ADMIN:DOCS

Linux Administration #6: Datensicherung und Wiederherstellung

Lerne die Grundlagen der Linux-Datensicherung: Von Backup-Strategien über Automatisierung bis hin zur Wiederherstellung. Mit praktischen Beispielen für Administratoren.

Kategorie: Linux-Administration **Veröffentlicht:** 03.11.2024 **Zuletzt geprüft:** 02.11.2026

Willkommen zum sechsten Teil unserer technischen Wiki-Serie über Linux-Administration!

Nachdem wir in den vorherigen Modulen die Grundlagen, die Benutzerverwaltung, das Prozess- und Ressourcenmanagement, die Netzwerkkonfiguration und das Shell-Scripting erarbeitet haben, widmen wir uns heute einem der kritischsten Pfeiler jedes zuverlässigen Serverbetriebs: der Datensicherung und Wiederherstellung.

> ⚠ **Hinweis:** In diesem Artikel verwenden wir Ubuntu/Debian als Beispiel-Distribution. Die grundlegenden Konzepte sind auf allen Linux-Systemen gleich, aber die Installation von Paketen und einige Konfigurationspfade können sich je nach Distribution unterscheiden. Wenn du eine andere Distribution verwendest, konsultiere bitte die entsprechende Dokumentation für die spezifischen Installationsbefehle und Pfade.

Als Linux-Administrator ist die Sicherung und Wiederherstellung von Daten eine deiner wichtigsten Aufgaben.

Stell dir vor, dein System ist wie ein wertvolles Buch – und Backups sind wie Kopien dieses Buches, die du an verschiedenen sicheren Orten aufbewahrst. Wenn das Original beschädigt

Quelle: https://admindocs.de/de/linux-administration/linux-administration-datensicherung-und-wiederherstellung

wird oder verloren geht, kannst du es jederzeit wiederherstellen.

Backup-Strategien

Die Wahl der richtigen Sicherungsmethode bestimmt den Spagat zwischen Speicherplatzbedarf, Dauer des Backup-Fensters und der Zeitspanne, die im Ernstfall für einen Restore benötigt wird (*Recovery Time Objective*, RTO). In modernen Linux-Umgebungen unterscheidet man primär zwischen vier grundlegenden Ansätzen:

1. Vollbackup (Full Backup)

Ein Vollbackup sichert den gesamten definierten Datenbestand vollständig und eigenständig, unabhängig davon, ob sich Dateien seit dem vorherigen Lauf geändert haben.

Vorteile:

- **Maximale Unabhängigkeit:** Jeder Stand ist vollständig autark und kann ohne Zwischenschritte restauriert werden.
- **Einfacher Restore:** Minimale Fehleranfälligkeit bei der Wiederherstellung, da nur ein einziges Archiv oder Image benötigt wird.

Nachteile:

- **Hoher Speicherverbrauch:** Wiederholte Speicherung unveränderter Daten führt rasch zu massivem Kapazitätsbedarf.
- **Lange Laufzeit & hohe I/O-Last:** Erfordert ein großes Zeitfenster und belastet Festplatten-I/O und Netzwerkbandbreite.

2. Differentielles Backup

Ein differentielles Backup sichert alle Daten und Dateien, die seit dem **letzten Vollbackup** modifiziert oder neu angelegt wurden.

- **Tag 1 (Sonntag):** Vollbackup des Systems (100 GB).
- **Tag 2 (Montag):** Differentiell – alle Änderungen seit Sonntag (15 GB).
- **Tag 3 (Dienstag):** Differentiell – alle Änderungen seit Sonntag (25 GB).

Vorteile:

- **Überschaubarer Restore-Aufwand:** Zur Wiederherstellung werden immer nur genau zwei Elemente benötigt: das letzte Vollbackup sowie das aktuellste differentielle Backup.
- **Schneller als tägliche Vollbackups:** Spart Zeit und I/O gegenüber einer ständigen Vollsicherung.

Nachteile:

- **Wachsender Speicherbedarf:** Mit jedem Tag bis zum nächsten Vollbackup wächst die Größe des differentiellen Dumps an.
- **Redundante Übertragungen:** Daten, die sich an Tag 2 geändert haben, werden an Tag 3 erneut mitgesichert.

3. Inkrementelles Backup

Ein inkrementelles Backup sichert ausschließlich jene Blöcke oder Dateien, die sich seit der **unmittelbar vorangegangenen Sicherung** (egal ob Voll- oder Inkrementell-Backup) geändert haben.

- **Tag 1 (Sonntag):** Vollbackup des Bestands (100 GB).
- **Tag 2 (Montag):** Inkrement 1 – Änderungen von Montag (5 GB).
- **Tag 3 (Dienstag):** Inkrement 2 – nur Änderungen von Dienstag (3 GB).

Vorteile:

- **Minimaler Speicherplatz:** Extrem ressourcenschonend, da keine doppelten Stände erfasst werden.
- **Kürzestes Backup-Fenster:** Perfekt für stündliche oder tägliche Automatisierungen über langsame Netzwerkverbindungen.

Nachteile:

- **Verkettete Restore-Reihenfolge:** Für eine vollständige Rekonstruktion wird das Vollbackup plus **jede einzelne** Inkrementstufe in strikt chronologischer Reihenfolge benötigt.

- **Kettenrisiko:** Ist ein einziges Inkrement in der Kette beschädigt, lassen sich spätere Stände oft nicht mehr konsistent wiederherstellen.

4. Snapshot-Backups (Copy-on-Write)

Snapshots frieren den Zustand eines Dateisystems oder Block-Devices zu einem exakten Sekundenbruchteil ein. Moderne Linux-Dateisysteme wie **Btrfs** und **OpenZFS** sowie LVM-Thin-Pools nutzen hierfür das *Copy-on-Write*-Prinzip (CoW):

- Beim Auslösen des Snapshots werden lediglich Metadaten und Blockzeiger dupliziert – der Vorgang dauert wenige Millisekunden und belegt anfangs 0 Bytes zusätzlichen Speicher.
- Erst wenn eine Datei auf dem Produktivsystem modifiziert wird, schreibt das System die neuen Daten in freie Blöcke, während der Snapshot weiterhin auf die unveränderten Originalblöcke zeigt.

Vorteile:

- **Kein I/O-Freeze:** Erstellung in Sekundenbruchteilen ohne spürbare Systemunterbrechung.
- **Konsistente Datengrundlage:** Ideal, um im laufenden Betrieb einen konsistenten Stand einzufrieren und anschließend im Hintergrund per `btrfs send`, `zfs send` oder rsync wegzusichern.
- **Sofortiger Rollback:** Fehlgeschlagene Paket-Upgrades oder Systemänderungen lassen sich innerhalb von Sekunden rückgängig machen.

Nachteile:

- **Kein echtes Backup:** Ein lokaler Snapshot verbleibt auf demselben physischen Speichermedium bzw. Pool. Stirbt die Festplatte oder der Controller, sind Produktivdaten und Snapshots gleichermaßen verloren. Snapshots müssen daher immer auf externe Medien repliziert werden.

Einsatzgebiete:

- Vor Systemupdates

- Entwicklungsumgebungen
- Virtuelle Maschinen

Backup-Tools und -Technologien

Unter Linux steht Administratoren ein ausgereiftes Spektrum an Werkzeugen zur Verfügung – von klassischen POSIX-Kommandozeilenwerkzeugen bis hin zu modernen, deduplizierenden Speicher-Engines mit nativer Cloud- und Objektspeicher-Anbindung.

1. Traditionelle Standardwerkzeuge

rsync (Remote Sync)

Das unverzichtbare Standardwerkzeug für dateibasierte Synchronisation und inkrementelle Backups. `rsync` analysiert Quelldateien anhand von Größe und Modifikationszeitpunkt (mtime) und überträgt über das eigene Delta-Transfer-Protokoll nur geänderte Datenblöcke.

```
# Lokale Verzeichnissynchronisation mit Erhalt aller Rechte (-a) und Statusanzeige (-v)
rsync -av /var/www/ /backup/www/

# Sicheres Remote-Backup über SSH mit Kompression (-z)
rsync -avz -e ssh /var/www/ admin@backup01.intern:/srv/backups/www/

# Exakte Spiegelung inklusive Löschung verwaister Zieldateien (--delete)
rsync -av --delete --exclude='*.tmp' --exclude='cache/' /srv/data/ /backup/data/
```

tar und moderne Kompression (zstd)

`tar` fasst beliebige Verzeichnisbäume in einer einzigen Archivdatei zusammen und erhält dabei Dateieigentümer, Unix-Berechtigungen und Zeitstempel. Während historisch oft `gzip` (`-czf`) oder `bzip2` (`-cjf`) genutzt wurden, gilt heute **Zstandard (`zstd`)** als De-facto-Standard in modernen Linux-Distributionen (Debian, Ubuntu, RHEL). `zstd` erreicht deutlich bessere Kompressionsraten als `gzip` bei um ein Vielfaches höherer Geschwindigkeit und nativer

Mehrkernunterstützung (`-T0`).

```
# Schnelles Archiv mit Zstandard-Kompression auf allen CPU-Kernen erstellen:
tar --zstd -cf backup_$(date +%F).tar.zst /etc /home /var/log

# Zstandard-Archiv entpacken:
tar --zstd -xf backup_2026-11-02.tar.zst -C /zielverzeichnis/

# Archivinhalt verifizieren und auflisten:
tar --zstd -tvf backup_2026-11-02.tar.zst
```

dd (Disk Dump)

Ein elementares Low-Level-Werkzeug für bitgenaue 1:1-Klone von Partitionen oder ganzen Blockgeräten (Master Boot Record, GPT-Tabellen, Dateisystemgrenzen).

> ⚠ **Achtung vor Datenverlust:** `dd` führt keine Validierung durch und überschreibt Zielmedien gnadenlos. Ein falscher Buchstabe bei `of=` (Output File) überschreibt Produktivdaten unwiderruflich. Prüfe Zielgeräte stets vorab mit `lsblk` und `fdisk -l` .

```
# Bitgenaues Image einer Festplatte mit 4 MB Blockgröße und Fortschrittsanzeige erstellen:
sudo dd if=/dev/nvme0n1 of=/backup/images/system_disk.img bs=4M status=progress conv=fsync

# Image auf ein identisches Ersatzlaufwerk zurückspielen:
sudo dd if=/backup/images/system_disk.img of=/dev/nvme1n1 bs=4M status=progress conv=fsync
```

2. Moderne Backup-Engines

Traditionelle `tar`-Archive oder einfache `rsync`-Kopien stoßen bei großen Datenmengen an Grenzen: Sie bieten von Haus aus weder blockbasierte Deduplizierung über mehrere Versionen hinweg noch kryptografisch gesicherte Ende-zu-Ende-Verschlüsselung. Moderne Backup-Lösungen lösen diese Herausforderungen:

BorgBackup (Deduplizierung & Verschlüsselung)

BorgBackup zerlegt Daten in variable Chunks, dedupliziert diese über alle Backup-Generationen hinweg und verschlüsselt die Daten clientseitig mit AES-256 oder ChaCha20-Poly1305. Nur neue oder veränderte Datenblöcke belegen zusätzlichen Speicherplatz im Repository.

```
# Repository mit Passphrase initialisieren (repokey-Modus):
borg init --encryption=repokey-blake2 /srv/backup-repo

# Dedupliziertes Backup mit Statistik und Fortschritt anlegen:
borg create --stats --progress \
    /srv/backup-repo::srv01-{now:%Y-%m-%d} \
    /etc /var/www /home

# Vorhandene Backup-Archive auflisten:
borg list /srv/backup-repo

# Veraltete Stände nach Aufbewahrungsrichtlinie bereinigen (Pruning):
borg prune -v --list /srv/backup-repo --keep-daily=7 --keep-weekly=4 --keep-monthly=12
```

Restic (Cloud- und Objektspeicher-Sicherung)

Restic ist ein eigenständiges Go-Binary ohne externe Laufzeitabhängigkeiten. Es eignet sich hervorragend zur direkten Sicherung in S3-kompatible Objektspeicher (AWS S3, MinIO, Wasabi, Backblaze B2), SFTP-Server oder lokale Repositories. Restic arbeitet stets verschlüsselt und inkrementell auf Chunk-Ebene.

```
# Lokales oder S3-Repository initialisieren:
restic init --repo /srv/restic-repo

# Inkrementelles verschlüsseltes Backup erstellen:
restic -r /srv/restic-repo backup /srv/daten --exclude="*.log"

# Snapshot-Übersicht anzeigen:
restic -r /srv/restic-repo snapshots

# Backup-Integrität auf Bit-Ebene prüfen:
restic -r /srv/restic-repo check
```

Btrfs & ZFS Send/Receive

Auf Servern mit Btrfs oder OpenZFS lassen sich schreibgeschützte Snapshots direkt über einen binären Datenstrom differentiell auf externe Backup-Server replizieren:

```
# Schreibgeschützten Btrfs-Snapshot anlegen:
sudo btrfs subvolume snapshot -r /data /data/.snapshots/snapshot-$(date +%F)

# Snapshot differentiell über SSH an einen Backup-Server übertragen:
sudo btrfs send -p /data/.snapshots/snapshot-gestern /data/.snapshots/snapshot-heute \
    | ssh backupuser@backuphost "btrfs receive /srv/remote-backups/data/"
```

Vertiefung: Die moderne 3-2-1-1-0-Backup-Regel

Als Erweiterung der klassischen 3-2-1-Regel definiert der moderne Standard **3-2-1-1-0** verlässliche Schutzmaßnahmen gegen Ransomware, Hardware-Ausfälle und menschliche Fehlkonfigurationen:

- **3 Kopien aller kritischen Daten:** Mindestens drei Instanzen vorhalten (1x Produktivdaten, 1x lokale Sicherung, 1x externe Sicherung).
- **2 verschiedene Medientypen:** Vermeidung herstellerspezifischer oder technologischer Monokulturen (z. B. lokale NVMe-SSDs kombiniert mit Objektspeicher S3 oder Bandlaufwerken).
- **1 Kopie an einem externen Standort (Off-Site):** Geografische Trennung zur Absicherung gegen lokale Katastrophen (Brand, Diebstahl, Rechenzentrumsausfall).
- **1 unveränderbare oder Air-Gapped Kopie (Immutable):** Schutz vor Ransomware durch WORM-Speicher (*Write Once, Read Many*), S3 Object Lock mit Compliance Mode oder getrennte Offline-Medien, die auch bei kompromittierten Root-Credentials nicht gelöscht werden können.
- **0 Fehler nach Wiederherstellungs-Validierung:** Regelmäßige, automatisierte Integritätsprüfungen und Test-Restores, um sicherzustellen, dass gesicherte Stände tatsächlich funktionsfähig sind.

Vertiefung: Datenbank-Backups im laufenden Betrieb (Hot-Backups)

Dateibasierte Backups von aktiven Datenbankverzeichnissen (wie `/var/lib/mysql` oder `/var/lib/postgresql`) führen unweigerlich zu inkonsistenten oder korrupten Datenständen, da Schreibzugriffe während des Kopierens stattfinden. Daher nutzen Administratoren native Dump-Werkzeuge:

1. MySQL und MariaDB konsistent sichern (`mysqldump`)

```
# Transaktionssicherer Dump aller Datenbanken im laufenden Betrieb:
sudo mysqldump --all-databases --single-transaction --quick \
    --lock-tables=false | gzip > /backup/db/all_databases_$(date +%Y%m%d).sql.gz
```

```
# Einzelne Datenbank wiederherstellen:
gunzip < /backup/db/appdb_20260220.sql.gz | sudo mysql appdb
```

2. PostgreSQL konsistent sichern (`pg_dump` und `pg_dumpall`)

```
# Alle PostgreSQL-Cluster-Datenbanken sichern:
sudo -u postgres pg_dumpall | gzip > /backup/db/postgres_all_$(date +%Y%m%d).sql.gz

# Einzelne Datenbank im Custom-Format (schnell, parallelisierbar) sichern:
sudo -u postgres pg_dump -Fc mydb -f /backup/db/mydb_$(date +%Y%m%d).dump

# Mit pg_restore wiederherstellen:
sudo -u postgres pg_restore -d mydb /backup/db/mydb_20260220.dump
```

Vertiefung: Hardlink-Snapshots mit `rsync --link-dest`

Mit dem Schalter `--link-dest` erstellt `rsync` inkrementelle Snapshots, die sich wie vollwertige Vollbackups verhalten, aber nur für geänderte Dateien neuen Festplattenplatz belegen:

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

BACKUP_PATH="/backup/snapshots"
TODAY=$(date +%Y-%m-%d)
YESTERDAY=$(ls -1d $BACKUP_PATH/20* 2>/dev/null | tail -n 1 || true)

mkdir -p "$BACKUP_PATH/$TODAY"
```

```
if [ -n "$YESTERDAY" ] && [ -d "$YESTERDAY" ]; then
    echo "Verwende $YESTERDAY als Hardlink-Basis..."
    rsync -a --delete --link-dest="$YESTERDAY" /var/www/ "$BACKUP_PATH/$TODAY/"
else
    echo "Initiales Vollbackup..."
    rsync -a /var/www/ "$BACKUP_PATH/$TODAY/"
fi
```

Backup-Automatisierung

Eine verlässliche Backup-Strategie steht und fällt mit ihrer lückenlosen Automatisierung. Manuelle Backups werden im Betriebsalltag vergessen oder bei Zeitdruck aufgeschoben.

1. Robustes Backup-Skript mit Fehlerbehandlung

Moderne Backup-Skripte nutzen strikte Shell-Optionen (`set -Eeuo pipefail`), sauberes Logging und Zstandard-Kompression:

```
#!/usr/bin/env bash
# /usr/local/sbin/backup_system.sh - Vollständiges System-Backup mit zstd
set -Eeuo pipefail

# Konfiguration
readonly SOURCE_DIRS=("/etc" "/var/www" "/home")
readonly BACKUP_DIR="/srv/backups/daily"
readonly DATE="$(date +%Y-%m-%d_%H%M%S)"
readonly LOG_FILE="/var/log/backup_system.log"
readonly RETENTION_DAYS=14
readonly BACKUP_FILE="${BACKUP_DIR}/full_backup_${DATE}.tar.zst"

log() {
    local -r msg="$1"
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] [INFO] ${msg}" | tee -a "${LOG_FILE}"
}
```

```
error_log() {
    local -r msg="$1"
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] [ERROR] ${msg}" | tee -a "${LOG_FILE}" >&2
}

trap 'error_log "Backup durch unerwarteten Fehler in Zeile $LINENO abgebrochen!"' ERR

# Verzeichnis vorbereiten
mkdir -p "${BACKUP_DIR}"

# Speicherplatzprüfung (mindestens 10 GB freier Speicher auf der Zielpartition)
FREE_KB=$(df -P "${BACKUP_DIR}" | awk 'NR==2 {print $4}')
if (( FREE_KB < 10485760 )); then
    error_log "Nicht genügend freier Speicher auf ${BACKUP_DIR} (verfügbar: ${FREE_KB}
KB)!"
    exit 1
fi

log "Starte Sicherung mit Zstandard-Kompression..."
tar --zstd -cf "${BACKUP_FILE}" "${SOURCE_DIRS[@]}" 2>> "${LOG_FILE}"
log "Archiv erfolgreich erzeugt: ${BACKUP_FILE} ($(du -h "${BACKUP_FILE}" | cut -f1))"

# SHA-256 Prüfsumme zur späteren Verifikation generieren
sha256sum "${BACKUP_FILE}" > "${BACKUP_FILE}.sha256"
log "Prüfsumme gesichert in ${BACKUP_FILE}.sha256"

# Veraltete Archive bereinigen
log "Bereinige Archive älter als ${RETENTION_DAYS} Tage..."
find "${BACKUP_DIR}" -name "full_backup_*.tar.zst*" -mtime +"${RETENTION_DAYS}" -delete
log "Backup-Lauf erfolgreich abgeschlossen."
```

2. Zeitsteuerung: Cron vs. systemd-Timer

Während `cron` der historische Standard für zeitgesteuerte Aufträge ist, setzen moderne Enterprise-Distributionen für produktive Backup-Pipelines bevorzugt auf **systemd-Timer**:

Vorteile von systemd-Timern für Backups:

- **`Persistent=true`:** Verpasste Backup-Läufe (z. B. wenn der Server nachts wegen Wartung ausgeschaltet war) werden beim nächsten Systemstart sofort automatisch nachgeholt.

- **Integriertes Logging:** Sämtliche Standard- und Fehlerausgaben landen direkt strukturiert in `journalctl -u backup.service`.
- **Ressourcengrenzen (cgroups v2):** Verhindert, dass speicherintensive Kompressionen den Produktivbetrieb ausbremsen (`MemoryMax=`, `CPUQuota=`, `IOSchedulingClass=idle`).
- **Fehler-Hooks (`OnFailure=`):** Löst bei einem Exit-Code ungleich 0 automatisch Benachrichtigungs-Units aus.

a) Service-Unit anlegen (`/etc/systemd/system/backup.service`):

```
[Unit]
Description=Taegliches System-Backup
After=network.target local-fs.target
OnFailure=backup-notify@%n.service

[Service]
Type=oneshot
ExecStart=/usr/local/sbin/backup_system.sh
IOSchedulingClass=idle
CPUSchedulingPolicy=idle
MemoryMax=2G
```

b) Timer-Unit anlegen (`/etc/systemd/system/backup.timer`):

```
[Unit]
Description=Timer fuer taegliches System-Backup

[Timer]
OnCalendar=*-*-* 02:30:00
Persistent=true
RandomizedDelaySec=15m
```

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

```
# Timer aktivieren und starten:
sudo systemctl daemon-reload
sudo systemctl enable --now backup.timer

# Nächste Ausführungszeitpunkte kontrollieren:
systemctl list-timers backup.timer
```

c) Traditionelle Alternative via Cron (`crontab -e`):

```
# Format: Minute Stunde Tag Monat Wochentag Befehl
30 2 * * * /usr/local/sbin/backup_system.sh >> /var/log/cron-backup.log 2>&1
```

Monitoring und Benachrichtigungen

Backups, die im Hintergrund unbemerkt scheitern, wiegen Administratoren in falscher Sicherheit. Eine zuverlässige Überwachung umfasst drei essenzielle Ebenen:

1. **Ausführungskontrolle:** Wurde das Skript ausgeführt und mit Exit-Code `0` beendet?
2. **Aktualitätsprüfung:** Existiert eine Sicherungsdatei, deren Modifikationszeitpunkt jünger als 26 Stunden ist?
3. **Plausibilität & Mindestgröße:** Weist das Archiv eine realistische Dateigröße auf oder wurde lediglich ein leeres Archiv erzeugt?

```
#!/usr/bin/env bash
# /usr/local/sbin/backup_check.sh - Monitoring-Prüfung für Icinga/Nagios/Zabbix
set -euo pipefail

BACKUP_DIR="/srv/backups/daily"
MIN_BYTES=$((50 * 1024 * 1024)) # Mindestens 50 MB

# Finde neueste Datei innerhalb der letzten 26 Stunden
LATEST=$(find "${BACKUP_DIR}" -type f -name "full_backup_*.tar.zst" -mtime -1 | sort -r |
head -n 1)

if [[ -z "${LATEST}" ]]; then
    echo "CRITICAL: Kein Backup der letzten 24 Stunden gefunden!" >&2
    exit 2
fi

SIZE=$(stat -c %s "${LATEST}")
if (( SIZE < MIN_BYTES )); then
    echo "WARNING: Backup ${LATEST} ist auffaellig klein (${SIZE} Bytes)!" >&2
    exit 1
fi

echo "OK: Aktuelles Backup ${LATEST} (${SIZE} Bytes) vorhanden."
exit 0
```

Backup-Verifizierung und Tests

Die beste Backup-Strategie ist wertlos, wenn sich die Daten im Ernstfall nicht wiederherstellen lassen (*Schrödingers Backup: Der Zustand jedes Backups ist unbekannt, bis ein Restore versucht wird*). Daher ist eine regelmäßige, teils automatisierte Validierung unverzichtbar.

1. Prüfsummen und Archivintegrität

Kryptografische Prüfsummen stellen sicher, dass Backup-Dateien bei der Übertragung über das Netzwerk oder durch fehlerhafte Speichermedien (*Bit Rot*) nicht beschädigt wurden:

```
# SHA-256 Prüfsumme beim Backup erzeugen:
sha256sum full_backup_2026-11-02.tar.zst > full_backup_2026-11-02.tar.zst.sha256

# Prüfsumme automatisiert verifizieren (Rückgabewert 0 bei Erfolg):
sha256sum -c full_backup_2026-11-02.tar.zst.sha256

# Zstandard-Archivintegrität ohne Entpacken auf Dateisystemebene testen:
tar --zstd -tvf full_backup_2026-11-02.tar.zst > /dev/null

# Borg- und Restic-Repositories auf Block- und Chunk-Ebene prüfen:
borg check -v /srv/backup-repo
restic -r /srv/restic-repo check --read-data-subset=10%
```

2. Automatisierte Test-Restores (Sandbox & Container)

Die zuverlässigste Verifikation besteht darin, Archive regelmäßig in einer isolierten Umgebung (z. B. temporäre virtuelle Maschine oder Container) tatsächlich zu entpacken und Kernanwendungen zu starten:

```
# Isoliertes Testverzeichnis im RAM (/dev/shm) oder temporären Speicher anlegen:
TEST_DIR=$(mktemp -d /tmp/restore_test.XXXXXX)

# Archiv entpacken:
tar --zstd -xf /srv/backups/daily/full_backup_2026-11-02.tar.zst -C "${TEST_DIR}"

# Wichtige Schlüsseldateien und Berechtigungen validieren:
test -f "${TEST_DIR}/etc/passwd" && echo "Basis-Dateien vorhanden"
test -d "${TEST_DIR}/var/www" && echo "Webroot vorhanden"

# Aufräumen:
rm -rf "${TEST_DIR}"
```

3. Betriebsdokumentation für den Ernstfall

Eine lückenlose Dokumentation stellt sicher, dass auch vertretende Administratoren unter Hochdruck handlungsfähig sind:

- **Sicherungsübersicht:** Welche Verzeichnisse und Datenbanken werden mit welchem Tool und welchen Exclude-Regeln gesichert?
- **Schlüsselverwaltung:** Wo liegen GPG-Schlüssel, Borg-Passphrasen oder LUKS-Header, und wer hat im Vier-Augen-Prinzip Zugriff darauf?
- **Restore-Runbook:** Präzise, getestete Schritt-für-Schritt-Befehle zur vollständigen System- und Einzeldatei-Wiederherstellung.

Disaster Recovery

Disaster Recovery umfasst alle strategischen und technischen Maßnahmen, um nach einem katastrophalen Schadensfall (Hardware-Defekt, Ransomware-Befall, Rechenzentrumsausfall, menschlicher Bedienfehler) den regulären Betrieb schnellstmöglich und mit minimalem Datenverlust wiederherzustellen.

1. **1. Lageanalyse & Alarmierung** Schadensausmaß feststellen, Ursache isolieren und Notfallteam aktivieren
2. **2. Bereitstellung der Infrastruktur** Ersatzhardware booten, Notfall-VMs hochfahren oder Live-Rettungssystem laden
3. **3. Daten-Restore** Konsistente Voll- und Inkrement-Backups einspielen und Datenbanken einlesen
4. **4. Service-Start & Validierung** Dienste in definierter Reihenfolge starten, Integrität und Ports prüfen
5. **5. Monitoring & Freigabe** Produktiv-DNS umstellen, Last schrittweise freigeben und Logs überwachen

Quelle: https://admindocs.de/de/linux-administration/linux-administration-datensicherung-und-wiederherstellung

1. Strategische Kennzahlen: RPO und RTO

Jedes Notfallkonzept orientiert sich an zwei zentralen Service Level Agreements (SLAs):

- **RPO (Recovery Point Objective):** Der maximal akzeptable Datenverlust, gemessen in Zeit. Ein RPO von 4 Stunden bedeutet, dass bei einem Ausfall höchstens die Daten der letzten 4 Stunden verloren gehen dürfen (erfordert mindestens 4-stündliche Inkremente oder synchrone Replikation).
- **RTO (Recovery Time Objective):** Die maximal tolerierbare Ausfallzeit bis zur Wiederaufnahme des Geschäftsbetriebs. Ein RTO von 1 Stunde verlangt automatisierte Restore-Skripte oder Hot-Standby-Failover.

Eskalationsstufen im Vorfallmanagement:

- **Stufe 1 (First Responder):** On-Call-Administrator stellt den Ausfall fest, isoliert potenziell kompromittierte Systeme vom Netzwerk und dokumentiert den Erstbefund.
- **Stufe 2 (Technische Eskalation):** Spezialisten für Linux-Kernel, Storage, Netzwerk und Datenbanken beginnen mit der gezielten Wiederherstellung.
- **Stufe 3 (Management- & Krisenstab):** Information der Geschäftsführung, Kundenkommunikation und rechtliche Bewertung (z. B. DSGVO-Meldepflicht bei Datenlecks).

2. Technische System- und Daten-Wiederherstellung

Im Notfall unterscheidet man zwischen der Rekonstruktion des Betriebssystems und dem Zurückspielen der reinen Anwendungsdaten:

a) System-Wiederherstellung (Bare Metal & VM):

1. Booten eines neutralen Rettungssystems (z. B. Debian Live, Arch Live oder Hetzner/Gandi Rescue-Modus).
2. Partitionierung und Formatierung des Speichers (`parted` , `mkfs.ext4` , `mkfs.btrfs` oder ZFS-Pool-Import).
3. Wiederherstellung der Kernkonfiguration (`/etc`) und fstab-Anpassung an neue UUIDs (`blkid`).
4. Bootloader (`grub-install`) neu schreiben und System neu starten.

b) Daten-Wiederherstellung aus Archiven:

```
# Zstandard-Vollbackup in Zielverzeichnis restaurieren:
sudo tar --zstd -xf /srv/backups/daily/full_backup_2026-11-02.tar.zst -C /recovery/

# Dateiberechtigungen und Eigentümer überprüfen:
sudo chown -R www-data:www-data /recovery/var/www/
sudo chmod -R u=rwX,go=rX /recovery/var/www/
```

Bei inkrementellen Ketten ohne deduplizierende Engine müssen zunächst das letzte Vollbackup und anschließend alle darauffolgenden Inkremente in chronologischer Reihenfolge eingespielt werden. Bei Werkzeugen wie **Borg** oder **Restic** entfällt dieser Aufwand: Jeder Snapshot verhält sich wie ein vollständiges Vollbackup und wird mit einem einzigen Befehl wiederhergestellt:

```
# Borg-Snapshot direkt an Zielort extrahieren:
borg extract --progress /srv/backup-repo::srv01-2026-11-02 /var/www

# Restic-Snapshot direkt an Zielort restaurieren:
restic -r /srv/restic-repo restore latest --target /
```

Best Practices

Die langfristige Stabilität einer Backup-Infrastruktur basiert auf festen organisatorischen und technischen Sicherheitsprinzipien:

1. Verschlüsselung: Symmetrisch vs. Asymmetrisch

Backup-Medien lagern häufig auf externen Servern, NAS-Systemen oder Cloud-Speichern. Eine unverschlüsselte Sicherung stellt ein massives Sicherheitsrisiko dar.

Symmetrische Verschlüsselung

Schlüssel-Architektur
: Ein geheimer Schlüssel / Passphrase für Ver- und Entschlüsseln

Einsatz im Backup
: Lokale Disks, BorgBackup, Restic, LUKS-Festplattenverschlüsselung

Sicherheitsvorteil
: Sehr hohe Performance dank CPU-Hardwarebeschleunigung (AES-NI)

Asymmetrische Verschlüsselung

Schlüssel-Architektur
: Schlüsselpaar: Öffentlicher Schlüssel (Public) & privater Schlüssel (Private)

Einsatz im Backup
: Automatisierte Server (nur Public Key zum Verschlüsseln auf dem Client)

Sicherheitsvorteil
: Selbst bei voller Root-Kompromittierung des Backup-Clients unlesbar

a) Unattended Backup mit asymmetrischem GPG-Schlüssel:

Auf einem automatisierten Server sollte niemals der private Entschlüsselungsschlüssel hinterlegt werden. Es genügt der öffentliche GPG-Schlüssel (`backup-pub.gpg`):

```
# Daten im Stream komprimieren und mit öffentlichem Schlüssel verschlüsseln:
tar --zstd -cf - /var/www | gpg --encrypt --recipient "backup-key@firma.intern" \
    --output /srv/backups/www_backup_$(date +%F).tar.zst.gpg

# Die Entschlüsselung gelingt ausschließlich mit dem isoliert verwahrten Private Key:
gpg --decrypt /srv/backups/www_backup_2026-11-02.tar.zst.gpg | tar --zstd -xf - -C
/target/
```

b) Verschlüsselte Repositories mit Borg:

```
# Repository mit Argon2/BLAKE2 und Repokey initialisieren:
borg init --encryption=repokey-blake2 /srv/backup-repo

# Verschlüsseltes Backup erstellen:
borg create /srv/backup-repo::srv01-{now:%Y-%m-%d} /var/www
```

2. Aufbewahrungsfristen und das Generationenprinzip (GFS)

Die richtige Aufbewahrungsdauer balanciert Speicherplatz gegen die Anforderung ab, auch ältere, unbemerkt aufgetretene Beschädigungen wiederherzustellen:

- **Tägliche Backups (Sohn):** Aufbewahrung für 7 bis 14 Tage für schnelle Wiederherstellung kürzlich gelöschter Dateien.
- **Wöchentliche Backups (Vater):** Aufbewahrung für 4 bis 8 Wochen (jeden Sonntag erstellt).
- **Monatliche Backups (Großvater):** Aufbewahrung für 12 bis 24 Monate zur Jahresabgrenzung und Archivierung.
- **Jährliche Archivstände:** 5 bis 10 Jahre (für steuer- oder handelsrechtliche Aufbewahrungspflichten).

```
# Automatisiertes Pruning mit BorgBackup (Großvater-Vater-Sohn):
borg prune -v --list /srv/backup-repo \
    --keep-daily=7 \
    --keep-weekly=4 \
    --keep-monthly=12 \
    --keep-yearly=3
```

3. Betriebsdokumentation

Eine vollständige Betriebsdokumentation umfasst:

- **Hardware & Netzwerk:** MAC-Adressen, VLAN-Zugehörigkeiten, Festplattenlayouts (`lsblk -f`), UUIDs.
- **Dienste & Ports:** Welche Services müssen nach dem Booten in welcher Reihenfolge laufen?
- **Schlüssel-Verwahrung:** Sichere Hinterlegung von Master-Keys (z. B. in einem Offline-Tresor oder Secret Management).
- **Kontaktkette:** Klare Telefonnummern und Stellvertreterregelungen für den Notfall.

Übung: Backup-Pipeline mit Verschlüsselung und Monitoring einrichten

Szenario: Du richtest für einen Unternehmens-Server eine automatisierte Backup-Pipeline ein. Die Anwendungsdaten unter `/var/www` und `/etc` müssen verschlüsselt gesichert, veraltete Stände bereinigt und der Erfolg im System-Monitoring erfasst werden.

Kernanforderungen:

- Erstellung eines verschlüsselten Archivs mit moderner Kompression.
- Getrennte Verzeichnisse für Daten, Schlüssel und Protokolle.
- Status-Prüfung auf Vorhandensein und Plausibilität der Archivgröße.

Aufgaben

1. **Struktur vorbereiten:**

- Verzeichnishierarchie unter `/srv/backups/` mit restriktiven Rechten anlegen (`chmod 700`).
- GPG-Schlüsselpaar erzeugen und den öffentlichen Schlüssel exportieren.

1. **Backup-Skript schreiben:**

- Shell-Skript mit Fehlerbehandlung und Zstandard-Kompression erstellen.

- Verschlüsselung über Pipe an GPG durchführen.

1. **Monitoring-Skript implementieren:**

- Prüfen, ob ein aktuelles Archiv existiert und eine Mindestgröße aufweist.

- Backup-Verzeichnisse anlegen
- Berechtigungen setzen
- Verschlüsselung einrichten

2. Backup-Skripte entwickeln:

- Tägliches inkrementelles Backup
- Wöchentliches Vollbackup
- Automatische Rotation alter Backups

3. Monitoring einrichten:

- Backup-Status überwachen
- E-Mail-Benachrichtigungen konfigurieren
- Logging implementieren

Mögliche Lösung

1. Backup-Struktur und Schlüssel erstellen

```
# Verzeichnisstruktur mit restriktiven Berechtigungen anlegen:
sudo mkdir -p /srv/backups/{daily,weekly,logs}
sudo mkdir -p /etc/backup-security

sudo chmod 700 /etc/backup-security
sudo chmod 750 /srv/backups

# Asymmetrischen GPG-Schlüssel für Unattended Backups generieren:
gpg --batch --passphrase '' --quick-generate-key "backup-service@firma.intern" default
```

```
default never

# Öffentlichen Schlüssel für den Backup-Agenten exportieren:
gpg --armor --export "backup-service@firma.intern" > /etc/backup-security/backup-pub.asc
```

2. Backup-Skript entwickeln

```
#!/usr/bin/env bash
# /usr/local/sbin/backup_pipeline.sh
set -Eeuo pipefail

SOURCE_DIRS=("/etc" "/var/www")
BACKUP_DIR="/srv/backups/daily"
LOG_FILE="/srv/backups/logs/backup.log"
DATE="$(date +%Y-%m-%d_%H%M)"
RECIPIENT="backup-service@firma.intern"
ARCHIVE_NAME="backup_${DATE}.tar.zst.gpg"
TARGET="${BACKUP_DIR}/${ARCHIVE_NAME}"

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

log "Starte verschlüsseltes Vollbackup..."

# Daten komprimieren und direkt verschlüsseln
if tar --zstd -cf - "${SOURCE_DIRS[@]}" 2>> "${LOG_FILE}" \
    | gpg --batch --yes --encrypt --recipient "${RECIPIENT}" --output "${TARGET}"; then
    log "Backup erfolgreich erzeugt: ${TARGET} ($(du -h "${TARGET}" | cut -f1))"
    sha256sum "${TARGET}" > "${TARGET}.sha256"
else
    log "FEHLER: Backup-Vorgang fehlgeschlagen!" >&2
    exit 1
fi

# Lokale Rotation: Behalte die letzten 7 Tage
find "${BACKUP_DIR}" -name "backup_*.tar.zst.gpg*" -mtime +7 -delete
log "Bereinigung älterer Backups abgeschlossen."
```

3. Monitoring-Skript einrichten

```
#!/usr/bin/env bash
# /usr/local/sbin/backup_health_check.sh
set -euo pipefail

BACKUP_DIR="/srv/backups/daily"
MIN_BYTES=1048576 # 1 MB Mindestgröße

# Prüfe, ob in den letzten 25 Stunden ein valides Backup erstellt wurde
LATEST=$(find "${BACKUP_DIR}" -type f -name "backup_*.tar.zst.gpg" -mtime -1 | sort -r |
head -n 1)

if [[ -z "${LATEST}" ]]; then
    echo "[ERROR] Kein aktuelles Backup innerhalb der letzten 25 Stunden gefunden!" >&2
    exit 2
fi

SIZE=$(stat -c %s "${LATEST}")
if (( SIZE < MIN_BYTES )); then
    echo "[WARNING] Backup ${LATEST} ist auffällig klein (${SIZE} Bytes)!" >&2
    exit 1
fi

# Prüfsummenvalidierung
if ! sha256sum -c "${LATEST}.sha256" --status; then
    echo "[CRITICAL] Prüfsumme von ${LATEST} fehlerhaft – Datenintegrität beschädigt!" >&2
    exit 2
fi

echo "[OK] Backup ${LATEST} intakt (${SIZE} Bytes, Prüfsumme valide)."
exit 0
```

Befehlsreferenz (Cheatsheet)

Für den schnellen Zugriff bei der Planung und Umsetzung von Backup- und Recovery-Prozessen fasst die folgende Referenz die wichtigsten Linux-Kommandos und Werkzeuge zusammen:

Archiv mit Zstandard komprimieren Erstellt ein mehrkern-komprimiertes Archiv mit Zeitstempel

$ `tar --zstd -cf backup_$(date +%F).tar.zst /pfad/`

Zstandard-Archiv entpacken Entpackt ein .tar.zst-Archiv in ein definiertes Zielverzeichnis

$ `tar --zstd -xf backup.tar.zst -C /zielverzeichnis/`

Archivintegrität ohne Entpacken prüfen Listet alle enthaltenen Dateien auf und prüft die Archivstruktur

$ `tar --zstd -tvf backup.tar.zst > /dev/null`

Verzeichnisse über SSH synchronisierenÜberträgt Dateien im Archivmodus mit Erhalt aller Rechte und Symlinks

$ `rsync -avz -e ssh /quelle/ user@host:/ziel/`

Inkrementeller Hardlink-Snapshot Erstellt platzsparendes Backup mit Hardlinks auf unveränderte Vorversion

$ `rsync -a --delete --link-dest=/backup/gestern /data/ /backup/heute/`

Bitgenaues Datenträger-Image erstellen Schreibt 1:1 Block-Image einer Partition oder SSD mit Fortschrittsanzeige

$ `dd if=/dev/nvme0n1 of=/backup/disk.img bs=4M status=progress`

Verschlüsseltes Borg-Repository initialisieren Erstellt ein dedupliziertes Repository mit Repokey-Verschlüsselung

$ `borg init --encryption=repokey-blake2 /srv/backup-repo`

Dedupliziertes Borg-Backup anlegen Sichert Verzeichnisse mit variabler Chunk-Deduplizierung und Statistik

$ `borg create --stats --progress /srv/backup-repo::srv-{now:%F} /data`

Veraltete Borg-Generationen bereinigen (Pruning) Rotiert Archivstände automatisch nach täglichen und monatlichen Fristen

```
$ borg prune -v --keep-daily=7 --keep-weekly=4 --keep-monthly=12 /srv/backup-repo
```

Inkrementelles Backup mit Restic Sichert Verzeichnisse verschlüsselt in ein lokales oder Cloud-Repository

```
$ restic -r /srv/restic-repo backup /srv/data
```

Restic-Repository validieren Prüft Integrität von Metadaten und Chunks im Restic-Repository

```
$ restic -r /srv/restic-repo check --read-data-subset=10%
```

Btrfs-Snapshot schreibgeschützt erstellen Friert Subvolume in Millisekunden per Copy-on-Write schreibgeschützt ein

```
$ btrfs subvolume snapshot -r /data /data/.snapshots/backup_$(date +%F)
```

Btrfs-Snapshot differentiell übertragen Sendet Block-Unterschiede zweier Snapshots über SSH an Backup-Server

```
$ btrfs send -p /snap/gestern /snap/heute | ssh user@host "btrfs receive /srv/backup/"
```

Kryptografische Prüfsumme generieren Berechnet SHA-256 Prüfsumme zur Erkennung von Bitfäule und Transferfehlern

```
$ sha256sum datei.tar.zst > datei.sha256
```

Prüfsummen-Abgleich durchführen Validiert Dateien automatisiert gegen die gespeicherten Prüfsummen

```
$ sha256sum -c datei.sha256
```

MySQL/MariaDB Online-Dump (Hot-Backup) Erstellt transaktionssicheren Dump aller Datenbanken ohne Tabellensperren

```
$ mysqldump --all-databases --single-transaction --quick | zstd > dump.sql.zst
```

PostgreSQL Cluster-Dump Sichert alle Datenbanken, Rollen und Tabellenbereiche komprimiert

```
$ pg_dumpall | zstd > pg_all.sql.zst
```

Weiterführende Ressourcen

Die folgenden Leitfäden, Handbücher und internen Kursmodule vertiefen die behandelten Strategien zur Datensicherung und Systemverwaltung:

BorgBackup Documentation Offizielle Dokumentation zur deduplizierenden Archivierung mit BorgBackup

Restic Documentation Offizieller Leitfaden für sichere Cloud- und On-Premises-Backups mit Restic

GNU Tar Manual Vollständiges Referenzhandbuch des GNU-Tar-Archivierungswerkzeugs

Linux Administration #5: Shell-Scripting Das vorherige Modul: Shell-Scripting, Traps, Cron und systemd-Timer

Linux Administration #7: Systemsicherheit Das nachfolgende Modul: SSH-Härtung, Firewalls, Fail2ban und LUKS-Verschlüsselung

Befehlszeilenprozessor in Linux Fundamentales Wissen zu Shells, I/O-Streams, Pipes und Prozessumleitungen

Fazit

In diesem sechsten Teil unserer Linux-Administrations-Serie hast du gelernt, wie du ein zuverlässiges Backup-System aufbaust und verwaltest. Du verstehst nun die verschiedenen Backup-Strategien, von Vollbackups über inkrementelle und differentielle Sicherungen bis hin zu Hardlink-Snapshots und Deduplizierung, und kennst ihre jeweiligen Stärken und Herausforderungen.

Die Sicherung und Wiederherstellung von Daten gehört zu den verantwortungsvollsten Aufgaben im Alltag eines Linux-Administrators. Mit dem erworbenen Wissen kannst du eigenständig Backup-Konzepte entwickeln, automatisierte Backup-Pipelines einrichten und im Notfall Daten zielsicher wiederherstellen. Besonders wichtig ist das Verständnis für die verschiedenen Werkzeuge und deren Einsatzmöglichkeiten sowie die Fähigkeit, Backups kontinuierlich zu überwachen und Integritätsprüfungen durchzuführen.

Quelle: https://admindocs.de/de/linux-administration/linux-administration-datensicherung-und-wiederherstellung

Die praktischen Übungen haben dir gezeigt, wie du ein vollständiges Backup-System aufbaust, verschlüsselte Archive erstellst und mit Monitoring-Skripten absicherst. Eine sorgfältige Dokumentation und regelmäßige Tests stellen sicher, dass du im Ernstfall schnell und gelassen reagieren kannst.

> □□ **Praxis-Tipp:** Ein ungetestetes Backup ist kein Backup! Richte einen quartalsweisen Notfall-Test (*Disaster Recovery Drill*) ein, bei dem du ein Backup auf eine frische, isolierte Test-VM zurückspielst und verifizierst, dass alle Dienste und Datenbanken fehlerfrei anlaufen.

Im nächsten Modul unseres Kurses widmen wir uns der Systemhärtung und Abwehr von Cyber-Angriffen: □□ **Weiter geht es mit:** Linux Administration #7: Systemsicherheit und Härtung

□□ **Zur Kursübersicht:** Alle Linux-Administrations-Artikel & Module

Quelle: https://admindocs.de/de/linux-administration/linux-administration-datensicherung-und-wiederherstellung