hash_tool

Vérification d'intégrité de fichiers par hachage BLAKE3

Version 0.12 — Février 2026
Shell · Docker · Linux / macOS / WSL

En une phrase

hash_tool détecte la corruption silencieuse de fichiers en comparant des **empreintes BLAKE3** prises à des instants différents. Il fonctionne depuis la ligne de commande, ne requiert aucune installation complexe et produit un rapport HTML lisible sans outil supplémentaire.

Table des matières

1. Présentation générale

1.1 Problème adressé

Les fichiers se corrompent silencieusement. Erreurs de secteurs sur disque dur, coupures réseau pendant un transfert, bitrot sur supports optiques ou magnétiques, manipulation accidentelle — dans tous ces cas, les fichiers restent présents, affichent la bonne taille, mais leur contenu a changé. Aucun outil système standard ne détecte ce type de corruption de façon proactive.

hash_tool répond à ce problème par une approche simple : prendre une **empreinte cryptographique** de chaque fichier au moment où l'on sait que les données sont saines, puis comparer cette empreinte à intervalles réguliers ou après toute opération risquée (transfert, sauvegarde, restauration, migration).

1.2 Algorithme : pourquoi BLAKE3

Critère	MD5	SHA-256	BLAKE3
Vitesse sur fichiers volumineux	Rapide	Lent	Très rapide (x3 SHA-256)
Sécurité cryptographique	Cassé	Solide	Solide
Parallélisation	Non	Non	Oui (SIMD, multi-thread)
Adapté à l'intégrité de fichiers	Oui*	Oui	Oui (référence actuelle)

* MD5 est suffisant pour détecter la corruption accidentelle mais ne doit plus être utilisé pour des usages sécuritaires.

1.3 Ce que l'outil fait

- compute — Calcule les empreintes BLAKE3 de tous les fichiers d'un dossier et les stocke dans un fichier .b3
- verify — Relit les fichiers et compare leurs empreintes au fichier .b3 de référence ; signale toute divergence
- compare — Compare deux fichiers .b3 (snapshots) et identifie les fichiers modifiés, disparus ou nouveaux
- runner — Exécute un pipeline de plusieurs opérations décrit dans un fichier JSON (compute + verify + compare en séquence)

Chaque opération produit un dossier de résultats horodaté contenant un rapport texte (`recap.txt`), les listes de fichiers problématiques, et un **rapport HTML autonome** lisible hors connexion.

1.4 Ce que l'outil ne fait pas

- Il ne chiffre pas les données et ne les déplace pas
- Il ne surveille pas en temps réel (pas de daemon, pas d'inotify)

- Il ne gère pas de base de données centralisée ni d'historique au-delà des fichiers .b3
- Il ne remplace pas un logiciel de sauvegarde

2. Fonctionnalités détaillées

2.1 Calcul d'empreintes (compute)

Parcourt récursivement un dossier, calcule une empreinte BLAKE3 par fichier, stocke le résultat dans un fichier .b3 avec des chemins relatifs. La progression et l'ETA sont affichés en temps réel sans polluer le fichier de sortie.

> ¡ **Chemins relatifs**
> Le fichier .b3 utilise des chemins relatifs au dossier source. Il peut être déplacé et utilisé sur n'importe quelle machine tant que la structure de dossier est identique.

2.2 Vérification d'intégrité (verify)

Relit chaque fichier référencé dans le .b3, recalcule son empreinte et la compare. Trois états de sortie : **OK** (aucune divergence), **ECHEC** (au moins un fichier corrompu ou manquant), **ERREUR** (problème d'exécution). L'exit code est propagé, ce qui permet l'intégration dans des scripts ou pipelines CI.

2.3 Comparaison de snapshots (compare)

Compare deux fichiers .b3 produits à des instants différents sur le même dossier ou sur deux dossiers structurellement identiques. Produit trois listes distinctes :

- modifies.b3 — fichiers présents dans les deux snapshots mais dont l'empreinte a changé
- disparus.txt — fichiers présents dans le snapshot A absents du snapshot B
- nouveaux.txt — fichiers présents dans le snapshot B absents du snapshot A

Un **rapport HTML** autonome est généré automatiquement : il affiche les compteurs par catégorie, un badge de statut global (IDENTIQUES / DIFFÉRENCES DÉTECTÉES) et les listes de fichiers par section. Il fonctionne hors connexion, sans serveur ni dépendance externe.

2.4 Pipeline JSON (runner)

runner.sh lit un fichier `pipeline.json` et exécute les blocs compute / verify / compare en séquence. Chaque bloc est isolé dans un sous-shell — un échec arrête le pipeline avec un message explicite indiquant le numéro et le type du bloc fautif.

Le champ optionnel `"resultats"` sur un bloc compare permet de rediriger les résultats vers un dossier spécifique, indépendamment du répertoire global de résultats.

2.5 Mode silencieux (--quiet)

L'option `--quiet` supprime toute sortie terminal. Les résultats sont écrits dans les fichiers habituels. L'exit code est propagé. Conçu pour une intégration dans des crons, des hooks Git ou des pipelines CI.

3. Architecture technique

3.1 Dépendances

Dépendance	Usage	Disponibilité
bash >= 4	Interpréteur shell	Linux, macOS (via brew), WSL
b3sum	Calcul des empreintes BLAKE3	Packages Alpine, Debian, Homebrew
jq	Parsing pipeline.json	Packages universels
coreutils	sort, comm, join, awk	Inclus dans toutes les distributions
find, stat	Parcours de dossiers	POSIX standard

3.2 Structure du projet

Fichier / Dossier	Rôle
src/integrity.sh	Logique métier principale (compute / verify / compare)
src/lib/report.sh	Génération des rapports HTML
runner.sh	Exécuteur de pipeline JSON
pipelines/pipeline.json	Exemple de pipeline (compute + verify + compare)
Dockerfile	Image Alpine ~14 Mo (b3sum + jq via apk)
docker-compose.yml	Orchestration 3 services (integrity / pipeline / cron)
tests/run_tests.sh	15 tests unitaires (T00–T14)
tests/run_tests_pipeline.sh	13 tests pipeline (TP01–TP12b)

3.3 Docker

L'image Docker est basée sur **Alpine 3.19** et pèse environ 14 Mo. Elle embarque bash, jq et b3sum (installés via apk). Elle supporte les architectures amd64 et arm64 (NAS Synology, Raspberry Pi). Quatre volumes conventionnels sont définis : `/data`, `/bases`, `/pipelines`, `/resultats`.

¡ **Aucune installation requise avec Docker**

docker run --rm -v /mes/donnees:/data:ro -v /mes/bases:/bases hash_tool compute /data /bases/hashes.b3

Cette seule commande suffit sur n'importe quelle machine disposant de Docker.

4. Utilisation rapide

4.1 Installation native

- Dépendances : `b3sum` + `jq` + bash >= 4
- Cloner le dépôt et rendre les scripts exécutables : `chmod +x src/integrity.sh runner.sh`
- Aucune compilation, aucune dépendance système au-delà des outils standard

4.2 Workflow typique

Étape	Commande	Moment
1. Indexer	./src/integrity.sh compute ./dossier bases/hashes_2024-01-15.b3	Données saines connues
2. Vérifier	./src/integrity.sh verify bases/hashes_2024-01-15.b3	Après transfert / stockage
3. Comparer	./src/integrity.sh compare bases/avant.b3 bases/apres.b3	Entre deux états
4. Pipeline	./runner.sh pipelines/pipeline.json	Automatisation multi-étapes

4.3 Environnements supportés

- Linux (Debian, Ubuntu, Alpine, Arch…)
- macOS (avec bash >= 4 via Homebrew)
- Windows via WSL2
- NAS Synology via Docker (image arm64)
- Serveur headless via cron en mode --quiet

5. Tests et fiabilité

5.1 Couverture

Suite	Cas	Périmètre
run_tests.sh	T00–T14 (15 cas)	compute, verify, compare, chemins, corruption, résultats, rapport HTML
run_tests_pipeline.sh	TP01–TP12b (13 cas)	JSON invalide, champs manquants, opérations inconnues, isolation sous-shell, champ resultats

5.2 Philosophie de test

- Chaque test crée son propre répertoire temporaire isolé dans /tmp
- Les tests de corruption introduisent volontairement une modification puis vérifient la détection
- Les tests pipeline vérifient l'isolation des sous-shells (pas de fuite de répertoire courant)
- Résultat coloré PASS / FAIL avec compteur final

6. Licence et contribution

hash_tool est distribué sous licence MIT. Contributions bienvenues via pull request. Les issues GitHub sont le canal privilégié pour signaler des bugs ou proposer des fonctionnalités.

¡ **Contributions prioritaires**

Rapport HTML : enrichissement du contenu et de la mise en page
install.sh : script d'installation one-liner avec vérification des dépendances
GitHub Actions : CI automatique sur push (run_tests.sh + run_tests_pipeline.sh)
--format json : sortie machine-readable pour les opérations verify et compare