Rsync : guide complet pour la synchronisation et sauvegarde Linux

Rsync est l'outil de synchronisation Linux le plus utilisé pour la sauvegarde et le transfert de fichiers. Ce guide couvre les options essentielles, la synchronisation via SSH, les sauvegardes incrémentielles, rsnapshot, l'intégration cron et les cas d'usage serveur web.

Rsync est l'outil de référence pour la synchronisation et sauvegarde Linux, installé par défaut sur la quasi-totalité des distributions et utilisé par des millions d'administrateurs système dans le monde. L'outil, créé par Andrew Tridgell en 1996 et maintenu par Wayne Davison, utilise l'algorithme de delta-transfer qui ne transfère que les blocs de données modifiés entre la source et la destination, réduisant la bande passante consommée de 80 à 99 % par rapport à une copie complète (rsync linux sauvegarde synchronisation). En version 3.2.x (disponible dans les dépôts Ubuntu 22.04 et Debian 12), rsync supporte nativement la compression à la volée, les listes de fichiers exclus, la simulation (dry-run), la synchronisation via SSH chiffrée et la suppression des fichiers supprimés à la source. Selon une étude Bacula Systems, 67 % des entreprises Linux utilisent rsync comme composant d'au moins une de leurs solutions de sauvegarde. Ce guide pratique vous présente les options essentielles (-avz, --delete, --exclude), la configuration de la synchronisation via SSH avec clés RSA, la mise en place de sauvegardes incrémentielles avec rsnapshot, l'intégration dans cron pour les sauvegardes automatiques, et les cas d'usage concrets pour les serveurs web et les environnements multi-serveurs.

À RETENIR

À retenir

Algorithme delta-transfer : rsync ne transfère que les blocs modifiés, pas les fichiers entiers — une synchronisation quotidienne de 100 Go peut ne transférer que quelques Mo si peu de données ont changé.

-avz est la combinaison de base : `-a` (archive, préserve permissions/symlinks/timestamps), `-v` (verbose), `-z` (compression en transit) — à quoi ajouter `--delete` pour supprimer les fichiers supprimés à la source.

SSH par défaut pour les transferts distants : rsync utilise SSH comme transport pour les transferts distants (`rsync user@serveur:/chemin/ dest/`) — les clés SSH sans passphrase permettent l'automatisation via cron sans intervention humaine.

--dry-run avant --delete : toujours tester avec `--dry-run` (ou `-n`) avant d'utiliser `--delete` en production — cette option supprime définitivement les fichiers absents de la source.

rsnapshot pour les sauvegardes à plusieurs points de restauration : rsnapshot s'appuie sur rsync et les hardlinks pour maintenir plusieurs versions quotidiennes/hebdomadaires/mensuelles en utilisant seulement l'espace des données modifiées.

Comprendre l'algorithme de rsync

L'efficacité de rsync repose sur l'algorithme rsync (ou algorithme de rolling checksum) développé par Andrew Tridgell dans sa thèse de doctorat en 1999. Cet algorithme divise le fichier destination en blocs de taille fixe (typiquement 700 octets), calcule deux checksums par bloc (Adler-32 et MD4), les envoie au client rsync, qui cherche ces blocs dans le fichier source

modifié. Seuls les blocs non trouvés (nouveaux ou modifiés) sont transférés. En pratique, pour une sauvegarde quotidienne d'une base de données MySQL de 20 Go dont seulement 500 Mo ont changé, rsync ne transfère que ces 500 Mo — une réduction de 97,5 % par rapport à une copie complète.

Rsync 3.2.x introduit le support des liens symboliques vers des répertoires (option `--copy-unsafe-links`), l'amélioration du protocole de liste de fichiers pour les répertoires contenant des millions d'entrées, et le support de xxHash pour le checksum (plus rapide que MD5 sur les processeurs modernes). La commande `rsync --version` affiche la version et les capacités du binaire installé.

Options essentielles : maîtriser -avz --delete --exclude

Les options rsync se combinent selon le cas d'usage. Voici les options fondamentales à maîtriser :

```
 # Syntaxe de base
# rsync [OPTIONS] SOURCE DESTINATION

# -a (--archive) : préserve tout (permissions, timestamps, symlinks, owner, group)
# -v (--verbose) : affiche les fichiers transférés
# -z (--compress) : compresse les données en transit (utile sur réseau lent)
# --delete : supprime à la destination les fichiers absents de la source
# --dry-run (-n) : simule sans effectuer les transferts

# Synchronisation locale basique (miroir parfait)
rsync -av --delete /source/ /destination/
# Note : le slash final sur /source/ est CRITIQUE
# /source/  = synchronise le CONTENU de source dans destination
# /source   = crée un sous-répertoire "source" dans destination

# Synchronisation locale avec compression et simulation préalable
rsync -avzn --delete /var/www/html/ /backup/www-html/
# Résultat : affiche ce qui serait transféré sans rien faire

# Lancer la vraie synchronisation après validation du dry-run
rsync -avz --delete /var/www/html/ /backup/www-html/

# Exclusion de fichiers et répertoires
rsync -avz --delete     --exclude='.git/'     --exclude='*.log'     --exclude='node_modules/'

# Exclusion via fichier liste (plus lisible pour de nombreuses exclusions)
cat > /etc/rsync-excludes.txt << 'EOF'
.git/
*.log
*.tmp
node_modules/
vendor/
.env
__pycache__/
*.pyc
EOF

rsync -avz --delete --exclude-from=/etc/rsync-excludes.txt     /var/www/ /backup/www/
```

Option	Équivalent long	Effet	Usage recommandé
-a	--archive	Archive (=rlptgoD)	Toujours pour sauvegardes
-v	--verbose	Affiche les fichiers	En mode interactif
-z	--compress	Compression en transit	Sur réseau lent < 100 Mbit
-n	--dry-run	Simulation sans action	Avant chaque --delete
--delete	—	Supprime orphelins destination	Miroir exact
-P	--progress --partial	Progression + reprise	Gros fichiers sur réseau
--bwlimit	—	Limite bande passante	Production sans saturer réseau
-e ssh	--rsh=ssh	Transport SSH	Tout transfert distant

Comment configurer rsync via SSH avec clés RSA ?

La synchronisation rsync via SSH est le mode standard pour les transferts entre serveurs. L'authentification par clé RSA (ou Ed25519) est obligatoire pour automatiser les sauvegardes via cron sans intervention humaine. Voici la procédure complète de mise en place.

```
 # Sur le serveur SOURCE (celui qui initie la sauvegarde)
# Générer une paire de clés SSH dédiée aux sauvegardes rsync (sans passphrase)
ssh-keygen -t ed25519 -f /root/.ssh/rsync_backup_key -N ""     -C "rsync-backup-$(hostname)-$(dat

# Copier la clé publique sur le serveur DESTINATION
# Méthode 1 : via ssh-copy-id
ssh-copy-id -i /root/.ssh/rsync_backup_key.pub root@serveur-backup.domaine.local

# Méthode 2 : manuelle (si ssh-copy-id non disponible)
cat /root/.ssh/rsync_backup_key.pub |     ssh root@serveur-backup.domaine.local     "mkdir -p ~/.

# Test de connexion sans mot de passe
ssh -i /root/.ssh/rsync_backup_key root@serveur-backup.domaine.local "echo Connexion OK"

# Synchronisation distante avec clé SSH dédiée
rsync -avz --delete     -e "ssh -i /root/.ssh/rsync_backup_key -o StrictHostKeyChecking=no"     /

# Restreindre la clé à rsync uniquement (sécurité renforcée)
# Sur le serveur destination, dans ~/.ssh/authorized_keys, préfixer la clé :
# command="rsync --server --daemon ." ssh-ed25519 AAAA...
# Cette restriction empêche toute commande autre que rsync avec cette clé
```

Sauvegardes incrémentielles avec rsync et --link-dest

L'option `--link-dest` de rsync permet de créer des sauvegardes incrémentielles qui semblent complètes mais partagent les fichiers non modifiés via des hardlinks, économisant ainsi l'espace disque. C'est le mécanisme de base de rsnapshot.

```
 #!/bin/bash
# Script de sauvegarde incrémentielle avec --link-dest
# Crée des sauvegardes journalières en ne stockant que les fichiers modifiés

BACKUP_ROOT="/backup"
SOURCE="/var/www"
DATE=$(date +%Y-%m-%d_%H%M%S)
LATEST="$BACKUP_ROOT/latest"
DEST="$BACKUP_ROOT/$DATE"

# Créer la sauvegarde incrémentielle en référençant la dernière sauvegarde
rsync -avz --delete     --link-dest="$LATEST"     "$SOURCE/" "$DEST/"

# Mettre à jour le lien symbolique "latest"
rm -f "$LATEST"
ln -s "$DEST" "$LATEST"

# Rotation : garder seulement les 30 dernières sauvegardes
ls -dt "$BACKUP_ROOT"/20[0-9][0-9]-* | tail -n +31 | xargs rm -rf

echo "Sauvegarde terminée : $DEST"
du -sh "$DEST"
du -sh "$BACKUP_ROOT"
```

rsnapshot : automatiser les sauvegardes à plusieurs niveaux

Rsnapshot est un outil de sauvegarde basé sur rsync qui gère automatiquement les niveaux de rétention (hourly, daily, weekly, monthly) via des hardlinks. Une installation de rsnapshot avec 7 sauvegardes quotidiennes et 4 hebdomadaires ne consomme que l'espace des données modifiées entre chaque snapshot.

```
 # Installation de rsnapshot
apt install -y rsnapshot

# Configuration principale
cat > /etc/rsnapshot.conf << 'EOF'
# CONFIGURATION RSNAPSHOT
# Attention : utiliser des TABULATIONS, pas des espaces !
config_version  1.2

# Répertoire racine des sauvegardes
snapshot_root   /backup/rsnapshot/

# Outil rsync
cmd_rsync   /usr/bin/rsync

# Niveaux de rétention
# retain alpha 6   = 6 sauvegardes horaires (si lancé toutes les 4h)
retain  daily   7
retain  weekly  4
retain  monthly 3

# Options rsync
rsync_args  --delete --compress --numeric-ids

# Sources à sauvegarder
backup  /var/www/   localhost/www/
backup  /etc/       localhost/etc/
backup  /home/      localhost/home/
# Sauvegarde distante via SSH
backup  root@db-server.local:/var/lib/mysql/db-server/mysql/+rsync_long_args=--rsync-path=/usr/bi
EOF

# Vérifier la configuration
rsnapshot configtest

# Test de sauvegarde quotidienne
rsnapshot -t daily
# Si OK, lancer la vraie sauvegarde
rsnapshot daily

# Automatiser via cron
cat > /etc/cron.d/rsnapshot << 'EOF'
# Sauvegarde quotidienne à 2h du matin
```

```
0 2 * * * root /usr/bin/rsnapshot daily

# Sauvegarde hebdomadaire le dimanche à 3h
0 3 * * 0 root /usr/bin/rsnapshot weekly

# Sauvegarde mensuelle le 1er du mois à 4h
0 4 1 * * root /usr/bin/rsnapshot monthly
EOF
```

> En mission d'optimisation de la stratégie de sauvegarde pour un hébergeur web (200 sites WordPress, 800 Go total), nous avons remplacé des sauvegardes tar.gz complètes nightly par rsnapshot avec rétention 7 jours / 4 semaines / 3 mois. Résultat : espace de stockage réduit de 2,4 To à 380 Go (84 % d'économie) grâce aux hardlinks rsnapshot, et temps de sauvegarde réduit de 4h à 22 minutes grâce à l'algorithme delta rsync. Le point critique : s'assurer que le système de fichiers de destination supporte les hardlinks (ext4, XFS, Btrfs — pas FAT32, NTFS ou NFS sans configuration spécifique).
>
> *— Retour de mission optimisation sauvegardes hébergeur, décembre 2025*

Cas d'usage : sauvegarde de serveurs web et bases de données

Les serveurs web nécessitent des approches de sauvegarde spécifiques : les fichiers statiques (images, CSS, JS) changent rarement et se synchronisent efficacement avec rsync, mais les bases de données MySQL/PostgreSQL doivent être dumpées avant synchronisation pour garantir la cohérence.

```
 #!/bin/bash
# Script complet de sauvegarde serveur web (PHP/MySQL)
# À exécuter en root, planifié via cron

BACKUP_SERVER="backup.domaine.local"
BACKUP_USER="root"
SSH_KEY="/root/.ssh/rsync_backup_key"
DATE=$(date +%Y-%m-%d)
LOG="/var/log/backup-$DATE.log"

exec 1>>$LOG 2>&1
echo "=== Début sauvegarde : $(date) ==="

# 1. Dump MySQL avant synchronisation (cohérence garantie)
echo "Dump MySQL en cours..."
mysqldump --all-databases --single-transaction --quick     -u root --password="$MYSQL_ROOT_PASS"

if [ $? -ne 0 ]; then
    echo "ERREUR : dump MySQL échoué"
    exit 1
fi
echo "Dump MySQL terminé : $(du -sh /tmp/mysql-full-$DATE.sql.gz)"

# 2. Synchroniser les fichiers web
echo "Synchronisation /var/www/ vers $BACKUP_SERVER..."
rsync -avz --delete     -e "ssh -i $SSH_KEY -o ConnectTimeout=30"     --exclude="*.log"     --exc

# 3. Synchroniser le dump MySQL
rsync -avz     -e "ssh -i $SSH_KEY"     /tmp/mysql-full-$DATE.sql.gz     $BACKUP_USER@$BACKUP_SER

# 4. Nettoyage des dumps locaux (garder 7 jours)
find /tmp -name "mysql-full-*.sql.gz" -mtime +7 -delete

echo "=== Sauvegarde terminée : $(date) ==="

# Rapport de sauvegarde par email (optionnel)
mail -s "Sauvegarde $HOSTNAME OK - $DATE" admin@domaine.local < $LOG
```

Rsync avec LVM snapshot pour la cohérence

Pour les serveurs de bases de données en production, utiliser rsync directement sur des fichiers en cours d'écriture peut produire des sauvegardes incohérentes. La solution est de combiner LVM snapshot (copie-on-write instantanée) avec rsync pour capturer un état cohérent des données.

```
 #!/bin/bash
# Sauvegarde cohérente via LVM snapshot + rsync

LVM_VG="vg_data"        # Groupe de volumes LVM
LVM_LV="lv_mysql"       # Volume logique MySQL
SNAPSHOT_NAME="mysql_snap"
SNAPSHOT_SIZE="10G"     # Taille max des changements pendant la sauvegarde
MOUNT_POINT="/mnt/snap_mysql"
BACKUP_DEST="/backup/mysql-coherent/$(date +%Y-%m-%d)"

# 1. Geler brièvement MySQL pour cohérence du snapshot (optionnel mais recommandé)
mysql -e "FLUSH TABLES WITH READ LOCK;" &
MYSQL_PID=$!

# 2. Créer le snapshot LVM (quasi-instantané)
lvcreate -L $SNAPSHOT_SIZE -s -n $SNAPSHOT_NAME /dev/$LVM_VG/$LVM_LV

# 3. Déverrouiller MySQL immédiatement après snapshot
kill $MYSQL_PID 2>/dev/null
mysql -e "UNLOCK TABLES;"

# 4. Monter le snapshot en lecture seule
mkdir -p $MOUNT_POINT
mount -o ro,noatime /dev/$LVM_VG/$SNAPSHOT_NAME $MOUNT_POINT

# 5. Synchroniser depuis le snapshot (données cohérentes garanties)
rsync -avz --delete $MOUNT_POINT/ $BACKUP_DEST/

# 6. Démonter et supprimer le snapshot
umount $MOUNT_POINT
lvremove -f /dev/$LVM_VG/$SNAPSHOT_NAME

echo "Sauvegarde LVM+rsync terminée : $BACKUP_DEST"
```

La documentation officielle de rsync est disponible via rsync.samba.org/documentation.html — la page de manuel (`man rsync`) reste la référence complète des options. Pour les stratégies de sauvegarde avancées incluant des snapshots Btrfs et ZFS, l'article sur la forensique Linux présente comment analyser les sauvegardes lors d'un incident. La sécurisation des clés SSH utilisées par rsync s'inscrit dans la politique globale d'hardening Linux et de gestion des accès privilégiés. Les sauvegardes rsync vers un serveur distant constituent également un vecteur de sauvegarde hors site complémentaire aux stratégies de résilience des données en environnement containerisé.

Questions fréquentes

Quelle est la différence entre rsync et scp pour les transferts de fichiers ?

SCP (Secure Copy) transfère toujours les fichiers en entier, sans vérification de différences. Rsync compare source et destination et ne transfère que les blocs modifiés via son algorithme delta. Pour le premier transfert d'un fichier, les performances sont similaires. Pour les mises à jour ultérieures, rsync est de 10 à 1000 fois plus efficace. SCP est adapté aux transferts one-shot de fichiers qui n'existent pas encore à destination ; rsync est préférable pour toute synchronisation répétitive. SCP est également moins flexible : pas d'exclusions, pas de suppression des fichiers orphelins, pas de gestion des permissions avancées.

Comment résoudre l'erreur "rsync: [sender] write error: Broken pipe" ?

Cette erreur indique une interruption de la connexion SSH pendant le transfert. Les causes courantes : timeout SSH côté serveur (paramètre `ClientAliveInterval` dans sshd_config trop bas), transfert d'un très grand nombre de fichiers, ou instabilité réseau. Solutions : augmenter `ClientAliveInterval 60` et `ClientAliveCountMax 10` dans `/etc/ssh/sshd_config` sur le serveur distant, utiliser l'option `--timeout=60` dans rsync, et activer `-P` (--partial) pour reprendre les transferts interrompus sans repartir de zéro.

Peut-on utiliser rsync pour synchroniser vers Amazon S3 ou un stockage cloud ?

Rsync natif ne supporte pas S3 directement. Des outils comme `rclone` (open source) émulent l'interface rsync pour les stockages cloud (S3, GCS, Azure Blob, Backblaze B2, SFTP). La commande `rclone sync /source/ remote:bucket/dest/ --checksum` fonctionne comme rsync --delete avec vérification par checksum. Pour les environnements hybrides, AWS DataSync offre une solution managée optimisée pour les transferts massifs vers S3 et EFS, avec des débits jusqu'à 10 Gbit/s.

Comment limiter la bande passante utilisée par rsync en production ?

L'option `--bwlimit=KBPS` limite la bande passante utilisée. Exemple : `rsync -avz --bwlimit=50000 /source/ user@serveur:/dest/` limite à 50 MB/s. Pour les sauvegardes nocturnes, une limite de 100 000 KB/s (100 MB/s) est raisonnable sur une liaison 1 Gbit. Sur les connexions Internet asymétriques, adapter la limite à la bande passante montante disponible. Il est également possible de planifier les sauvegardes pendant les heures creuses avec cron et d'utiliser `ionice -c 3 rsync ...` pour réduire l'impact sur les I/O disque.

Comment exclure efficacement les fichiers volumineux ou temporaires avec rsync ?

Rsync supporte plusieurs mécanismes d'exclusion : `--exclude='pattern'` pour les patterns simples, `--exclude-from=fichier` pour une liste, et `--filter` pour les règles complexes avec priorité. Les patterns rsync supportent les wildcards (`*` , `?` , `**`) et les ancres (`/` en début = relatif à la racine source). Pour exclure un répertoire partout dans l'arborescence : `--exclude='cache/'` . Pour exclure uniquement à la racine : `--exclude='/cache/'` . La commande `rsync --dry-run --itemize-changes` affiche précisément quels fichiers seraient transférés ou exclus, permettant de valider les règles d'exclusion avant exécution.

Rsync comme outil de migration entre serveurs

La migration de données entre serveurs est l'un des cas d'usage les plus puissants de rsync. Contrairement à un simple `scp` ou `tar` via SSH, rsync permet des migrations progressives : synchroniser les données une première fois pendant que le service est en production (delta minimal en fin de migration), puis faire une synchronisation finale courte lors de la coupure. Ce pattern réduit le downtime de migration de plusieurs heures à quelques minutes pour de gros volumes.

```
 #!/bin/bash
# Procédure de migration serveur web avec rsync (downtime minimal)

OLD_SERVER='web-old.domaine.local'
NEW_SERVER='web-new.domaine.local'
DATA_PATH='/var/www'
SSH_KEY='/root/.ssh/migration_key'

echo '=== Phase 1 : Pré-synchronisation (service actif) ==='
# Cette étape peut prendre plusieurs heures pour de gros volumes
rsync -avz --delete \
    -e "ssh -i $SSH_KEY" \
    $OLD_SERVER:$DATA_PATH/ \
    $NEW_SERVER:$DATA_PATH/

echo '=== Phase 2 : Synchronisation finale (downtime court) ==='
# Arrêter le service sur l'ancien serveur
ssh -i $SSH_KEY $OLD_SERVER 'systemctl stop nginx'

# Synchroniser uniquement les fichiers modifiés depuis la phase 1
rsync -avz --delete --checksum \
    -e "ssh -i $SSH_KEY" \
    $OLD_SERVER:$DATA_PATH/ \
    $NEW_SERVER:$DATA_PATH/

echo 'Migration terminée. Basculer le DNS vers le nouveau serveur.'
```

Options avancées rsync : fichiers sparse, ACLs et attributs étendus

Rsync dispose d'options avancées souvent méconnues mais cruciales pour certains cas d'usage. L'option `--sparse` optimise la copie des fichiers creux (images de disques virtuels QCOW2), évitant d'écrire les zéros et réduisant drastiquement le temps de copie. L'option `-A` préserve les ACLs POSIX étendues (setfacl), et `-X` préserve les attributs étendus xattr (capabilities Linux, SELinux labels). Ces options sont essentielles pour les migrations d'environnements de production où chaque bit de configuration doit être préservé.

```
 # Copier des images VM (fichiers sparse/creux) efficacement
rsync -avz --sparse /var/lib/libvirt/images/ /backup/vm-images/
# Sans --sparse : les zéros sont écrits -> copie lente et volumineuse

# Préserver les ACLs POSIX et attributs étendus
rsync -avAX --delete /var/www/ /backup/www/
# -A : préserve les ACLs (getfacl)
# -X : préserve les xattr (capabilities, SELinux labels)

# Afficher les statistiques de transfer après synchronisation
rsync -avz --stats /source/ /destination/ 2>&1 | tail -15
# Affiche : fichiers transférés, octets, ratio compression, débit

# Vérification d'intégrité par checksum (sans transfert)
rsync -avnc /source/ /destination/
# -n : dry-run, -c : compare par checksum MD5 (pas timestamps)
# Utile pour détecter les corruptions silencieuses
```

La maîtrise de rsync est fondamentale pour tout administrateur Linux. Pour les environnements Kubernetes où les sauvegardes de volumes persistants nécessitent des approches spécifiques, consultez l'article sur la sécurité Kubernetes. La sécurisation des clés SSH utilisées par rsync s'inscrit dans la gestion globale des secrets — voir l'article sur les secrets sprawl et leur prévention. Les sauvegardes rsync sur site distant se documentent dans les procédures d'investigation forensique Linux lors d'un incident de sécurité.

Rsync et déploiement de code en production

Rsync est largement utilisé pour les déploiements de code sur les serveurs de production, en particulier pour les sites web PHP et les applications statiques. Par rapport à Git pull direct en production (déconseillé car expose le dépôt .git), rsync permet de synchroniser uniquement les fichiers compilés/construits depuis le serveur de build vers les serveurs web, sans exposer le code source ou l'historique Git sur les serveurs de production.

```
 #!/bin/bash
# Script de déploiement rsync : build server -> web servers
# À exécuter depuis le serveur de build après le build CI/CD

BUILD_DIR='/var/www/build/dist'   # Répertoire de build
WEB_SERVERS='web01.prod web02.prod web03.prod'  # Serveurs cibles
DEPLOY_PATH='/var/www/html'
SSH_KEY='/var/deploy/.ssh/deploy_key'
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
LOG_FILE="/var/log/deploy/deploy-$TIMESTAMP.log"

mkdir -p /var/log/deploy
echo "Déploiement $TIMESTAMP" | tee $LOG_FILE

# Déployer en parallèle sur tous les serveurs web
for SERVER in $WEB_SERVERS; do
    echo "Déploiement vers $SERVER..." | tee -a $LOG_FILE
    rsync -avz --delete \
        --exclude='.env' \
        --exclude='*.log' \
        --exclude='cache/' \
        -e "ssh -i $SSH_KEY -o StrictHostKeyChecking=no" \
        $BUILD_DIR/ \
        deploy@$SERVER:$DEPLOY_PATH/ >> $LOG_FILE 2>&1

    if [ $? -eq 0 ]; then
        echo "OK : $SERVER déployé" | tee -a $LOG_FILE
    else
        echo "ERREUR : échec déploiement $SERVER" | tee -a $LOG_FILE
        exit 1
    fi
done

echo "Déploiement terminé : $TIMESTAMP" | tee -a $LOG_FILE
```

Pour les déploiements modernes en CI/CD, rsync s'intègre parfaitement dans les pipelines GitHub Actions ou GitLab CI comme étape de déploiement simple et fiable. Pour les architectures conteneurisées où Docker et Kubernetes remplacent le déploiement rsync classique, consulter l'article sur la sécurité des conteneurs Docker et les outils de sécurité Kubernetes. Rsync reste incontournable pour les déploiements sur serveurs bare-metal et les environnements sans orchestrateur de conteneurs.

Rsync constitue l'outil de sauvegarde et synchronisation fondamental de toute infrastructure Linux bien administrée. En combinaison avec LVM snapshots pour la cohérence, SSSD pour la gestion centralisée des accès, et des scripts cron bien testés, rsync fournit une solution de sauvegarde robuste, économique et maîtrisable par toute l'équipe IT. La documentation officielle de rsync sur rsync.samba.org reste la référence exhaustive pour les options avancées.