Engineer

Bash

>_ le shell GNU pour lequel la plupart des scripts sont écrits

Ce que c’est

Bash, Bourne-Again SHell, distribué sous le nom GNU Bash et lancé par la commande `bash`, est un programme qui lit des commandes saisies à l’invite ou enregistrées dans un fichier et les exécute. Il démarre d’autres programmes, relie leurs entrées et sorties et ajoute variables, boucles, tests et fonctions pour qu’une suite de commandes puisse prendre des décisions. Le nom est un jeu de mots sur Stephen Bourne, dont `sh` était livré avec la Seventh Edition de Research Unix chez Bell Labs. Le manuel décrit Bash comme largement compatible avec ce shell, enrichi de fonctions de Korn shell et C shell, et conçu pour respecter la partie shell de POSIX. La version actuelle est 5.3, et le manuel 5.3 date de mai 2025.

Il est facile de confondre un shell avec la fenêtre qui l’héberge : le terminal dessine les caractères et transmet les frappes, tandis que le shell est le programme qui les interprète de l’autre côté. Bash fonctionne interactivement, avec invite, historique et complétion par tabulation, ou sans interaction, en lisant seul un script du début à la fin. C’est dans ce second usage que se trouvent la plupart de ses pièges.

Bash est l’un des programmes capables de remplir ce rôle. POSIX définit un langage shell portable, `sh`, et le programme installé comme `/bin/sh` varie selon le système. Sur Debian et Ubuntu, il s’agit de dash, petit shell POSIX descendant d’Almquist. Debian Policy cite bash, dash et ksh comme les shells les plus courants à y utiliser. Zsh est le shell de connexion préconfiguré pour les nouveaux comptes macOS depuis la version 10.15, bien que le Mac conserve `/bin/bash`. Un script choisit son interpréteur par sa première ligne, le shebang : `#!/bin/sh` demande le shell POSIX du système, tandis que `#!/bin/bash` ou `#!/usr/bin/env bash` demande explicitement Bash.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que Bash ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. L'entretien quotidien des serveurs, de la messagerie, des comptes et des mises à jour, pour que les outils indispensables restent sans histoire.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 10 serveurs PostgreSQL et MS SQL sur Ubuntu — 10 serveurs de bases de données déployés et maintenus

10 serveurs PostgreSQL et MS SQL sur Ubuntu Contactez-nous →

Pourquoi c’est important

Bash est le langage qui relie les outils des serveurs Unix. Il est présent sur presque toute machine Linux, et les commandes saisies par un ingénieur sont déjà du Bash. En enregistrer quelques-unes dans un fichier donne donc un script utilisant ce que la machine possède. Pour enchaîner des programmes existants — exporter une base, compresser l’export et le copier ailleurs — peu de langages sont aussi brefs, car tubes, redirections et statuts de sortie constituent son vocabulaire natif.

Ces mêmes caractéristiques deviennent coûteuses lorsque le script grandit. Toutes les valeurs sont des chaînes et l’arithmétique ne fonctionne que sur les entiers : les structures comme JSON, les dates ou les montants décimaux doivent passer par un autre outil. Les expansions sans guillemets sont découpées en mots puis comparées aux noms de fichiers. Une valeur contenant un espace ou un astérisque change donc le sens de la commande qui la reçoit. La gestion des erreurs doit être activée : sans activation explicite, un script continue après l’échec d’une commande, et les options `set -e` et `pipefail` comportent des exceptions que le manuel met plusieurs paragraphes à énumérer. Une règle pratique consiste à considérer qu’un script nécessitant structures de données, analyseur ou suite de tests est devenu un programme, moins cher à maintenir dans un langage comme Python.

La portabilité est elle aussi une décision. Debian Policy exige que les scripts `/bin/sh` respectent POSIX et une courte liste d’extensions. Tout script ayant besoin de davantage doit nommer son shell sur la première ligne, par exemple `#!/bin/bash`. Elle suggère `checkbashisms` ou une exécution sous un shell plus strict comme posh pour trouver les infractions, et conseille `/bin/bash` en cas de doute. La norme évolue lentement : Bash propose `pipefail` depuis la version 3.0, mais POSIX ne l’a ajouté que dans son édition 2024.

Un shell transforme le texte en commandes, ce qui en fait une frontière de sécurité dès que du texte extérieur l'atteint. Shellshock, enregistré comme CVE-2014-6271 en septembre 2014, a montré l'étendue possible de cette frontière. Jusqu'à la version 4.3, Bash exécutait des commandes suivant une définition de fonction dans une variable d'environnement. Tout programme copiant du texte contrôlé par un attaquant dans cet environnement avant de lancer Bash — notamment CGI sous Apache, ForceCommand d'OpenSSH et certains clients DHCP — devenait donc une voie d'exécution sur la machine. La première correction était incomplète et le reste a reçu l'identifiant CVE-2014-7169. La leçon quotidienne est plus limitée : le texte extérieur atteint une commande comme argument, jamais comme partie d'une chaîne passée à `eval` ou `sh -c`.

À quoi cela ressemble en pratique

Dans les travaux de l'entreprise, Bash conserve le rôle que lui attribue la page des technologies : Python pour les données, l'automatisation et les services, Bash pour relier les outils, écrit pour être lu.

Dans les six pipelines ETL et ELT alimentant Google BigQuery depuis un entrepôt derrière un VPN IPsec, les archives téléchargées contenaient des fichiers Parquet, CSV et .bak. Des scripts Bash et Python les décompressaient et classaient chaque fichier selon son type et la disposition de ses colonnes. bcp, psql et SSIS les chargeaient ensuite dans des schémas intermédiaires, puis gzip, Google Cloud Storage et l'outil bq transportaient les résultats vers BigQuery. Une fois décompression et classement automatisés, une exécution qui prenait au moins trois heures à la main s'est achevée en moins de 20 minutes. Les données pouvaient être synchronisées chaque jour plutôt qu'une fois par semaine.

Les cent sauvegardes critiques automatisées montrent le même partage du travail. Barman gérait PostgreSQL, Google Cloud conservait les copies hors site et Bash et Python orchestraient et validaient l'ensemble. Les politiques de conservation maintenaient les copies à jour, et les contrôles d'intégrité confirmaient que chacune était valide plutôt que simplement présente. C'est le travail auquel Bash convient : une suite d'outils existants où la valeur du script réside dans la vérification du résultat de chaque étape.

Où les problèmes surviennent

Un nom de fichier contenant un espace devient deux arguments. Un `$file` sans guillemets est découpé sur les caractères d'espacement, puis développé comme motif de fichiers. Ainsi, `report 2024.csv` atteint `rm` comme deux noms, et un `*` égaré correspond à tout un répertoire. Placez chaque expansion entre guillemets, comme `"$file"`, transmettez les listes d'arguments par `"$@"`, conservez les listes de noms dans des tableaux et laissez ShellCheck trouver le reste.

Un pipeline de sauvegarde annonce un succès après l'échec de l'export. Le statut de sortie d'un pipeline est celui de sa dernière commande. Dans `pg_dump app | gzip > app.sql.gz`, gzip peut donc réussir sur le peu reçu alors que l'export a échoué. Activez `set -o pipefail`, qui fait retourner au pipeline le dernier statut non nul, et vérifiez le résultat avec `gzip -t` ou, mieux, une restauration de test.

Un script utilisant `set -e` continue après une commande échouée. L'option est ignorée dans les tests d'un `if` ou d'un `while`, dans toutes les parties d'une liste `&&` ou `||` sauf la dernière, ainsi que dans toute fonction appelée depuis ces endroits. L'export ci-dessous peut donc échouer, et le script afficher quand même son succès :

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

backup() {
  pg_dump app > /var/backups/app.sql
  gzip -f /var/backups/app.sql
}

if backup; then # set -e is ignored inside backup here
  echo "backup done"
fi
```

Un piège voisin est `local out=$(cmd)`, qui retourne le statut de `local` et cache l'échec de `cmd`. Dans les fonctions, faites suivre chaque commande importante de `|| return 1`, et déclarez une variable avant de lui affecter une valeur sur une ligne distincte.

Un script fonctionnel sur une machine échoue sur une autre. Un fichier commençant par `#!/bin/sh`, mais utilisant des tests `[[` ou des tableaux, s'exécute sous Bash sur un système et sous dash sur l'autre. `[[` y est inconnu : un `if` fondé dessus affiche une erreur et suit sa branche else. Une affectation de tableau provoque une erreur de syntaxe qui termine le script. Sur Mac, `/bin/bash` annonce 3.2.57, avant les tableaux associatifs et `mapfile` arrivés en 4.0 : `declare -A` y échoue pour option invalide. Nommez dans le shebang le shell réellement nécessaire et testez la version la plus ancienne à prendre en charge.

Une ligne de nettoyage supprime bien plus que prévu. Si `build_dir` n'est pas définie, `rm -rf "$build_dir"/*` devient `rm -rf /*`. `set -u` transforme une variable absente en erreur, et `"${build_dir:?}"` arrête un script non interactif lorsque la variable est absente ou contient une chaîne vide.

Pour aller plus loin

- GNU Bash Reference Manual — manuel 5.3 : expansions, guillemets, commande interne `set` et cas précis où `set -e` est ignoré.
- Shell Command Language — langage portable `sh` de POSIX.1-2024, limite de tout script `#!/bin/sh`.
- Debian Policy : scripts — prise en charge requise par `/bin/sh` sur Debian, choix explicite de Bash et conseils sur `set -e`.
- Bash NEWS — nouveautés version par version maintenues par l'auteur, pour vérifier la version requise par une construction.
- CVE-2014-6271 — fiche NVD de Shellshock avec ses voies d'attaque et l'identifiant complémentaire.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que Bash ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/bash/