server>
LOG=/volume1/<backup-share>/logs
mkdir -p "$LOG"
day=$(date +%A)
for path in /etc/httpd /var/www/html /opt; do
  rsync -a --delete -e "ssh -i ~/.ssh/backup_pull" \
    --log-file="$LOG/${basename "$path"}_$day.log" \
    "$SRC:$path/" "$DST$path/" || echo "pull of $path failed"
```

```
done
```

```
find "$LOG" -type f -name '*.log' -mtime +7 -delete
```

- The first line must be `#!/bin/bash`. Without the `!` it is just a comment, and the script runs in whatever shell starts it.
- Test folders with `[-d path]`. A bare `[ path ]` is always true.

Schedule

Control Panel → Task Scheduler → Create → Scheduled Task → User-defined script. Run it as the user that owns the key.

A mirror is not a backup

`--delete` makes the NAS copy an exact mirror. A file deleted or encrypted on the server disappears from the NAS on the next run. Choose one of:

- Turn on **Btrfs snapshots** for the shared folder (Snapshot Replication); this is the easiest fix.
- Use `rsync --link-dest=<yesterday> ...` into dated folders.
- Use Hyper Backup on top of the mirror.

Scheduling & Certificates

Deploy cron jobs to many servers from one script folder

A lightweight way to manage scheduled jobs on a handful of Linux servers without Ansible. Keep one folder of scripts and cron files on an admin box, copy it to every server, and let each server install the cron files meant for it.

Folder layout

```
/opt/scripts/  
  backup_scripts/...  
  crontab_scripts/  
    cron0hourly      # all servers  
    cronloadmonitor  # all servers  
    web01_croncertbot # only on the host whose short name is "web01"  
    db01_cronbackupmysql # only on db01  
  profile.d/*.sh      # login scripts for every server
```

- **Naming rule:** files starting with `cron` go to every server. Files named `<shorthost>_<name>` go only to that host.
- Every file uses `/etc/cron.d` format, which includes the user field:

```
*/5 * * * * root /opt/scripts/loadmon.sh >/dev/null 2>&1
```

- Put shell logic in scripts, never in a cron file. Cronie skips the **whole file** if one line isn't a valid cron entry.

The installer (runs on each server)

Install only files with your own prefix, and remove only your own stale files. Don't wipe `/etc/cron.d`, because packages put their own jobs there (`0hourly`, `raid-check`, monitoring agents).

```
#!/bin/bash
set -euo pipefail
[[ $EUID -eq 0 ]] || exit 1
SRC=/opt/scripts/crontab_scripts
host=$(hostname -s | tr '[:upper:]' '[:lower:]')
want=()
for f in "$SRC"/cron* "$SRC"/"${host}"_*; do
    [[ -f $f ]] || continue
    name="optscripts-$(basename "$f")"
    install -m 644 "$f" "/etc/cron.d/$name"; want+=("$name")
done
for f in /etc/cron.d/optscripts-*; do
    [[ -e $f ]] || continue
    [[ " ${want[*]} " == *" $(basename "$f") "*" ]] || rm -f "$f"
done
for f in /opt/scripts/profile.d/*.sh; do
    install -C -m 644 "$f" /etc/profile.d/
done
```

Run the installer from its own cron file, so you never overwrite root's personal crontab:

```
# /etc/cron.d/optscripts-installer
*/15 * * * * root /opt/scripts/crontab_scripts/install_cron.sh >/dev/null 2>&1
```

If you put a file in `/var/spool/cron/` instead, it must be mode 0600, or cronie refuses it ("BAD FILE MODE").

Pushing the folder

From the admin box, loop over a host list:

```
while read -r h; do
    rsync -a --delete --chmod=D755,F755 --exclude 'secrets/' /opt/scripts/ "root@$h:/opt/scripts/" \
    || echo "push to $h failed"
done < /opt/scripts/hosts.txt
```

- Keep anything containing secrets **out** of the pushed folder.
- `--chmod` stops loose file modes from being copied everywhere.
- Fix ownership at the source: `chown -R root:root /opt/scripts`.

Why cron0hourly?

The installer above leaves stock files alone. If you use a "replace everything" approach instead, remember that the RHEL `cronie` package's `/etc/cron.d/0hourly` is what runs `/etc/cron.hourly`. Put an equivalent back:

```
01 * * * * root run-parts /etc/cron.hourly
```

Better long-term

Once you have more than a few servers, use Ansible (the `cron` and `copy` modules) or a git pull on each host.

Reliable Let's Encrypt renewals on RHEL / Rocky

A certificate can only be renewed in its last 30 days. A cron job that tries once a month can miss that window after a single failure, and the certificate expires.

Use the packaged timer

The EPEL `certbot` package ships a systemd timer that runs twice a day:

```
sudo systemctl enable --now certbot-renew.timer
systemctl list-timers certbot-renew.timer
```

Remove any old monthly certbot cron entries.

Reload the web server after renewal

A renewed certificate isn't used until the web server reloads. Add a deploy hook:

```
sudo install -m 755 /dev/stdin /etc/letsencrypt/renewal-hooks/deploy/reload-web.sh <<'H'
#!/bin/sh
systemctl reload httpd 2>/dev/null || systemctl reload nginx
H
```

Or, for the packaged timer, set it in `/etc/sysconfig/certbot`:

```
DEPLOY_HOOK="--deploy-hook 'systemctl reload httpd'"
```

If you must use cron

```
17 3,15 * * * root /usr/bin/certbot renew -q --deploy-hook "systemctl reload httpd"
```

Test

```
sudo certbot renew --dry-run
```

Docker & Media Library

A menu script for Docker media containers, with a VPN routing check

If you run a group of media containers (the `*arr` apps, download clients, `flaresolverr` and so on), a small menu script saves typing for the common jobs. The most useful part is a check that download clients really leave through the VPN.

Pick the "media" group by name

```
media_containers() {  
  docker ps --format '{{.Names}}' |  
    grep -E 'sonarr|radarr|lidarr|prowlarr|bazarr|sabnzbd|qbittorrent|deluge|transmission|flaresolverr' | sort  
}
```

A simple menu

```
PS3="Choose: "  
select opt in "List containers" "Restart media group" "Check VPN routing" "Quit"; do  
  case $REPLY in  
    1) docker ps --format 'table {{.Names}}\t{{.Status}}' ;;  
    2) read -rp "Restart $(media_containers | wc -l) containers? (y/N) " a  
       [[ $a == [yY] ]] && media_containers | xargs -r docker restart ;;  
    3) vpn_check ;;  
    4) break ;;  
  esac  
done
```

Ask for confirmation before bulk actions. A "restart all" that also **starts stopped containers** will bring back anything you stopped on purpose.

Check that download clients use the VPN

Compare the public IP seen from inside each client with the host's own public IP. They should be **different**.

```
vpn_check() {
  local host_ip c ip
  host_ip=$(curl -fsS https://ifconfig.me)
  for c in qbittorrent deluge transmission sabnzbd; do
    docker inspect "$c" >/dev/null 2>&1 || continue
    ip=$(docker exec "$c" curl -fsS --max-time 10 https://ifconfig.me 2>/dev/null) || ip="(no answer)"
    if [[ $ip == "$host_ip" ]]; then echo "LEAK $c: $ip (same as host)"
    else echo "OK $c: $ip"; fi
  done
}
```

- Many images don't include `curl`. Use `wget -qO-`, or run the check from the VPN container (gluetun) that the clients share.
- Inside `[[ ... ]]`, put spaces before `]]`. Writing `"$ip"]` is a syntax error that only shows up when that branch runs.

Small bash pitfalls in scripts like this

- Send status messages to **stderr** (`>&2`) in functions whose stdout you capture with `$(...)`.
- Counters set inside `cmd | while ...` are lost, because the loop runs in a subshell. Use `while ...; done <<(cmd)` instead.
- `local x=$(cmd); echo $?` always reports 0. Declare the variable first: `local x; x=$(cmd)`.

Clean up leftover {tmdb-...} folders in a media library

Media managers (Radarr, Sonarr, Jellyfin) name folders like `Movie (2020) {tmdb-12345}`. When an item is removed or re-imported, a folder can be left behind with just artwork or an `.nfo` file. This how-to finds those folders and removes them, after showing you the list first.

Find candidates (dry run)

A candidate is a folder with `{` or `}` in its name, no subfolders, exactly two files, and no video file:

```
#!/bin/bash
root=${1:?usage: tmdb-clean <library> [--delete]}
find "$root" -mindepth 1 -type d \( -name '*{' -o -name '}' \) -print0 |
while IFS= read -r -d '' d; do
    subs=$(find "$d" -mindepth 1 -maxdepth 1 -type d | wc -l)
    files=$(find "$d" -maxdepth 1 -type f | wc -l)
    vids=$(find "$d" -maxdepth 1 -type f \( -iname '*.mkv' -o -iname '*.mp4' -o -iname '*.avi' -o -iname '*.m4v' \) |
wc -l)
    if (( subs == 0 && files == 2 && vids == 0 )); then
        if [[ ${2:-} == --delete ]]; then rm -rf -- "$d"; echo "deleted: $d"
        else echo "would delete: $d"; fi
    fi
done
```

Use it

```
./tmdb-clean.sh /mnt/media/movies      # review the list
./tmdb-clean.sh /mnt/media/movies --delete # then delete
```

- The **video check** is important. Without it, a folder with one movie and one subtitle file also matches "exactly two files".
- `-print0` with `read -d "` handles any file name safely.
- Rescan the library in your media server afterwards.

Remove movie folders that have no video files (safely)

Over time a movie library collects folders that hold only posters, `.nfo` files or subtitles, because the movie itself was deleted or moved. This finds them, shows the list, and deletes only after you confirm.

The logic

A folder is a candidate when **nothing under it** is a video file. Never include the library root itself (`-mindepth 1`).

```
#!/bin/bash
set -euo pipefail

root=${1:?usage: clean-empty-movies <library> [--delete]}

has_video() {
    find "$1" -type f \( -iname '*.mkv' -o -iname '*.mp4' -o -iname '*.m4v' -o -iname '*.avi' -o -iname '*.wmv' \) \
        -print -quit | grep -q .
}

mapfile -d '' dirs < <(find "$root" -mindepth 1 -depth -type d -print0)

todo=()
for d in "${dirs[@]}"; do
    has_video "$d" || todo+=("$d")
done

printf '%s\n' "${todo[@]}"

[[ ${2:-} == --delete ]] || { echo "Dry run. Re-run with --delete."; exit 0; }

read -rp "Delete ${#todo[@]} folders? (y/N) " a

[[ $a == [yY] ]] || exit 0

for d in "${todo[@]}"; do rm -rf -- "$d"; done
```

Key points

- **Get the test the right way round.** `has_video` returns success (0) when a folder *has* video, and you delete when it does **not**. If you invert this by mistake, the list holds every real movie folder. That's why the script defaults to a dry run; always read the list.
- `-print -quit` stops at the first video found, which is fast on big libraries.
- Subtitle-only folders (`.srt`) count as "no video" here. Adjust the pattern if you keep stray subtitles on purpose.
- Rescan the library in Plex or Jellyfin afterwards.

Photos, Files & Raspberry Pi

Set the "date taken" on photos with ExifTool

Scanned or undated photos often have no EXIF date, so photo libraries (Immich, PhotoPrism, Synology Photos) sort them by file date. ExifTool can set it.

Install

```
sudo dnf install perl-Image-ExifTool    # RHEL/Rocky (EPEL)
sudo apt install libimage-exiftool-perl # Debian/Ubuntu
```

Set one fixed date on every JPG in a folder (recursively)

```
exiftool -r -ext jpg -ext jpeg -AllDates="2005:11:06 03:20:00" /path/to/photos
```

`-AllDates` sets `DateTimeOriginal`, `CreateDate` and `ModifyDate` together.

Copy `DateTimeOriginal` into the other date tags

Quote the argument. Unquoted, bash reads `<` as "read input from a file".

```
exiftool -r -ext jpg '-AllDates<DateTimeOriginal' /path/to/photos
```

Backups

ExifTool keeps a `name.jpg_original` copy of every file it changes. Once you have checked the result:

```
exiftool -r -delete_original! /path/to/photos
```

Or use `-overwrite_original` up front if you have your own backup.

As a tiny script

```
#!/bin/bash
dir=${1:-.}; when=${2:?usage: setdate <dir> "YYYY:MM:DD HH:MM:SS"}
exiftool -r -ext jpg -ext jpeg -AllDates="$when" "$dir"
```

Convert images in bulk (convert_images.sh)

Location: `/opt/scripts/docker_scripts/convert_images.sh` (68 lines)

Purpose

Converts HEIC photos (iPhone) in a folder to JPG next to the originals, with `heif-convert`.

How it runs

By hand.

Usage

```
/opt/scripts/docker_scripts/convert_images.sh -n /path/to/photos # dry run  
/opt/scripts/docker_scripts/convert_images.sh /path/to/photos
```

Dependencies

`heif-convert` from `libheif-tools` (EPEL).

What it touches

Writes `<name>.jpg` beside each HEIC file. It does not recurse into subfolders and does not delete anything.

Delete HEIC files that already have a JPG copy

Location: `/opt/scripts/docker_scripts/delete_heic_jpg.sh` and `/opt/scripts/docker_scripts/delete_jpg_heic.sh` (56 lines each)

Purpose

Two mirror-image clean-up scripts for use after converting photos. Each works in the **current folder only**:

- `delete_heic_jpg.sh` deletes each `.heic` file that has a matching `.jpg` (keeps the JPG).
- `delete_jpg_heic.sh` deletes each `.jpg` file that has a matching `.heic` (keeps the HEIC).

Extensions are matched in any case.

How it runs

By hand, from inside the photo folder.

Usage

```
cd /path/to/photos
/opt/scripts/docker_scripts/delete_heic_jpg.sh dry-run # show what would go
/opt/scripts/docker_scripts/delete_heic_jpg.sh          # delete
```

Dependencies

bash only.

What it touches

Deletes files in the current folder. The log goes to stderr.

Rename all files in a folder to upper case

Location: `/opt/scripts/renameuppercase.sh` (May 2016, 1 line of logic)

Purpose

Renames every file in the **current folder** to upper case (for example `photo.jpg` becomes `PHOTO.JPG`).

How it runs

By hand, from inside the folder you want to change.

Usage

```
cd /path/to/folder && /opt/scripts/renameuppercase.sh
```

Dependencies

`tr`, `mv`.

What it touches

Every file name in the current folder. There is no undo.

Shrink Raspberry Pi SD-card images with PiShrink

Location: `/opt/scripts/pishrink.sh` (April 2018)

Purpose

A copy of the open-source **PiShrink** tool. It shrinks a Raspberry Pi SD-card image (`.img`) to the smallest size its root filesystem allows. Unless you pass `-s`, it also sets the Pi to grow the filesystem back to full size on first boot.

How it runs

By hand, as root.

Usage

```
sudo /opt/scripts/pishrink.sh pi-backup.img          # shrink in place
sudo /opt/scripts/pishrink.sh -s pi-backup.img small.img  # copy first, no auto-expand
```

Dependencies

`parted`, `losetup`, `tune2fs`, `e2fsck`, `resize2fs`, `md5sum`, `which`.

What it touches

Only the image file. It attaches a loop device, mounts the image in a temporary folder, and may replace `/etc/rc.local` inside the image.